

**ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ИРКУТСКОЙ ОБЛАСТИ  
«ЧЕРЕМХОВСКИЙ ГОРНОТЕХНИЧЕСКИЙ КОЛЛЕДЖ  
ИМ. М.И. ЩАДОВА»**

**РАССМОТРЕНО**

на заседании ЦК  
«Информатики и ВТ»  
Протокол № 6  
«04» февраля 2025 г.  
Председатель: Коровина Н.С.

**УТВЕРЖДАЮ**

Зам. директора  
О.В. Папанова  
« 26» мая 2025 г.

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ**

по практическим занятиям студентов  
профессионального модуля

*ПМ. 02 Осуществление интеграции профессионального модуля  
09.02.07 Информационные системы и программирование*

Разработал:  
Литвинцева Е.А.

2025 г.

## СОДЕРЖАНИЕ

	<b>СТР.</b>
1. ПОЯСНИТЕЛЬНАЯ ЗАПИСКА	3
2. ПЕРЕЧЕНЬ ПРАКТИЧЕСКИХ ЗАНЯТИЙ	4
3. СОДЕРЖАНИЕ ПРАКТИЧЕСКИХ ЗАНЯТИЙ	7
4. ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ	174
ЛИСТ ИЗМЕНЕНИЙ И ДОПОЛНЕНИЙ, ВНЕСЁННЫХ В МЕТОДИЧЕСКИЕ УКАЗАНИЯ	176

## 1. ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Методические указания по практическим занятиям профессионального модуля ПМ.02 «**Осуществление интеграции программного модуля**» составлены в соответствии с учебным планом и рабочей программы дисциплины по специальности **09.02.07 Информационные системы и программирование**

Цель проведения практических занятий: формирование практических умений, необходимых в последующей профессиональной и учебной деятельности.

Методические указания практических занятий являются частью учебно-методического комплекса по учебной дисциплине и содержат:

- тему занятия (согласно тематическому плану учебной дисциплины);
- цель;
- оборудование (материалы, программное обеспечение, оснащение, раздаточный материал и др.);
- методические указания (изучить краткий теоретический материал по теме практического занятия);
- ход выполнения;
- форму отчета.

В результате выполнения полного объема заданий практических занятий студент должен **знать**:

- модели процесса разработки программного обеспечения;
- основные принципы процесса разработки программного обеспечения;
- основные подходы к интегрированию программных модулей;
- основы верификации и аттестации программного обеспечения
- современные технологии разработки программного обеспечения, инструментальных средства разработки
- применение методов математического моделирования в современном мире

должен **уметь**:

- использовать выбранную систему контроля версий;
- использовать методы для получения кода с заданной функциональностью и степенью качества
- использовать современные технологии разработки программного обеспечения, инструментальные средства разработки программного обеспечения при решении ситуационных задач
- применять методы математического моделирования при решении задач

При проведении практических работ применяются следующие технологии и методы обучения: чтение с маркировкой, «фишбон», информационные технологии, ментальные карты и т.д.

### **Оценка выполнения практических занятий**

**«Отлично»** - теоретическое содержание курса освоено полностью, без пробелов, умения сформированы, все предусмотренные программой учебные задания выполнены, качество их выполнения оценено высоко.

**«Хорошо»** - теоретическое содержание курса освоено полностью, без пробелов, некоторые умения сформированы недостаточно, все предусмотренные программой учебные задания выполнены, некоторые виды заданий выполнены с ошибками.

**«Удовлетворительно»** - теоретическое содержание курса освоено частично, но пробелы не носят существенного характера, необходимые умения работы с освоенным материалом в основном сформированы, большинство предусмотренных

программой обучения учебных заданий выполнено, некоторые из выполненных заданий содержат ошибки.

**«Неудовлетворительно»** - теоретическое содержание курса не освоено, необходимые умения не сформированы, выполненные учебные задания содержат грубые ошибки.

В соответствии с учебным планом и рабочей программы ПМ.02 **«Осуществление интеграции программного модуля»** на практические (лабораторные) занятия отводится **168 часов**.

## 2. ПЕРЕЧЕНЬ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

№ п/п	Тема практических занятий	Количество часов
Раздел 1. Разработка программного обеспечения		
1	Практическое занятие № 1. Анализ предметной области. Разработка и оформление технического задания	2
2	Практическое занятие № 2. Анализ предметной области. Разработка и оформление технического задания	2
3	Практическое занятие № 3. Анализ предметной области. Разработка и оформление технического задания	2
4	Практическое занятие № 4. Анализ предметной области. Разработка и оформление технического задания	2
5	Практическое занятие № 5. Анализ предметной области. Разработка и оформление технического задания	2
6	Практическое занятие № 6. Построение архитектуры программного средства. Изучение работы в системе контроля версий	2
7	Практическое занятие № 7. Построение архитектуры программного средства. Изучение работы в системе контроля версий	2
8	Практическое занятие № 8. Построение архитектуры программного средства. Изучение работы в системе контроля версий	2
9	Практическое занятие № 9. Построение функциональных диаграмм IDEF0 и диаграмм потоков данных DFD	2
10	Практическое занятие № 10. Построение функциональных диаграмм IDEF0 и диаграмм потоков данных DFD	2
11	Практическое занятие № 11. Построение диаграммы Вариантов использования и диаграммы Классов	2
12	Практическое занятие № 12. Построение диаграммы Вариантов использования и диаграммы Классов	2
13	Практическое занятие № 13. Построение диаграммы Состояний	2
14	Практическое занятие № 14. Построение диаграммы Состояний	2
15	Практическое занятие № 15. Построение диаграммы Деятельности и диаграммы Последовательности	2
16	Практическое занятие № 16. Построение диаграммы Деятельности и диаграммы Последовательности	2
17	Практическое занятие № 17. Разработка тестового сценария. Оценка необходимого количества тестов	2
18	Практическое занятие № 18. Разработка тестового сценария. Оценка необходимого количества тестов	2
19	Практическое занятие № 19. Разработка тестовых пакетов	2
20	Практическое занятие № 20. Разработка тестовых пакетов	2

21	Практическое занятие № 21. Оценка программных средств с помощью метрик	2
22	Практическое занятие № 22. Инспекция программного кода на предмет соответствия стандартам кодирования	2
Раздел 2. Средства разработки программного обеспечения		
1	Практическое занятие № 1 Разработка структуры проекта. Разработка модульной структуры проекта (диаграммы модулей)	2
2	Практическое занятие № 2 Разработка перечня артефактов и протоколов проекта.	2
3	Практическое занятие № 3 Настройка работы системы контроля версий. Настройка типов импортируемых файлов, путей, фильтров и др. параметров импорта в репозиторий. Разработка и интеграция модулей проекта (командная работа)	2
4	Практическое занятие № 4 Создание базы данных в MS Sql Server. Загрузка таблиц и данных. Создание таблиц спецификаций. Импортирование данных и SQL сценариев.	2
5	Практическое занятие № 5 Разработка приложения для однотабличной базы данных	2
6	Практическое занятие № 6 Разработка приложения для многотабличной базы данных	2
7	Практическое занятие № 7 Разработка приложения для многотабличной базы данных	2
8	Практическое занятие № 8 Элементы управления	2
9	Практическое занятие № 9 Оформление веб страниц	2
10	Практическое занятие № 10 Работа с базами данных при разработке веб приложений	2
11	Практическое занятие № 11 Разработка веб приложения	2
12	Практическое занятие № 12 Подключения к источнику данных	2
13	Практическое занятие № 13 Создание и выполнение команд ад источникам данных	2
14	Практическое занятие № 14 Работа с таблицами	2
15	Практическое занятие № 15 Строки и DataAdapter	2
16	Практическое занятие № 16 Отношения между таблицами	2
17	Практическое занятие № 17 Фильтрация и поиск	2
18	Практическое занятие № 18 DataSet со строгим контролем типов	2
19	Практическое занятие № 19 Обновление данных	2
20	Практическое занятие № 20 Модуль автоматизации приложения	2
21	Практическое занятие № 21 Создание проекта по юнит тестированию	2
22	Практическое занятие № 22 Модуль администратора приложения	2

23	Практическое занятие № 23 Отладка отдельных модулей программного проекта	2
24	Практическое занятие № 24 Организация обработки исключений	2
25	Практическое занятие № 25 Применение отладочных классов в проекте	2
26	Практическое занятие № 26 Отладка проекта	2
27	Практическое занятие № 27 Инспекция кода модулей проекта	2
28	Практическое занятие № 28 Тестирование интерфейса пользователя средствами инструментальной среды разработки	2
29	Практическое занятие № 29 Разработка тестовых модулей проекта для тестирования отдельных модулей	2
30	Практическое занятие № 30 Выполнение функционального тестирования	2
31	Практическое занятие № 31 Тестирование интеграции	2
Раздел 3. Моделирование в программных системах		
1	Практическое занятие № 1 Построение простейших математических моделей	2
2	Практическое занятие № 2 Построение простейших статистических моделей	2
3	Практическое занятие № 3 Задача Коши для уравнения теплопроводности	2
4	Практическое занятие № 4 Решение задач линейного программирования симплекс - методом	2
5	Практическое занятие № 5 Решение задач линейного программирования симплекс - методом	2
6	Практическое занятие № 6 Графический метод решения задач линейного программирования	2
7	Практическое занятие № 7 Нахождение начального решения транспортной задачи	2
8	Практическое занятие № 8 Нахождение начального решения транспортной задачи	2
9	Практическое занятие № 9 Решение транспортной задачи методом потенциалов	2
10	Практическое занятие № 10 Решение транспортной задачи	2
11	Практическое занятие № 11 Применение метод стрельбы для решения линейной краевой задачи	2
12	Практическое занятие № 12 Применение метод стрельбы для решения линейной краевой задачи	2
13	Практическое занятие № 13 Задача о распределении средств между предприятиями	2
14	Практическое занятие № 14 Задача о распределении средств между предприятиями	2
15	Практическое занятие № 15	2

	Задача о замене оборудования	
16	Практическое занятие № 16 Задача о замене оборудования	2
17	Практическое занятие № 17 Нахождение кратчайших путей на графе	2
18	Практическое занятие № 18 Нахождение кратчайших путей на графе	2
19	Практическое занятие № 19 Составление систем уравнений Колмогорова. Нахождение финальных вероятностей	2
20	Практическое занятие № 20 Составление систем уравнений Колмогорова. Нахождение финальных вероятностей	2
21	Практическое занятие № 21 Составление систем уравнений Колмогорова. Нахождение финальных вероятностей	2
22	Практическое занятие № 22 Составление систем уравнений Колмогорова. Нахождение финальных вероятностей	2
23	Практическое занятие № 23 Составление систем уравнений Колмогорова. Нахождение финальных вероятностей	2
24	Практическое занятие № 24 Нахождение характеристик простейших систем массового обслуживания	2
25	Практическое занятие № 25 Построение прогнозов	2
26	Практическое занятие № 26 Алгоритм поиска решения матричной антагонистической игры	2
27	Практическое занятие № 27 Алгоритм поиска решения матричной антагонистической игры	2
28	Практическое занятие № 28 Выбор оптимального решения с помощью дерева решений	2
29	Практическое занятие № 29 Решение матричной игры методом итераций	2
30	Практическое занятие № 30 Рассмотрение решения матричных игр симплексным методом	2
31	Практическое занятие № 31 Решение матричных игр графическим методом	2

### 3. СОДЕРЖАНИЕ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

#### Раздел 1. Разработка программного обеспечения

##### Практическое занятие № 1-5

**Тема:** Анализ предметной области. Разработка и оформление технического задания

**Цель:** изучить, описать и проанализировать предметную область, в которой будет создаваться информационная база. Ознакомление с процедурой разработки технического задания на создание программного продукта с применением ГОСТ 34.602-89.

**Оборудование:** тетрадь, ручка, ПК

**Методические указания:** ознакомьтесь с теоретической частью, выполните задания, ответьте на контрольные вопросы

**Ход выполнения:**

## **Теоретические сведения**

### **Анализ предметной области**

Выделяются следующие шаги работы над проектом (системой):

1. Описание предметной области, под которой понимается та часть реального мира, которая имеет существенное значение или непосредственное отношение к процессу функционирования программы. Другими словами, предметная область включает в себя только те объекты и взаимосвязи между ними, которые необходимы для описания требований и условий решения некоторой задачи. Следовательно, разработчикам необходимо выделить основные объекты (компоненты), участвующие в функционировании системы, определить их наиболее существенные характеристики, взаимосвязи в рамках решаемой задачи, а также определить основные информационные потоки в системе. При этом отдельные компоненты выбираются таким образом, чтобы при последующей разработке их было удобно представить в форме классов и объектов. В этом случае немаловажное значение приобретает и сам язык представления информации о концептуальной схеме предметной области.

Сложность предметной области определяет количество объектов и связей между ними, поэтому описание должно включать в себя базовые термины и определения, сопровождаться различными примерами, в нем могут приводиться различного рода классификации, поясняющие различные свойства описываемых объектов. Если в системе используются математические модели, то они также должны быть описаны с учетом специфики применения.

2. Обзор существующих систем-аналогов – неотъемлемая часть описания предметной области, которая позволяет разработчику определить основные концепции, необходимые для реализации в системе. Описание должно приводиться с указанием отличительных особенностей разработанных систем, с перечислением их достоинств и недостатков, в отчете обязательно приводятся экранные формы этих систем.

3. Результатом последнего этапа является диаграмма объектов предметной области и краткое описание их свойств и функций. При построении данной диаграммы нужно помнить о том, что в данном случае объект – это «конкретная материализация абстракции», а не экземпляр класса. Диаграмма объектов представляет статическую составляющую взаимодействующих между собой объектов, она должна включить в себя только те объекты предметной области, которые потом преобразуются в диаграмму классов. Связи между объектами показывают отношения между ними, при необходимости в диаграмме можно привести и атрибуты (свойства) объектов.

Диаграммы объектов не позволяют полностью описать объектную структуру системы, поэтому при их использовании нужно сосредоточиться на изображении интересующих вас наборов конкретных объектов.

Для сбора, хранения, поиска и выдачи информации о предметной области и ее объектов настоящее время в информационных системах широко используются базы данных.

**Анализ предметной области начинается с выделения сущностей и определения их свойств или атрибутов.**

Видимые сущности представляют собой объекты предметной области, которые может распознать человек.

Поддерживаемые сущности или абстрактные сущности разрабатываются для физической поддержки общей логической модели.

### **Пример анализа предметной области**

#### **Проектирование БД «Туристическая фирма»**

Туристическая фирма является юридическим лицом, имеет собственный баланс, расчетный счет в банке, печать и бланк со своим наименованием, и другие реквизиты. Деятельность осуществляется самостоятельно на принципах полного хозяйственного расчета. Предприятие имеет лицензию на право ведения деятельности по оказанию услуг в сфере туризма и сертификат соответствия.

Фирма предоставляет туристическое обслуживание по России, странам СНГ, ближнему и дальнему зарубежью. Управление фирмой осуществляется ее владельцем.

Целевой аудиторией считаются люди среднего и пожилого возраста, дети и молодежь, возможен подбор туров по индивидуальным параметрам, составление ознакомительных экскурсий по странам с учетом пожеланий клиента.

Туристические путевки, возможно, заказать практически в любые страны и направления вплоть до самых отдаленных и экзотических, предоставляемые услуги ориентированы в основном на индивидуальные заявки.

Документы, регламентирующие деятельность туристического агентства:

- внешние документы: законодательные и нормативные акты, касающиеся выполняемой работы;
- внутренние документы: устав туристического агентства, приказы и распоряжения учредителя туристического агентства;
- положение о туристическом агентстве, должностные инструкции сотрудников тур агентства, правила внутреннего трудового распорядка.

В офисе работают директор и менеджер.

Директор турагентства исполняет следующие обязанности:

- руководит в соответствии с действующим законодательством хозяйственной и финансово-экономической деятельностью турагентства, неся всю полноту ответственности за последствия принимаемых решений, сохранность и эффективное использование имущества турагентства, а также финансово-хозяйственные результаты его деятельности;
- обеспечивает достижение запланированных финансово-экономических показателей турагентства;
- руководит деятельностью структурных подразделений турагентства;
- обеспечивает выполнение турагентством всех обязательств перед сторонними организациями, заказчиками услуг, покупателями турпродуктов, а также хозяйственных, трудовых договоров и бизнес-планов;
- организует деятельность по оказанию туристических услуг и реализации турпродуктов на основе научных форм управления и организации труда, изучения конъюнктуры рынка туристических услуг и турпродуктов в целях повышения уровня их качества;
- принимает меры по обеспечению турагентства квалифицированными кадрами, рациональному использованию и развитию их профессиональных знаний и опыта, созданию безопасных и благоприятных для жизни и здоровья условий труда;
- обеспечивает правильное применение принципа материальной заинтересованности и ответственности каждого работника за порученное ему дело и результаты работы всего коллектива, выплату заработной платы в установленные сроки;
- обеспечивает соблюдение законности в деятельности турагентства и осуществлении его хозяйственно-экономических связей, использование правовых средств для финансового управления и функционирования в рыночных условиях, укрепления договорной и финансовой дисциплины, регулирования социально-трудовых отношений;
- осуществляет контроль за рациональным использованием материальных, финансовых ресурсов, дает оценку результатам деятельности турагентства и качества оказываемых услуг;
- утверждает правила внутреннего трудового распорядка, график отпусков, должностные инструкции, производственные инструкции и иные организационно-правовые документы;
- принимает решения о приеме, перемещении и увольнении подчиненных работников;
- применяет меры поощрения к отличившимся работникам; налагает взыскания на нарушителей трудовой дисциплины;
- решает в установленном порядке вопросы направления работников в служебные командировки.

Менеджер занимается следующими видами деятельности:

- изучает справочники по туризму, каталоги, иные источники туристской информации с целью формирования собственных информационных баз по туроператорам;
- изучает требования клиентов к туристским продуктам, осуществляет анализ маркетинговых исследований спроса на туристские услуги;
- устанавливает контакты с туроператорами с целью изучения программ туров, определения туров,

Исходя из анализа предметной области, нам потребуются следующие таблицы: Страны, Покупатели, Путешествия.

Далее выделяем структуру таблиц.

## Структура таблицы БД «СТРАНЫ МИРА».

ИМЯ ПОЛЯ	ТИП	Описание
 Код страны	числовой	Код страны
Страна	текстовый	Название страны
Столица	текстовый	Название столицы
Часть света	текстовый	Название части света
Население	числовой	Население (в тыс. человек)
Площадь	числовой	Площадь (в тыс. кв. км)

## Структура таблицы БД «Путешествия».

ИМЯ ПОЛЯ	ТИП	Описание
 Код путевки	числовой	Код путевки
Код страны	числовой	Код страны
Продолжительность	числовой	Кол-во дней
Дата заезда	дата	Дата выезда
Стоимость	денежный	Стоимость путевки

## Структура таблицы БД «Покупатели».

ИМЯ ПОЛЯ	ТИП	Описание
 Код покупателя	числовой	Код покупателя
Код путевки	числовой	Код путевки
Фамилия ИО	текстовый	Фамилия ИО
Оплата	логический	Да – оплатил, нет – бронь
Цель	текстовый	Цель поездки
Количество	числовой	Кол-во путевок

Модели сущность-связь основаны на выделении в предметной области, для которой осуществляется проектирование базы данных, различных типов объектов, информацию о которых требуется хранить в базе данных.

Набор однотипных объектов предметной области образует сущность. Между сущностями могут быть установлены информационные связи (зависимости), которые также могут быть учтены при проектировании схемы базы данных. Совокупность сущностей и связи между ними составляют информационную модель данных предметной области (Entity-Relationship диаграмму).

В настоящее время существует несколько приемов выделения сущностей и связей – нотации Чена, Мартина, Баркера, IDEF1X и т.д.

### Основные определения

*Сущность (Entity)* – реальный либо воображаемый объект, имеющий существенное значение для рассматриваемой предметной области. Каждая сущность должна иметь наименование, выраженное существительным в единственном числе. Каждая сущность должна обладать уникальным идентификатором. Каждый экземпляр сущности должен однозначно идентифицироваться и отличаться от всех других экземпляров данного типа сущности.

*Атрибут (Attribute)* – любая характеристика сущности, значимая для рассматриваемой предметной области и предназначенная для квалификации, идентификации, классификации, количественной характеристики или выражения состояния сущности. Наименование атрибута должно быть выражено существительным в единственном числе.

*Связь (Relationship)* – поименованная ассоциация между сущностями, значимая для рассматриваемой предметной области.

### Нотация Чена

В нотации Чена различают зависимые и независимые сущности.

Сущность называется **независимой**, если каждый экземпляр сущности может быть однозначно идентифицирован без определения его отношений с другими сущностями.

Сущность называется **зависимой** от идентификаторов или просто зависимой, если однозначная идентификация экземпляра сущности зависит от его отношения к другой сущности.

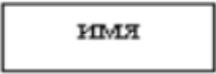
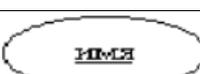
Связь соединяется с ассоциируемыми сущностями линиями. Возле каждой сущности на линии, соединяющей ее со связью, цифрами указывается класс принадлежности.

Сущности и связи могут иметь атрибуты. Для каждой сущности находится атрибут (или набор атрибутов), значение которого однозначно определяет экземпляр сущности. Этот атрибут является ключом сущности.

Связь также может иметь ключевой атрибут. В ряде случаев для удобства организации связей в состав атрибутов сущности вводится искусственный ключ (обычно число). Ключевой атрибут (набор атрибутов) на диаграмме отмечается двумя линиями снизу, внешние ключи отмечаются

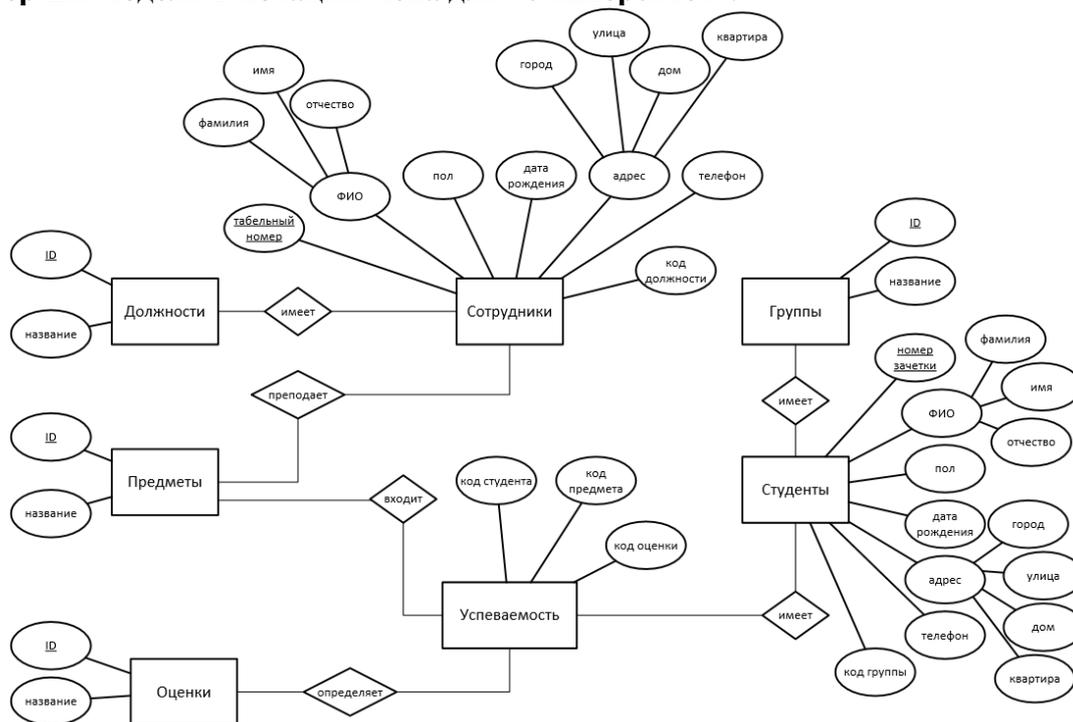
одной линией. В таблице представлены основные элементы, используемые для формирования ER-диаграммы в нотации Чена.

Таблица 1 – Элементы ER-диаграммы в нотации Чена

Элемент диаграммы	Описание
	Сущность
	Связь
	Атрибут
	Первичный ключ
	Внешний ключ

Связь имеет кардинальное число, определяющее, какое количество экземпляров одной сущности имеет информационную связь с экземплярами другой сущности.

**Пример ER-модели в нотации Чена для «Университет».**



**Понятие информационной модели. Уровни информационной модели**

Методология IDEF1X – язык для семантического моделирования данных, основанный на концепции «сущность-связь».

Различают два уровня информационной модели: **логический** и **физический**.

**Логическая модель** позволяет понять суть проектируемой системы, отражая логические взаимосвязи между сущностями.

Различают 3 подуровня логического уровня модели данных, отличающиеся по глубине представления информации о данных:

- диаграмма сущность-связь (Entity-Relationship Diagram (ERD));
- модель данных, основанная на ключах (Key Based Model (KB));
- полная атрибутивная модель (Fully Attributed Model (FA)).

**Физическая модель** отражает физические свойства проектируемой базы данных (типы данных, размер полей, индексы). Параметры физической информационной модели зависят от выбранной системы управления базами данных (СУБД).

### **Основные элементы информационной модели логического уровня**

#### *Сущности и атрибуты*

**Сущность** – это множество **реальных** или **абстрактных объектов** (людей, предметов, документов и т.п.), **обладающих общими атрибутами или характеристиками**. Любой объект системы может быть представлен только одной сущностью, которая должна быть уникально идентифицирована. *Именованная сущность* осуществляется с помощью *существительного в единственном числе*. При этом имя сущности должно отражать **тип** или **класс** объекта, а не его **конкретный экземпляр** (например, **Студент**, а не **Петров**).

Студент

ID студента
Фамилия
Имя
Отчество
Дата поступления
Номер билета

Любая сущность характеризуется набором атрибутов (**свойств**).

**Атрибут сущности** – характеристика сущности, то есть свойство реального объекта. Например, на рисунке атрибутами сущности «Студент» являются «ID студента», «Фамилия», «Имя», «Отчество», «Дата поступления» и «Номер билета».

В свою очередь, *атрибуты сущности* делятся на 2 вида: *собственные* и *наследуемые*. *Собственные* атрибуты являются уникальными в рамках модели. *Наследуемые* атрибуты передаются от сущности-родителя при установке связи с другими сущностями.

**Первичный ключ (Primary Key, PK)**. Каждая сущность должна обладать *атрибутом* или *комбинацией атрибутов*, чьи значения *однозначно определяют* каждый экземпляр сущности. Эти атрибуты образуют *первичный ключ* сущности.

**Внешний ключ (Foreign Key, FK)**. Если между двумя сущностями *имеется специфическое отношение* связи или *категоризации*, то *атрибуты*, входящие в *первичный ключ* родительской или *общей сущности*, *наследуются* в качестве *атрибутов сущностью-потомком* или *категориальной сущностью* соответственно. Эти атрибуты и называются *внешними ключами*. *Наследуемый атрибут* может использоваться в сущности в качестве части или целого *первичного ключа*, *альтернативного ключа* или не *ключевого атрибута*.

#### *Отношения в IDEF1X-модели*

При построении информационной модели различают следующие типы отношений между сущностями: *идентифицирующее, не идентифицирующее, не специфическое (многие-ко-многим) и отношения категоризации*.

**Мощность отношения** служит для обозначения отношения числа экземпляров родительской сущности к числу экземпляров дочерней.

### **Нормализация данных**

**Нормализация** – это процесс проверки и реорганизации сущностей и атрибутов с целью удовлетворения требований к реляционной модели данных. Процесс нормализации сводится к последовательному приведению структур данных к нормальным формам – *формализованным требованиям к организации данных*.

**Первая нормальная форма (1НФ)**. Сущность находится в первой нормальной форме тогда и только тогда, когда все атрибуты содержат атомарные значения. Среди атрибутов не должно встречаться повторяющихся групп, т.е. несколько значений для каждого экземпляра.

**Вторая нормальная форма (2НФ)**. Сущность находится во второй нормальной форме, если она находится в первой нормальной форме, и каждый не ключевой атрибут полностью зависит от первичного ключа (не может быть зависимости от части ключа).

**Третья нормальная форма (3НФ)**. Сущность находится в третьей нормальной форме, если она находится во второй нормальной форме и никакой не ключевой атрибут не зависит от другого не ключевого атрибута (не должно быть зависимости между не ключевыми атрибутами).



- повышение эффективности применения и снижение затрат в сфере сопровождения программных средств (ПС);
- увеличение коэффициента повторного использования ПС общего назначения;
- повышение объективности оценок качества ПС;
- создание условий для конкуренции разработчиков на внутреннем рынке ПС;
- обеспечение конкурентоспособности отечественных ПС на мировом рынке.

Стандарты комплекса ГОСТ 34 на создание и развитие автоматизированных систем (АС) – обобщенные, но воспринимаемые как весьма жесткие по структуре жизненного цикла (ЖЦ) и проектной документации. Наиболее популярными можно считать ГОСТ 34.601-90 (Автоматизированные системы. Стадии создания) и ГОСТ 34.602-89 (Техническое задание на создание АС). Введение единой, достаточно качественно определенной терминологии, наличие достаточно разумной классификации работ, документов, видов обеспечения и др. способствует более полной и качественной стыковке разных систем, что особенно важно в условиях, когда разрабатывается все больше сложных комплексных АС.

В последние годы в стране идет интенсивная работа по гармонизации государственных стандартов Российской Федерации с международными стандартами ISO в области создания нормативной базы управления жизненным циклом программных средств и информационных систем. В основе стандартизации - ГОСТ Р ИСО/МЭК 12207-99 "Информационная технология. Процессы жизненного цикла программных средств", который является аутентичным переводом международного стандарта ISO/IEC 12207: 1995-08-01.

Техническое задание (ТЗ) на АС - утвержденный в установленном порядке документ, определяющий цели, требования и основные исходные данные необходимые для разработки АС и содержащий предварительную оценку экономической эффективности.

ТЗ содержит основные технические требования, предъявляемые к изделию и исходные данные для разработки; в ТЗ указываются назначение объекта, область его применения, стадии разработки документации, её состав, сроки исполнения и т. д., а также особые требования, обусловленные спецификой самого объекта либо условиями его эксплуатации. Как правило, ТЗ составляют на основе анализа результатов, полученных в ходе предпроектного обследования.

Как инструмент коммуникации в связке общения заказчик-исполнитель, техническое задание позволяет:

**А) Обеим сторонам:**

- представить готовый продукт;
- выполнить проверку готового продукта по пунктам (приёмочное тестирование – проведение испытаний);
- уменьшить число ошибок, связанных с изменением требований в результате их неполноты или ошибочности (на всех стадиях и этапах создания, за исключением испытаний).

**Б) Заказчику:**

- осознать, что именно ему нужно, опираясь на существующие на данный момент технические возможности и свои ресурсы;
- требовать от исполнителя соответствия продукта всем условиям, оговорённым в ТЗ.

**В) Исполнителю:**

- правильно понять суть задачи, показать заказчику «технический облик» будущего программного изделия или АС;
- спланировать выполнение проекта и работать по намеченному плану;
- отказаться от выполнения работ, не указанных в ТЗ.

**Структура технического задания**

Титульный лист

Содержание

**1 ОБЩИЕ СВЕДЕНИЯ**

1.1 Полное наименование системы и ее условное обозначение

1.2 Номер договора

1.3 Наименования организации-заказчика и организации-исполнителя

1.4 Перечень документов, на основании которых создается система

1.5 Плановые сроки начала и окончания работы по созданию системы

1.6 Порядок оформления и предъявления заказчику результатов работ по созданию системы

1.7 Перечень нормативно-технических документов, методических материалов, использованных при разработке ТЗ

1.8 Определения, обозначения и сокращения

## 2 НАЗНАЧЕНИЕ И ЦЕЛИ СОЗДАНИЯ СИСТЕМЫ

2.1 Назначение системы

2.2 Цели создания системы

## 3 ХАРАКТЕРИСТИКА ОБЪЕКТА АВТОМАТИЗАЦИИ

3.1 Краткие сведения об объекте автоматизации или ссылки на документы, содержащие такую информацию

3.2 Сведения об условиях эксплуатации объекта автоматизации и характеристиках окружающей среды

## 4 ТРЕБОВАНИЯ К СИСТЕМЕ

4.1 Требования к системе в целом

4.2 Требования к функциям, выполняемым системой

4.3 Требования к видам обеспечения

4.3.1 Требования к математическому обеспечению системы

4.3.2 Требования к информационному обеспечению системы

4.3.3 Требования к лингвистическому обеспечению системы

4.3.4 Требования к методическому обеспечению системы

4.3.5 Требования к организационному обеспечению системы

4.3.6 Требования к правовому обеспечению системы

4.3.7 Требования к программному обеспечению системы

4.3.8 Требования к техническому обеспечению

4.3.9 Требования к эргономическому обеспечению

## 5 СОСТАВ И СОДЕРЖАНИЕ РАБОТ ПО СОЗДАНИЮ СИСТЕМЫ

## 6 ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ

7 ТРЕБОВАНИЯ К СОСТАВУ И СОДЕРЖАНИЮ РАБОТ ПО ПОДГОТОВКЕ ОБЪЕКТА АВТОМАТИЗАЦИИ К ВВОДУ СИСТЕМЫ В ДЕЙСТВИЕ

## 8 ТРЕБОВАНИЯ К ДОКУМЕНТИРОВАНИЮ

## 9 ИСТОЧНИКИ РАЗРАБОТКИ

Шаблон ТЗ задания приведен в приложении Б.

### Задание:

Изучить правила проведения анализа предметной области и составления ТЗ для «Туристической фирмы»

### Методика выполнения

В качестве примера возьмем *БД Туристическая фирма*.

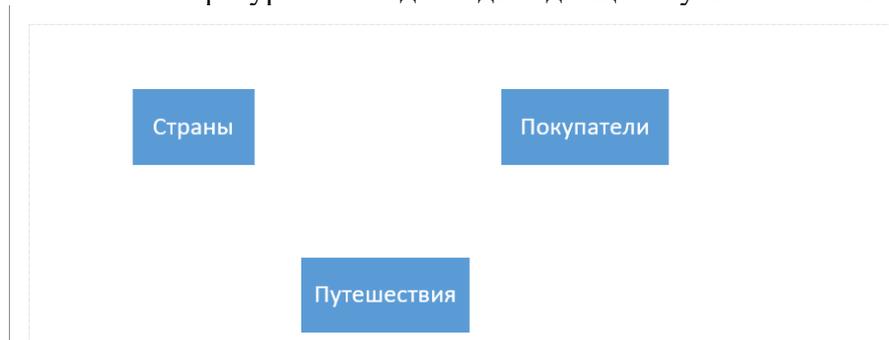
Исходя из анализа предметной области, нам потребуются следующие таблицы: Страны, Покупатели, Путешествия.

1. Запустите MS Visio.

2. В категориях шаблонов выберите *Базы данных* и в ней элемент *Нотация Чена*. Нажмите кнопку *Создать*.

3. В соответствии с анализом предметной области необходимо добавить на рабочий лист три фигуры *Сущность*.

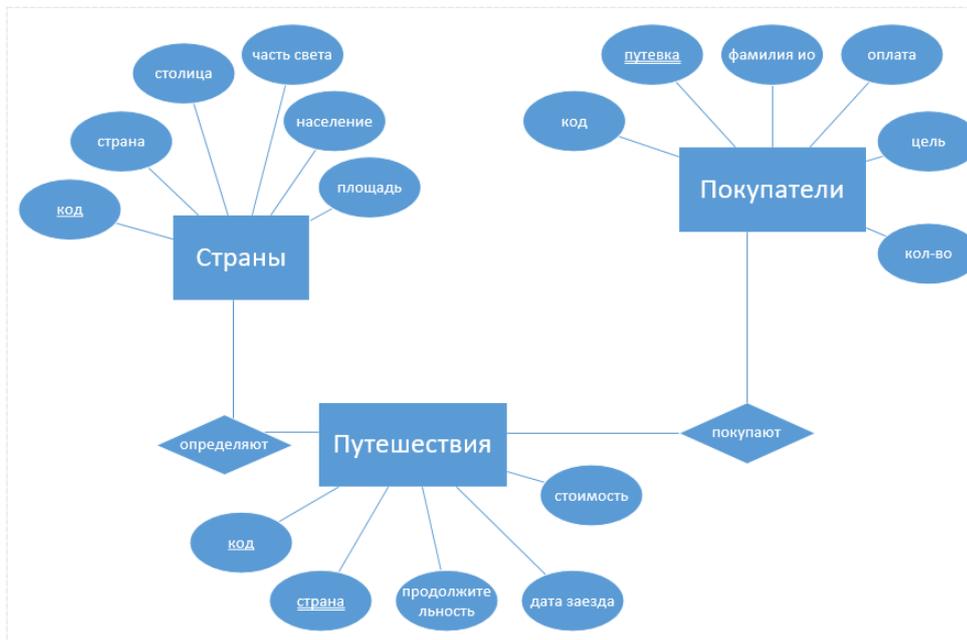
4. Для добавления текста фигуре необходимо дважды щелкнуть по ней левой кнопкой мыши.



5. Далее необходимо добавить связи между сущностями. Для этого необходимо использовать фигуру ромб.



6. На данном этапе получена общая диаграмма Чена. Для ее детализации необходимо добавить атрибуты ко всем сущностям. Атрибуты изображаются в виде овала



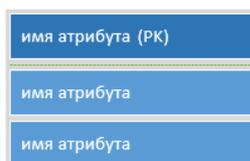
7. Запустите MS Visio.

8. В категориях шаблонов выберите *Базы данных* и в ней элемент *Нотация IDEF1X*. Нажмите кнопку *Создать*.

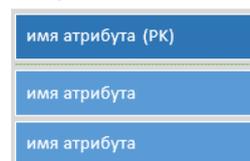
9. В соответствии с анализом предметной области необходимо добавить на рабочий лист три фигуры *Сущность*.

10. Для добавления текста фигуре необходимо дважды щелкнуть по ней левой кнопкой мыши.

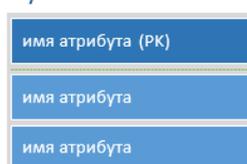
Страна



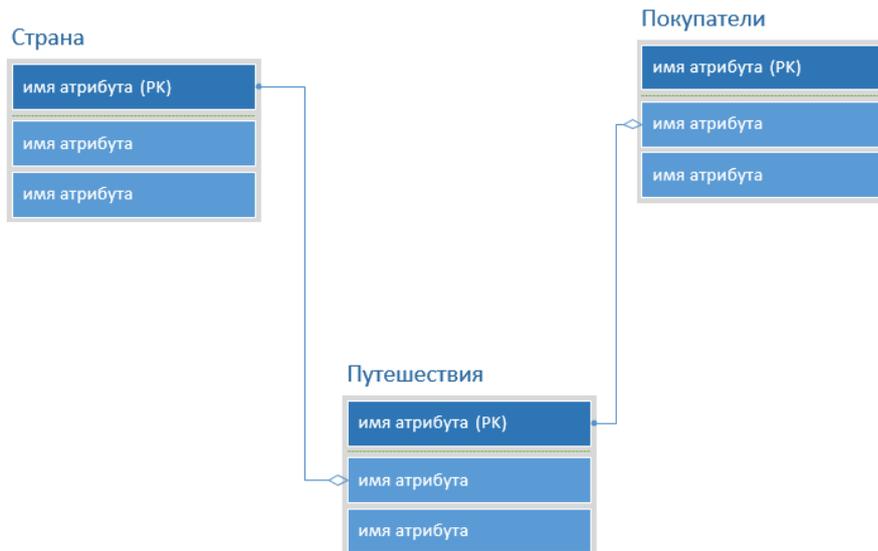
Покупатели



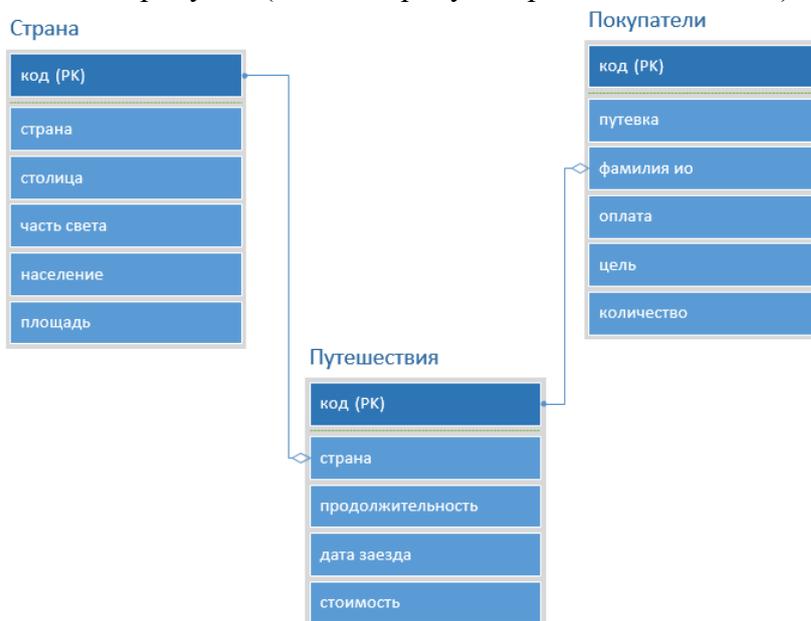
Путешествия



11. Далее необходимо добавить связи между сущностями.

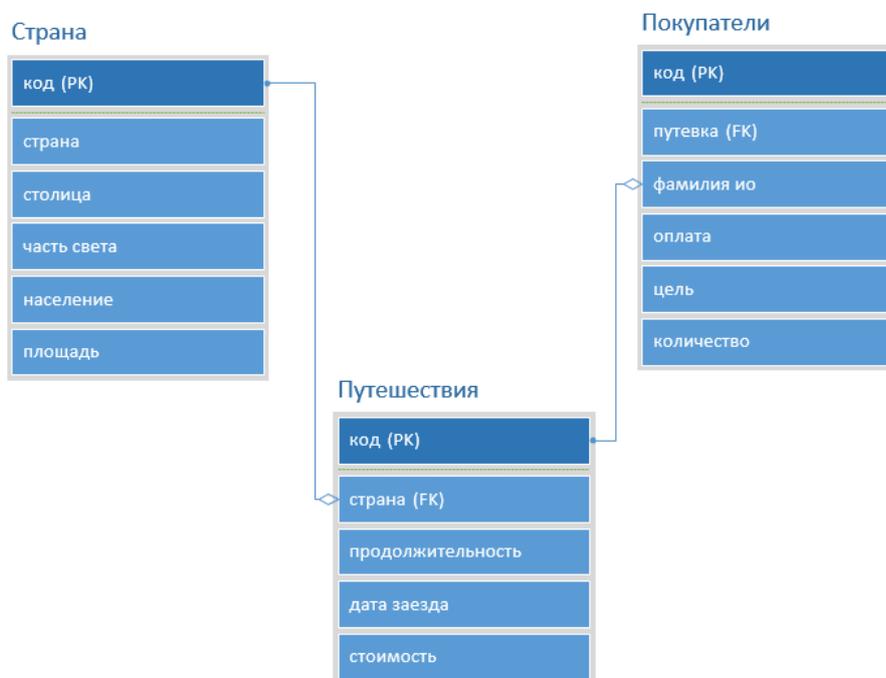


12. Добавить имена атрибутам (и сами атрибуты при необходимости).



13. Указать атрибуты, которые являются внешними ключами

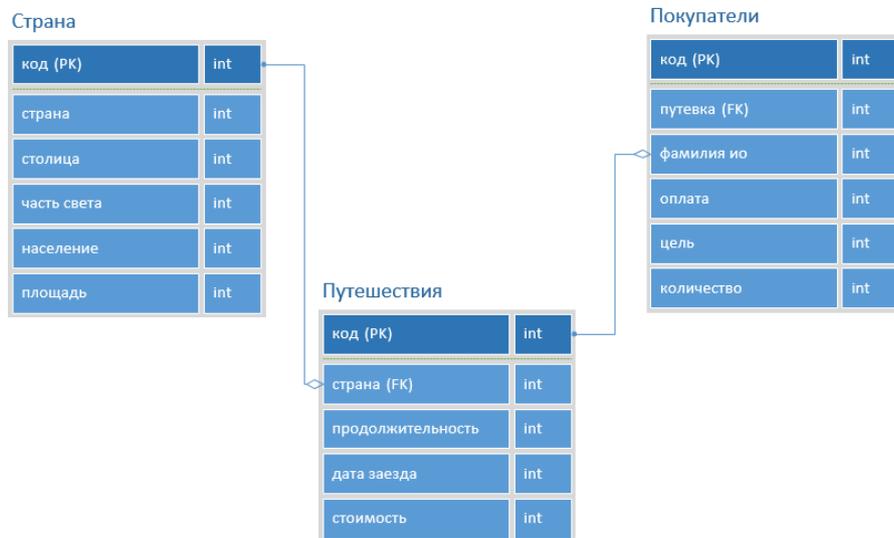
Для этого нужно выделить атрибут. Щелкнуть по нему правой кнопкой мыши и в контекстном меню выбрать пункт «Задать внешний ключ».



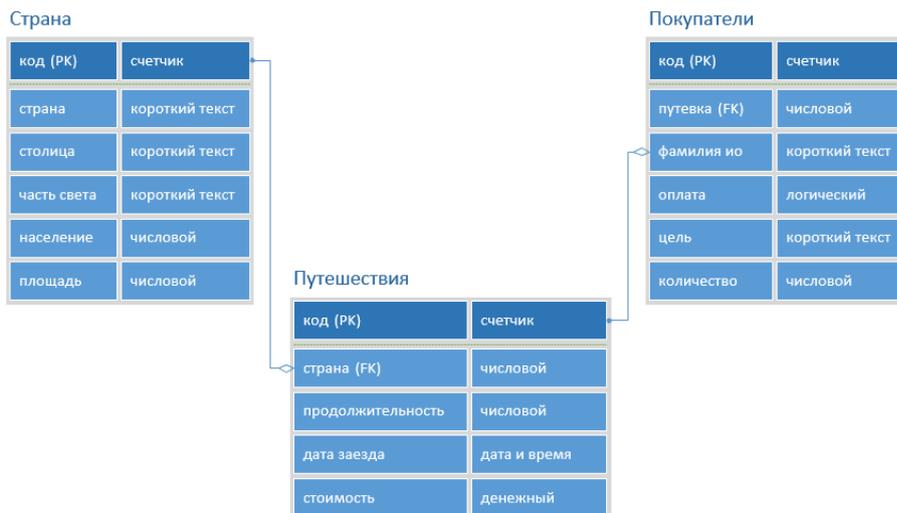
#### 14. Создание физической модели

Для этого необходимо дублировать созданную модель на новый лист. Для этого щелкнуть правой кнопкой мыши по текущему листу и в контекстном меню выбрать пункт Дублировать.

15. Затем на новом листе по очереди выбрать каждую сущность и в ее контекстном меню выбрать пункт Показать типы атрибутов.



16. Затем необходимо указать соответствующие типы данных, в зависимости от СУБД в которой будет выполняться реализация системы. Например, для MS Access модель примет вид:



Чтобы изменить размер ячейки, в которой указывается тип данных нужно выделить сущность и используя желтые точки изменить размер.



### **Задание самостоятельной работы**

В соответствии с индивидуальным вариантом, провести анализ предметной области, построить ER-модель в нотации Чена и IDEF1X и на основании ГОСТ 34.602-89, разработать документ Техническое задание.

Перечень индивидуальных вариантов приведен в приложении А.

#### **Контрольные вопросы**

1. Что такое предметная область?
2. Этапы анализа предметной области.
3. Что такое ER-модель?
4. В каких нотациях можно построить ER-модель?
5. Элементы диаграммы Чена.
6. Элементы диаграммы в нотации IDEF1X.
7. Для чего предназначена диаграмма «сущность-связь»?
8. Что представляет собой нормализация?
9. В чем разница между логическим уровнем модели данных и физическим?
10. Назначение, содержание и степень адаптивности стандарта ГОСТ 34.601-90?
11. Стадии и этапы создания АС в соответствии с ГОСТ 34.601-90?
12. Назначение, содержание стандарта ГОСТ 34.602-89?
13. Назначение программного документа Техническое задание на создание АС. Порядок разработки, согласования и утверждения документа?
14. Состав и содержание документа ТЗ?

**Форма отчета:** отчет, защита работы.

### **Практическое занятие № 6-8**

**Тема:** Построение архитектуры программного средства. Изучение работы в системе контроля версий

**Цель:** Изучение базовых принципов и шаблонов построения архитектуры приложений, выбор стратегии и шаблона проектирования, которые помогут при проектировании слоев, компонентов и сервисов решения, а также визуальное проектирование архитектуры приложения с использованием Microsoft Visio.

Изучить на практике понятия и компоненты систем контроля версий (СКВ), приемы работы с ними. Освоить специализированное ПО и распространенный сервис для работы с распределенной СКВ Git — TortoiseGit и GitHub.com.

**Оборудование:** тетрадь, ручка, ПК

**Методические указания:** ознакомьтесь с теоретической частью, выполните задания, ответьте на контрольные вопросы

**Ход выполнения:**

**Теоретические сведения**

#### **Построение архитектуры программного средства**

При возникновении потребностей в заказе, приобретении, разработке, эксплуатации и сопровождении программ перед всеми сторонами, вовлеченными в жизненный цикл программного средства (ПС), возникает целый ряд вопросов, связанных с определением и детальным структурированием жизненного цикла (ЖЦ) ПС, с организационными и техническими правами и обязанностями сторон, с управлением ЖЦ и контролем за его реализацией. Одним из действенных инструментов для решения данных вопросов является использование унифицированных подходов, закрепленных в современных международных и российских стандартах.

Понятия «жизненный цикл системы» или «жизненный цикл программного средства» часто появляются в статьях и звучат в разговорах разработчиков, по крайней мере, руководителей проектов и подразделений.

Всем понятно, что относятся они к тому, что и в какой последовательности должно делаться при создании и эксплуатации систем. Но прежде чем две организации или два специалиста договорятся о том, что конкретно входит или не входит в ЖЦ, проходит значительное время. А позже вполне может обнаружиться, что эти двое (две «стороны») все-таки по-разному понимают, какие работы будут входить в ЖЦ, а какие - нет, какие проверки будут планироваться, когда и т. д. Естественно, общие принципы организации работ описаны давно, но что делать сторонам в конкретном проекте — это каждый раз приходится решать заново.

В стандартах, регламентирующих жизненный цикл программных средств, обобщаются опыт и результаты исследований множества специалистов и рекомендуются наиболее эффективные современные методы и процессы создания и развития комплексов программ. В результате таких обобщений оттачиваются технологические процессы и приемы разработки, а также методическая база для их автоматизации.

ЖЦ ПС в стандартах представляет собой набор этапов, частных работ и операций в последовательности их выполнения и взаимосвязи, регламентирующих ведение работ от подготовки технического задания до завершения испытаний ряда версий и окончания эксплуатации ПС или информационной системы (ИС).

Стандарты включают правила описания исходной информации, способов и методов выполнения операций, устанавливают правила контроля технологических процессов, требования к оформлению их результатов, а также регламентируют содержание технологических и эксплуатационных документов на комплексы программ. Они определяют организационную структуру коллектива, обеспечивают распределение и планирование заданий, а также, контроль за ходом создания ПС.

Для того чтобы привести порядок и понимание, общие для любых сторон, участвующих в ЖЦ систем и ПС, давно разрабатывались стандарты различных уровней утверждения - национальные и международные.

В России основы построения и использования профилей стандартов ЖЦ ПС заложены принятием в качестве базового стандарта ГОСТ Р ИСО/МЭК 12207. Данный документ введен в действие с 1 июля 2000 г., тесно взаимоувязан с рядом стандартов, принятых ранее, и с некоторыми стандартами, разрабатываемыми в данное время на основе прямого применения стандартов ИСО.

Актуальность стандарта ГОСТ Р ИСО/МЭК 12207 для современных условий настолько высока, что принятие в ISO его исходного, международного варианта вскоре вызвало самую положительную оценку российских экспертов. Был дан ряд рекомендаций по его использованию в реальных условиях.

В данном стандарте программное обеспечение (ПО) или программный продукт определяется как набор компьютерных программ, процедур и связанной с ними документации и данных.

Процесс определяется как совокупность взаимосвязанных действий, преобразующих некоторые входные данные в выходные. Каждый процесс характеризуется определенными задачами и методами их решения, исходными данными, полученными от других процессов, и результатами.

В соответствии с ГОСТ Р ИСО/МЭК 12207 все процессы ЖЦ ПО разделены на три группы:

1) Основные процессы:

- приобретение;
- поставка;
- разработка;
- эксплуатация;
- сопровождение.

2) Вспомогательные процессы:

- документирование;
- управление конфигурацией;
- обеспечение качества;
- верификация;
- аттестация;
- совместная оценка;
- аудит;
- разрешение проблем.

3) Организационные процессы:

- управление;
- усовершенствование;
- создание инфраструктуры;
- обучение.

Процесс разработки предусматривает действия и задачи, выполняемые разработчиком, и включает следующие действия:

А) Подготовительная работа, которая начинается с выбора модели ЖЦ ПО, соответствующей масштабу, значимости и сложности проекта. Действия и задачи процесса должны соответствовать выбранной модели. Разработчик должен выбрать, адаптировать к условиям проекта и использовать согласованные с заказчиком стандарты, методы и средства разработки, а также составить план выполнения работ.

Б) Анализ требований к системе подразумевает определение ее функциональных возможностей, пользовательских требований, требований к надежности и безопасности, требований к внешним интерфейсам и т.д. Требования к системе оцениваются исходя из критериев реализуемости и возможности проверки при тестировании.

Анализ требований к ПО предполагает определение следующих характеристик для каждого компонента ПО:

– функциональных возможностей, включая характеристики производительности и среды функционирования компонента;

– внешних интерфейсов;

– спецификаций надежности и безопасности;

– эргономических требований;

– требований к используемым данным;

– требований к установке и приемке;

– требований к пользовательской документации;

– требований к эксплуатации и сопровождению.

Требования к ПО оцениваются исходя из критериев соответствия требованиям к системе, реализуемости и возможности проверки при тестировании.

В) Проектирование архитектуры системы на высоком уровне заключается в определении компонентов ее оборудования, ПО и операций, выполняемых эксплуатирующим систему персоналом. Архитектура системы должна соответствовать требованиям, предъявляемым к системе, а также принятым проектным стандартам и методам.

Проектирование архитектуры ПО включает следующие задачи:

– трансформацию требований к ПО в архитектуру, определяющую на высоком уровне структуру ПО и состав ее компонентов;

– разработку и документирование программных интерфейсов ПО и баз данных;

– разработку предварительной версии пользовательской документации;

– разработку и документирование предварительных требований к тестам и планам интеграции ПО.

Архитектура компонентов ПО должна соответствовать требованиям, предъявляемым к ним, а также принятым проектным стандартам и методам.

Г) Детальное проектирование ПО включает следующие задачи:

– описание компонентов и интерфейсов между ними на более низком уровне, достаточном для их последующего самостоятельного кодирования и тестирования;

– разработку и документирование детального проекта базы данных;

– обновление (при необходимости) пользовательской документации;

Д) Кодирование и тестирование ПО охватывает задачи:

– разработку и документирование каждого компонента ПО и базы данных, а также совокупности тестовых процедур и данных для их тестирования;

– тестирование каждого компонента ПО и базы данных на соответствие предъявляемых к ним требованиям. Результаты тестирования компонентов должны быть документированы;

– обновление (при необходимости) пользовательской документации;

– обновление плана интеграции ПО.

Е) Интеграция ПО предусматривает сборку разработанных компонентов ПО в соответствии с планом интеграции и тестирование агрегированных компонентов. Для каждого из агрегированных компонентов разрабатываются наборы тестов и тестовые процедуры, предназначенные для проверки каждого из квалификационных требований при последующем квалификационном тестировании.

Ж) Квалификационное тестирование - это набор критериев и условий,

которые необходимо выполнить, чтобы квалифицировать программный продукт как соответствующий своим спецификациям и готовый к использованию в условиях эксплуатации.

Квалификационное тестирование ПО проводится разработчиком, в присутствии заказчика (по возможности), для демонстрации того, что ПО удовлетворяет своим спецификациям и готово к использованию в условиях эксплуатации. Квалификационное тестирование выполняется для каждого компонента ПО по всем разделам требований при широком варьировании тестов.

З) Установка ПО осуществляется разработчиком в соответствии с планом в той среде и на том оборудовании, которые предусмотрены договором. В процессе установки проверяется работоспособность ПО и баз данных.

И) Приемка ПО предусматривает оценку результатов квалификационного тестирования ПО и системы и документирование результатов оценки, которые проводятся заказчиком с помощью разработчика.

Создание архитектуры приложения — это процесс формирования структурированного решения, отвечающего всем техническим и операционным требованиям и обеспечивающего оптимальные общие атрибуты качества, такие как производительность, безопасность и управляемость. Он включает принятие ряда решений на основании широкого диапазона факторов. Каждое из этих решений может иметь существенное влияние на качество, производительность, удобство обслуживания и общий успех приложения.

Как и любая другая сложная структура, программное обеспечение (ПО) должно строиться на прочном фундаменте. Неправильное определение ключевых сценариев, неправильное проектирование общих вопросов или неспособность выявить долгосрочные последствия основных решений могут поставить под угрозу все приложение. Современные инструменты и платформы упрощают задачу по созданию приложений, но не устраняют необходимости в тщательном их проектировании на основании конкретных сценариев и требований. Неправильно выработанная архитектура обуславливает нестабильность ПО, невозможность поддерживать существующие или будущие бизнес-требования, сложности при развертывании или управлении в среде производственной эксплуатации. Проектирование систем должно осуществляться с учетом потребностей пользователя, системы (ИТ-инфраструктуры) и бизнес-целей. Для каждой из этих составляющих определяются ключевые сценарии и выделяются важные параметры качества (например, надежность или масштабируемость), а также основные области удовлетворенности и неудовлетворенности. По возможности необходимо выработать и учесть показатели успешности в каждой из этих областей.

Основное назначение архитектуры — описание использования или взаимодействия основных элементов и компонентов приложения. Выбор структур данных и алгоритмов их обработки или деталей реализации отдельных компонентов являются вопросами проектирования. Часто вопросы архитектуры и проектирования пересекаются. Вместо того чтобы вводить жесткие правила, разграничивающие архитектуру и проектирование, имеет смысл комбинировать эти две области.

В некоторых случаях, принимаемые решения, очевидно, являются архитектурными по своей природе, в других — больше касаются проектирования и реализации архитектуры. Приступая к работе над архитектурой приложения, необходимо помнить об основных принципах проектирования. Это поможет создать архитектуру, которая будет следовать проверенным подходам, обеспечит минимизацию затрат, простоту обслуживания, удобство использования и расширяемость.

Рассмотрим основные принципы:

1. Разделение функций. Разделите приложение на отдельные компоненты, по возможности, минимальным перекрытием функциональности. Важным фактором является предельное уменьшение количества точек соприкосновения, что обеспечит высокую связность (high cohesion) и слабую связанность (low coupling). Неверное разграничение функциональности может привести к высокой связанности и сложностям взаимодействия, даже несмотря на слабое перекрытие функциональности отдельных компонентов.

2. Принцип единственности ответственности. Каждый отдельно взятый компонент или модуль должен отвечать только за одно конкретное свойство/функцию или совокупность связанных функций.

3. Принцип минимального знания (также известный как Закон Деметера (Law of Demeter, LoD)). Компоненту или объекту должны быть известны внутренние детали других компонентов или объектов.

4. Не повторяйтесь. Намерение должно быть обозначено только один раз. В применении к проектированию приложения это означает, что определенная функциональность должна быть реализована только в одном компоненте и не должна дублироваться ни в одном другом компоненте.

5. Минимизируйте проектирование наперед. Проектируйте только то, что необходимо. В некоторых случаях, когда стоимость разработки или издержки в случае неудачного дизайна очень высоки, может потребоваться полное предварительное проектирование и тестирование. В других случаях, особенно при гибкой разработке, можно избежать масштабного проектирования. Если требования к приложению четко не определены, или существует вероятность изменения дизайна со временем, старайтесь не тратить много сил на проектирование раньше времени.

Цель архитектора ПО при проектировании приложения или системы — максимальное упрощение дизайна через его разбиение на функциональные области. Например, пользовательский интерфейс (user interface, UI), выполнение бизнеспроцессов или доступ к данным — все это разные функциональные области. Компоненты в каждой из этих областей должны реализовывать данную конкретную функциональность и не должны смешивать в себе код разных функциональных областей. Так в компонентах UI не должно быть кода прямого доступа к источнику данных; для извлечения данных в них должны использоваться либо бизнес-компоненты, либо компоненты доступа к данным.

Также необходимо проанализировать соотношение затрат/выгод для инвестиций в приложение. В некоторых случаях может быть целесообразным упростить структуру и разрешить, например, связывание элементов UI с результирующими данными. В общем, оценивайте реализацию функциональности также и с коммерческой точки зрения. Далее приводятся обобщенные рекомендации, которые помогут учесть широкий диапазон факторов, влияющих на проектирование, реализацию, развертывание, тестирование и обслуживание приложения.

Архитектура программного обеспечения включает в себе ряд важных решений об организации программной системы, среди которых выбор структурных элементов и их интерфейсов, составляющих и объединяющих систему в единое целое; поведение, обеспечиваемое совместной работой этих элементов; организацию этих структурных и поведенческих элементов в более крупные подсистемы, а также архитектурный стиль, которого придерживается данная организация.

Выбор архитектуры ПО также касается функциональности, удобства использования, устойчивости, производительности, повторного использования, понятности, экономических и технологических ограничений, эстетического восприятия и поиска компромиссов.

Архитектурный стиль, иногда называемый архитектурным шаблоном — это набор принципов, высокоуровневая схема, обеспечивающая абстрактную инфраструктуру для семейства систем. Архитектурный стиль улучшает секционирование и способствует повторному использованию дизайна благодаря обеспечению решений часто встречающихся проблем. Архитектурные стили и шаблоны можно рассматривать как набор принципов, формирующих приложение.

Понимание архитектурных стилей обеспечивает несколько преимуществ. Самое главное из них — общий язык. Также они дают возможность вести диалог, не касаясь технологий, т. е. обсуждать схемы и принципы, не вдаваясь в детали. Например, архитектурные стили позволяют сравнивать схему клиент/сервер с n-уровневой схемой приложения. Архитектурные стили можно организовать по их фокусу.

В таблице приведен список типовых архитектурных стилей, и дается краткое описание каждого из них.

Архитектурный стиль/парадигма	Описание
Клиент/сервер	Система разделяется на два приложения, где клиент выполняет запросы к серверу. Во многих случаях в роли сервера выступает база данных, а логика приложения представлена процедурами хранения
Компонентная архитектура	Дизайн приложения разлагается на функциональные или логические компоненты с возможностью повторного использования, предоставляющие тщательно проработанные интерфейсы связи

Дизайн на основе предметной области 4	Объектно-ориентированный архитектурный стиль, ориентированный на моделирование сферы деловой активности и определяющий бизнес-объекты на основании сущностей этой сферы
Многослойная архитектура	Функциональные области приложения разделяются на многослойные группы (уровни)
Шина сообщений	Архитектурный стиль, предписывающий использование программной системы, которая может принимать и отправлять сообщения по одному или более каналам связи, так что приложения получают возможность взаимодействовать, не располагая конкретными сведениями друг о друге
N-уровневая / 3-уровневая	Функциональность выделяется в отдельные сегменты, во многом аналогично многослойному стилю, но в данном случае сегменты физически располагаются на разных компьютерах.
Объектно-ориентированная	Парадигма проектирования, основанная на распределении ответственности приложения или системы между отдельными многократно используемыми и самостоятельными объектами, содержащими данные и поведение
Сервисно-ориентированная архитектура (SOA)	Описывает приложения, предоставляющие и потребляющие функциональность в виде сервисов с помощью контрактов и сообщений

Архитектура программной системы практически никогда не ограничена лишь одним архитектурным стилем, зачастую она является сочетанием архитектурных стилей, образующих полную систему. Например, может существовать SOA-дизайн, состоящий из сервисов, при разработке которых использовалась многослойная архитектура и объектно-ориентированный архитектурный стиль.

Рассмотрим основные базовые типы приложений:

1. Мобильные приложения. Приложения этого типа могут разрабатываться как тонкий клиент или насыщенное клиентское приложение. Насыщенные клиентские мобильные приложения могут поддерживать сценарии без постоянного подключения или без подключения вообще. Веб-приложения или тонкие клиентские приложения поддерживают только сценарии с подключением. Ограничением при разработке мобильных приложений могут быть устройства, на которых их предполагается выполнять.

2. Насыщенные клиентские приложения. Приложения этого типа обычно разрабатываются как самодостаточные приложения с графическим пользовательским интерфейсом, который обеспечивает отображение данных с помощью набора элементов управления. Насыщенные клиентские приложения могут поддерживать сценарии без подключения или без постоянного подключения, если должны выполнять доступ к удаленным данным или функциональности.

3. Насыщенные Интернет-приложения. Приложения этого типа могут поддерживать множество платформ и браузеров. Насыщенные Интернет-приложения выполняются в изолированной программной среде браузера, которая ограничивает доступ к некоторым возможностям клиента.

4. Сервисные приложения. Сервисы предоставляют бизнес-функциональность для совместного использования и позволяют клиентам доступ к ней из локальной или удаленной системы. Вызов операций сервиса осуществляется с помощью сообщений, соответствующих XML-схемам и передаваемых по транспортным каналам. Целью данного типа приложений является обеспечение слабой связанности между клиентом и сервером.

5. Веб-приложения. Приложения этого типа, как правило, поддерживают сценарии с постоянным подключением и различные браузеры, выполняющиеся в разнообразнейших операционных системах и на разных платформах.

6. Существует множество других более специализированных типов приложений. Как правило, эти типы являются специализациями или сочетаниями базовых типов, перечисленных в данном списке.

В таблице перечислены преимущества и недостатки общих архетипов приложений.

Тип приложения	Преимущества	Недостатки
Насыщенные клиентские приложения	Возможность использования ресурсов клиента. Лучшее время отклика, насыщенная функциональность UI и улучшенное взаимодействие с пользователем. Очень динамичное взаимодействие с коротким временем отклика. Поддержка сценариев без подключения и сценариев без постоянного подключения	Сложность развертывания; при этом широкий выбор вариантов установки, таких как ClickOnce, Windows Installer и XCOPY. Сложности обеспечения совместимости версий. Зависимость от платформы
Мобильные приложения	Поддержка портативных устройств. Доступность и простота использования для мобильных пользователей. Поддержка сценариев без подключения и сценариев без постоянного подключения	Ограниченные возможности ввода и навигации. Ограниченная область отображения экрана
Насыщенные Интернетприложения (RIA)	Такие же насыщенные возможности пользовательского интерфейса, как и для насыщенных клиентов. Поддержка насыщенных и потоковых мультимедиа и графики. Простота развертывания с возможностями распределения (насыщенными) такими же, как и для Веб-клиентов. Простота обновления и смены версий. Поддержка различных платформ и браузеров	Большой объем памяти, занимаемый на клиенте, по сравнению с Веб-приложением. Ограниченное использование ресурсов клиента по сравнению с насыщенным клиентским приложением. Необходимость развертывания на клиенте подходящей среды выполнения
Сервисные приложения	Слабо связанное взаимодействие между клиентом и сервером. Могут использоваться различными и невязанными приложениями. Поддержка для обеспечения возможности взаимодействия	Отсутствие поддержки UI. Зависимость от возможности сетевого подключения
Веб-приложения	Широко доступный и основанный на стандартах UI, поддерживаемый на многих платформах. Простота развертывания и внесения изменений	Необходимость устойчивого сетевого подключения. Сложно обеспечить насыщенный пользовательский интерфейс

Каждый тип приложения может быть реализован с использованием одной или более технологий. Выбор технологии будет определяться сценариями и ограничениями технологий, а также возможностями и опытом группы разработки.

### Изучение работы в системе контроля версий

Система контроля версий Git представляет собой набор программ командной строки. Доступ к ним можно получить из терминала (в \*nix) или из специальной консольной оболочки Git Bash (в Windows). Однако, работа ориентирована на применение графической надстройки TortoiseGit (аналог в Linux — RabbitVCS).

TortoiseGit работает не как отдельная программа, а встраивается в контекстные меню «Проводника» Windows. Вместе с Git для Windows поставляется также программа gitk (Git GUI) — она гораздо менее популярна и пользоваться ей не следует.

### Методика выполнения

#### Построение архитектуры программного средства

Визуальное проектирование может быть осуществлено с помощью программ, поддерживающих базовые функции по построению диаграмм. Рассмотрим основные элементы, которые можно использовать для построения архитектуры в Microsoft Visio.

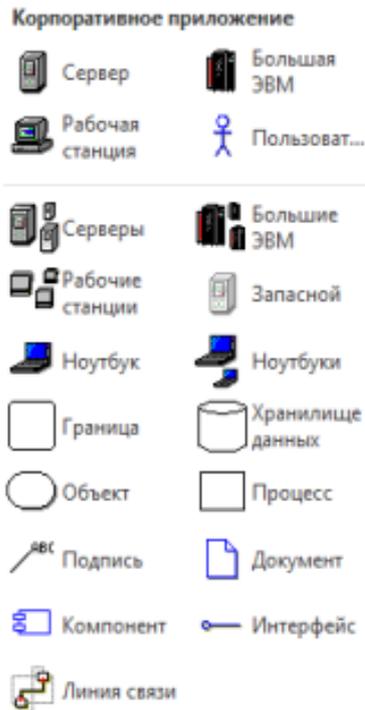
При разработке основные элементы можно взять из шаблонов «Программы и базы данных».



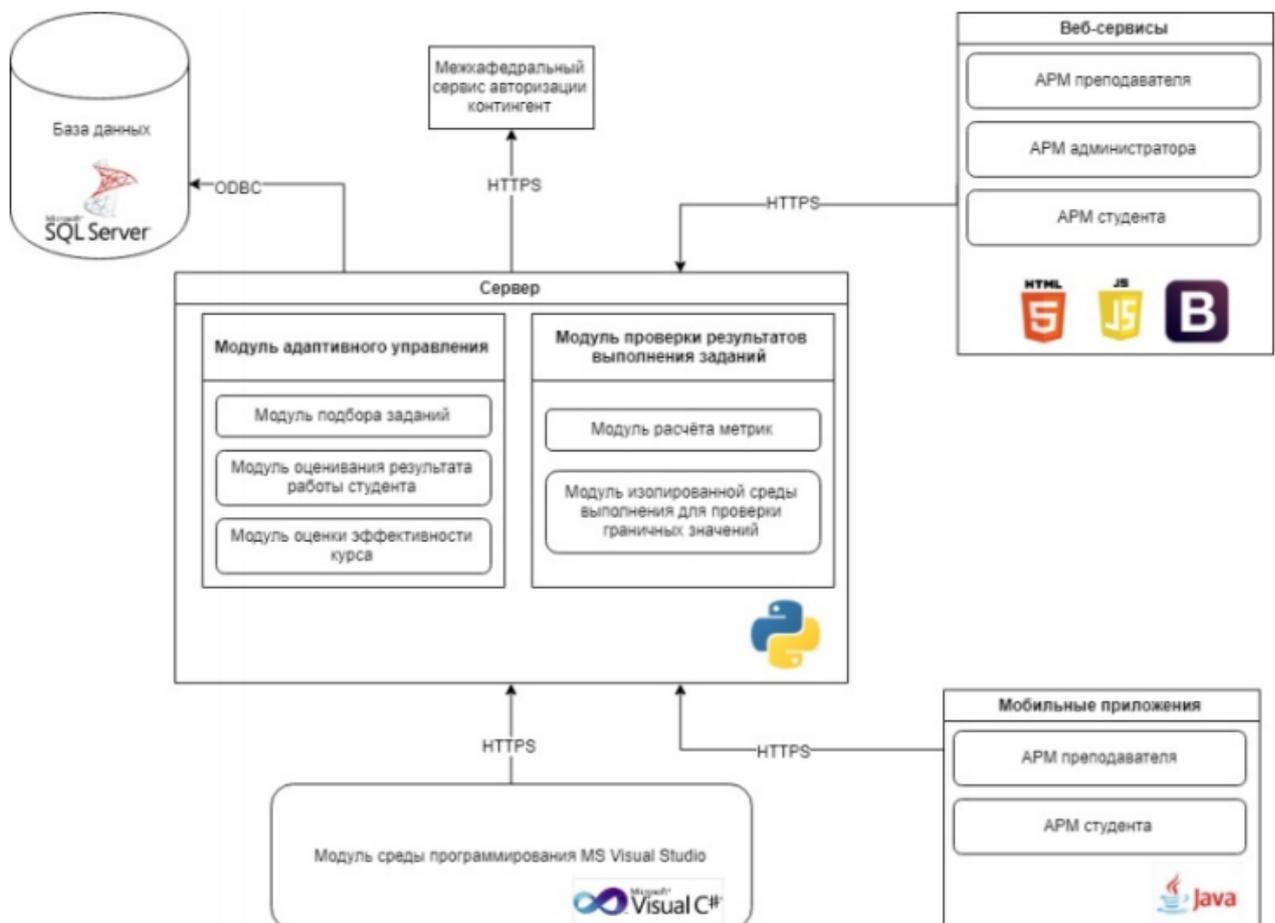
Дополнительно можно использовать элементы из шаблонов «Сеть».



При проектировании архитектуры используются блоки: «Процесс», «Объект», «Компонент», «Линия связи» и т. п.



Пример разработанной архитектуры приложения



## Изучение работы в системе контроля версий

1. Отработать навыки использования хранилища на локальной машине.

1.1. Настроить Git, указав имя и e-mail разработчика для подписи commit-ов.

*Указание. Диалог настроек вызывается пунктом TortoiseGit → Settings контекстного меню любого каталога, нужная вкладка называется «Git». Задавать следует глобальные настройки (для всех хранилищ), установив переключатель «Global».*

1.2. Создать хранилище для учебного проекта.

1.3. Совершить несколько commit-ов.

1.3.1. Скопировать sdt.h в каталог хранилища и создать файл main.cpp со включением sdt.h и пустой функцией main().

1.3.2. Добавить в программу ввод двух целых чисел с приглашением.

*Указание № 1. После выполнения каждого подпункта необходимо убеждаться, что программа работает, и совершать commit изменений.*

*Указание № 2. Следите за тем, какие файлы отмечены в списке для commita изменений в них, — кроме main.cpp и, иногда, sdt.h, больше никаких других не нужно.*

1.4. Предотвратить автоматическое добавление в хранилище файлов, не нуждающихся в контроле версий, — \*.o и \*.exe.

Правило об игнорировании следует помещать в файл .gitignore в корневом каталоге хранилища («.gitignore in repository root»). Этот файл также попадает под контроль версий, поэтому после создания правил требуется совершить commit изменений в файле .gitignore.

1.5. Добавить в программу вывод суммы введенных чисел и совершить commit.

1.6. Просмотреть историю (журнал) хранилища.

1.7. Просмотреть разность (diff) между пунктами истории 1.3.2 и 1.5.

2. Освоить передачу истории хранилища по сети.

2.1. Организовать общее хранилище на удаленном сервере.

2.1.1. Зарегистрироваться на GitHub.

2.1.2. Создать пустое удаленное хранилище с любым наименованием.

*Указание. Вопреки инструкции на GitHub, добавлять в хранилище файл README.md не нужно, удаленное хранилище должно быть пустым.*

2.1.3. Разрешить пользователям-преподавателям совершать commit-ы.

Требуется на странице хранилища выбрать «Settings» (справа), далее «Collaborators», где ввести имена пользователей, которым будет предоставлен полный доступ к хранилищу (эти имена можно узнать у лаборантов).

2.1.4. Настроить локальное хранилище для синхронизации с удаленным.

Необходимо в контекстном меню каталога локального хранилища выбрать TortoiseGit → Settings, где перейти к пункту Remote . Достаточно ввести условное имя удаленного хранилища «origin» и его адрес, который отображается на веб-странице удаленного хранилища в разделе «Quick setup» (вариант HTTP).

От загрузки сведений о ветвлениях в удаленном хранилище отказаться.

2.2. Передать локальное хранилище на удаленный сервер (push).

*Замечание. Здесь и далее при взаимодействии с удаленным сервером потребуются вводить имя пользователя и пароль, с которыми выполнялась регистрация на GitHub.*

2.3. Перейти к странице хранилища на GitHub (обновить её) и ознакомиться с возможностями просмотра содержимого через web-интерфейс.

2.4. Загрузить копию удаленного хранилища на локальную машину (clone).

*Замечание. Целью является имитация совместной работы с удаленным хранилищем. Для этого на одной машине организуются 2 локальных хранилища: созданное в пункте 1.1 (RepoA) и загруженное с удаленного сервера (RepoB).*

*Указание. Диалог «Git clone» следует вызывать из контекстного меню каталога вне локального хранилища. В качестве URL потребуется указать адрес удаленного хранилища, а в качестве Directory — имя каталога для нового локального хранилища.*

2.5. Сымитировать параллельную работу над проектом.

2.5.1. В локальном хранилище RepoB добавить в программу печать разности введенных чисел, сделать commit и передать изменения на сервер.

2.5.2. В локальном хранилище RepoA добавить над функцией main() комментарий о том, что программа является учебной, сделать commit, но не отправлять изменений на сервер.

2.6. На странице хранилища на GitHub перейти в раздел Commits и ознакомиться с возможностью просмотра истории изменений через web-интерфейс.

2.7. В локальном хранилище RepoA выполнить загрузку с сервера новейших ветвлений и изменений (fetch) и просмотреть журнал хранилища.

*Указание. По умолчанию показывается только текущая активная ветвь (по умолчанию — master). Просмотреть все commit-ы во всех ветвях, в том числе в загруженной из удаленного хранилища ветви origin/master, нужно включить флажок «All branches» слева снизу окна журнала.*

2.8. Совместить изменения в локальном хранилище с загруженными.

2.8.1. Использовать действие pull для загрузки изменений с удаленного сервера и автоматического совмещения их с имеющимися локально.

Просмотреть журнал изменений (или обновить кнопкой Refresh).

*Примечание. Фактически, при обновлении производится слияние ветвей master origin/master — то есть, двух версий истории, существовавших удаленно и локально. При этом история стала нелинейной и появился лишний commit слияния. Иногда такое усложнение имеет смысл, но в данном случае было бы желательно сохранить историю линейной и просто перенести локальные наработки вслед за новейшими. Добьемся желаемого.*

2.8.2. Отменить неудобный результат действия pull.

*Указание. В журнале изменений в контекстном меню commit-а, где был добавлен комментарий (то есть, последнего перед слиянием), выбрать «Reset master to this...» и указать тип отмены «Hard».*

*Указание. Журнал изменений не всегда обновляется автоматически, используйте кнопку Refresh, если изменения не появились сразу.*

2.8.3. Выполнить перенос (rebase) локальных изменений на основу новейшего загруженного состояния проекта.

*Указание. В журнале изменений в контекстном меню пункта, на котором находится конец ветви origin/master (прямоугольник в TortoiseGit), следует выбрать пункт «Rebase "master" onto this...» и далее нажать кнопку «Start rebase».*

2.9. Передать итоговое состояние локального хранилища RepoA на удаленный сервер, используя команду push.

2.10. Действуя аналогично п. п. 2.7 и 2.8.3, синхронизировать с удаленным локальное хранилище RepoB (в нем не хватает commit-а с комментарием).

*Замечание. На данном этапе во всех трех хранилищах (локальных RepoA и RepoB и удаленном на GitHub) должна быть одинаковая линейная история из пяти — шести commit-ов.*

3. Изучить действия, связанные с ветвлениями и разрешением конфликтов.

*Замечание. Все действия выполняются в одном локальном хранилище, например, в RepoA.*

3.1. Добавить в программу печать произведения чисел и совершите commit.

На данном этапе программа может быть такой:

```
1  #include "sdt.h"
2
3  // This program is just an example one under VCS.
4  int main()
5  {
6      int a, b;
7      cout << "Enter A and B: ";
8      cin  >> a >> b;
9      cout << "A + B = " << a + b << '\n'
10         << "A - B = " << a - b << '\n'
11         << "A * B = " << a * b << '\n';
12 }
```

3.2. Создать новую ветвь (branch) под названием division. из пункта истории, в котором был добавлен комментарий над main().

3.3. В новой ветви повторить пункт 3.1, заменив умножение делением.

*Указание. Переключиться на ветвь можно, выбрав в контекстном меню commit-а, которым эта ветвь оканчивается, пункт «Switch/checkout to this». При создании ветви можно сразу установить флажок «Switch to new branch». Переключаться можно только при чистом (clean) хранилище, то есть, без изменений в рабочей копии.*

3.4. Переключиться обратно на ветвь master.

3.5. Выполнить слияние ветви division в ветвь master так, чтобы в последней оказался код для печати и произведения, и частного.

3.5.1. В журнале изменений в контекстном меню пункта-окончания ветви division выбрать пункт «Merge into "master"...» и начать слияние, не меняя настроек.



```

Theirs - REMOTE
1 #include <string>
2
3 // This program is just an example one under VCS.
4 int main()
5 {
6     int a, b;
7     cout << "Enter A and B: ";
8     cin >> a >> b;
9     cout << "A+B=" << a+b << "\n";
10    cout << "A-B=" << a-b << "\n";
11    cout << "A/B=" << a/b << "\n";
}

Mine - LOCAL
1 #include <string>
2
3 // This program is just an example one under VCS.
4 int main()
5 {
6     int a, b;
7     cout << "Enter A and B: ";
8     cin >> a >> b;
9     cout << "A+B=" << a+b << "\n";
10    cout << "A-B=" << a-b << "\n";
11    cout << "A*B=" << a*b << "\n";
12    cout << "A/B=" << a/b << "\n";
}

Merged - main.cpp
1 #include <string>
2
3 // This program is just an example one under VCS.
4 int main()
5 {
6     int a, b;
7     cout << "Enter A and B: ";
8     cin >> a >> b;
9     cout << "A+B=" << a+b << "\n";
10    cout << "A-B=" << a-b << "\n";
11    cout << "A*B=" << a*b << "\n";
12    cout << "A/B=" << a/b << "\n";
}

```

### 3.5.4. Завершить процедуру разрешения конфликтов.

*Указание.* Следует нажать на кнопку «Mark as resolved», чтобы отметить файл как избавленный от конфликтов, и закрыть программу для их разрешения.

3.5.5. Завершить слияние ветви division в ветвь master, написав осмысленный комментарий к слиянию и совершив commit.

3.5.6. Убедиться, что программа компилируется и верно работает. Если это не так, исправить все ошибки и добиться правильной работы. Совершить commit.

*Замечание-указание.* Ситуация, когда после слияния программа все-таки оказывается не вполне корректной, случается на практике довольно часто. В этом случае commit, созданный при слиянии, оказывается логически неправильным, он не имеет ценности без последующего исправления. В Git имеется возможность изменить (amend) уже совершенный commit, пока он не передан на сервер. Это делается при следующем commit-исправлении: следует установить флажок «Amend Last Commit» в диалоге commit — нового commit не появится, а вместо этого изменения будут приписаны предыдущему пункту истории. Можно воспользоваться данной возможностью при выполнении пункта

### 3.5.7. Передать все изменения всех ветвей в удаленное хранилище.

*Указание.* По умолчанию передаются только изменения текущей ветви, для передачи изменений всех ветвей следует отметить флажок «Push all branches» диалога push.

*Замечание.* В данном задании отрабатывается навык слияния ветвей, существующих только в локальном хранилище и вступивших в конфликт с введом единственного автора. Постоянно возникают и ситуации, когда одну и ту же ветвь, но в локальном и удаленном хранилище независимо изменяют разные авторы. В этом случае действия push и rebase приведут к конфликтам. Их разрешение выполняется совершенно аналогично.

## Задание самостоятельной работы

В соответствии с индивидуальным вариантом, определить и описать особенности реализации системы (подходы, технологии и т. п.). На основе описанных особенностей системы обосновать выбор вида архитектуры, наиболее подходящей для реализации данной системы. Произвести визуальное проектирование архитектуры системы (рекомендуется использовать программу Microsoft Visio). Добавить текстовое описание к архитектуре, поясняющее ее структуру и связи.

Перечень индивидуальных вариантов приведен в приложении А.

В соответствии с индивидуальным вариантом разработать программу на любом известном языке программирования. Добавить ее в систему контроля версий и отследить изменения.

Перечень индивидуальных вариантов приведен в приложении В.

## Контрольные вопросы

1. Что подразумевается под созданием архитектуры приложения?
2. Каково основное назначение архитектуры приложения?

3. Перечислите основные принципы разработки архитектуры программного обеспечения.
4. Перечислите основные типовые архитектурные стили.
5. Перечислите основные архетипы приложений.
6. Перечислите основные элементы, которые можно использовать для построения архитектуры программного обеспечения.
7. Что такое системы контроля версий (СКВ) и для решения каких задач они предназначены?
8. Объясните следующие понятия СКВ и их отношения: хранилище, commit, история, рабочая копия.
9. Что представляют собой и чем отличаются централизованные и децентрализованные СКВ? Приведите примеры СКВ каждого вида.
10. Опишите действия с СКВ при единоличной работе с хранилищем.
11. Опишите порядок работы с общим хранилищем в централизованной СКВ.
12. Что такое и зачем может быть нужна разность (diff)?
13. Что такое и зачем может быть нужно слияние (merge)?
14. Что такое конфликты (conflict) и каков процесс их разрешения (resolve)?
15. Поясните процесс синхронизации с общим хранилищем («обновления») в децентрализованной СКВ.
16. Что такое и зачем могут быть нужны ветви (branches)?
17. Объясните смысл действия rebase в СКВ Git.
18. Как и зачем можно игнорировать некоторые файлы при commit?

**Форма отчета:** отчет, защита работы.

### **Практическое занятие № 9-10**

**Тема:** Построение функциональных диаграмм IDEF0 и диаграмм потоков данных DFD

**Цель:** получение навыков создания и редактирования функциональных моделей в нотации IDEF0. Изучение основных характеристик и основ работы с DFD-моделями в графическом редакторе.

**Оборудование:** тетрадь, ручка, ПК

**Методические указания:** ознакомьтесь с теоретической частью, выполните задания, ответьте на контрольные вопросы

**Ход выполнения:**

**Краткие теоретические сведения**

**Функциональная диаграмма IDEF0**

#### **1. Основные сведения по методологии IDEF0**

Модель в нотации IDEF0 представляет собой совокупность иерархически упорядоченных и взаимосвязанных диаграмм. Каждая диаграмма является единицей описания системы и располагается на отдельном листе.

**Цель моделирования.** Модель не может быть построена без четко сформулированной цели. Пример цели: «Описать функциональность предприятия с целью написания спецификаций ИС».

**Точка зрения.** Точку зрения можно представить как взгляд человека, который видит систему в нужном для моделирования аспекте. Как правило, выбирается точка зрения человека, ответственного за моделируемую работу в целом. Цель и точка зрения документируются.

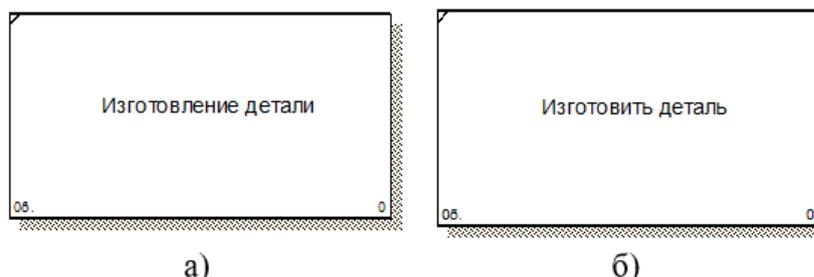
#### **Основные элементы IDEF0-модели**

В основе методологии IDEF0 лежат 4 основных понятия:

- функциональный блок;
- интерфейсная дуга (стрелка);
- декомпозиция;
- глоссарий.

#### **1. Функциональный блок**

Функциональные блоки обозначают поименованные процессы, функции или задачи, которые происходят в течение определенного времени и имеют распознаваемые результаты. Графически функциональные блоки изображаются в виде прямоугольников. Все блоки должны быть названы и определены. Имя функционального блока должно быть выражено сочетанием отглагольного существительного, обозначающего процесс, или глаголом:



Определение функционального блока заносится в глоссарий или словарь работ (Activity Dictionary).

Все функциональные блоки модели нумеруются. Номер состоит из префикса и числа. Может использоваться префикс любой длины, но обычно используется префикс А. Контекстная (корневая) работа (функциональный блок) имеет номер А0.

## 2. Интерфейсная дуга (стрелка – Arrow)

Взаимодействие функциональных блоков с внешним миром и между собой описывается в виде интерфейсных дуг (стрелок). Стрелки представляют собой некую информацию и обозначаются существительными (например, «Заготовка», «Изделие») или именуемыми сочетаниями (например, «Готовое изделие»). Все стрелки должны быть определены.

### В IDEF0 различают типы стрелок.

Каждая стрелка имеет свое расположение относительно функционального блока.



Вход (Input) – материал или информация, которые используются или преобразуются работой для получения результата (выхода). Стрелка Input рисуется входящей в левую грань работы.

Управление (Control) – правила, стратегии, процедуры или стандарты, которыми руководствуется работа. Каждая работа должна иметь хотя бы одну стрелку управления. Рисуется как входящая в верхнюю грань работы.

Выход (Output) – материал или информация, которые производятся работой. Каждая работа должна иметь хотя бы одну стрелку выхода. Работа без результата не имеет смысла и не должна моделироваться. Изображается исходящей из правой грани работы.

Механизм (Mechanism) – ресурсы, которые выполняют работу, например, персонал предприятия, станки, устройства и т.д. Рисуется как входящая в нижнюю грань работы.

## 3. Декомпозиция

Разбиение системы на крупные фрагменты – функции, функции – на подфункции и т.д. до конкретных процедур.

### Модель может содержать 4 типа диаграмм:

- контекстную (в каждой модели может быть только 1 контекстная диаграмма);
- декомпозиции;
- дерева узлов;
- только для экспозиции (FEO).

Контекстная диаграмма является вершиной древовидной структуры диаграмм и представляет собой общее описание системы и ее взаимодействия с внешней средой.

После описания системы в целом проводится разбиение ее на крупные фрагменты. Этот процесс называется функциональной декомпозицией, а диаграммы, которые описывают каждый фрагмент и взаимодействие фрагментов – диаграммами декомпозиции. После декомпозиции контекстной

диаграммы проводится декомпозиция каждого большого фрагмента системы на более мелкие и т.д., до достижения нужного уровня подробности описания.

Диаграмма дерева узлов показывает иерархическую зависимость работ, но не взаимосвязи между работами.

Диаграммы для экспозиции (FEO) строятся для иллюстрации отдельных фрагментов модели, для иллюстрации альтернативной точки зрения либо для специальных целей.

Все диаграммы имеют нумерацию. Контекстная диаграмма имеет номер А-0, декомпозиция контекстной диаграммы – номер А), остальные диаграммы-декомпозиции – номера по соответствующему узлу (например, А1, А2, А21 и т.д.).

## Диаграмма потоков данных DFD

### 1. Модель потоков данных

DFD – data flow diagrams – диаграммы потоков данных – методология графического структурного анализа, описывающая внешние по отношению к системе источники и адресаты данных, логические функции, потоки данных и хранилища данных, к которым осуществляется доступ.

Диаграмма потоков данных – один из основных инструментов структурного анализа и проектирования информационных систем, существовавших до широкого распространения UML.

Для описания диаграмм DFD используются две нотации – Йордана (Yourdon) и Гейна-Сарсона (Gane-Sarson), отличающиеся синтаксисом.

### 2. Основные элементы информационной модели логического уровня

Согласно DFD источники информации (внешние сущности) порождают информационные потоки (потоки данных), переносящие информацию к подсистемам или процессам. Те в свою очередь преобразуют информацию и порождают новые потоки, которые переносят информацию к другим процессам или подсистемам, накопителям данных или внешним сущностям – потребителям информации.

При построении диаграмм различают элементы графической нотации, представленные в табл. 1.

Таблица 1 – Элементы графической нотации DFD

Наименование	Нотация Йордана	Нотация Гейна-Сарсона
Поток данных	ИМЯ →	ИМЯ →
Процесс (система, подсистема)	ИМЯ номер	номер ИМЯ
Накопитель данных	ИМЯ	но- мер ИМЯ
Внешняя сущность	ИМЯ	ИМЯ

**Поток данных** определяет информацию (материальный объект), передаваемую через некоторое соединение от источника к приемнику. Реальный поток данных может быть информацией, передаваемой по кабелю между двумя устройствами, пересылаемыми по почте письмами, магнитными лентами или дискетами, переносимыми с одного компьютера на другой и т. д.

Каждый поток данных имеет имя, отражающее его содержание. Направление стрелки показывает направление потока данных. Иногда информация может двигаться в одном направлении, обрабатываться и возвращаться назад в ее источник. Такая ситуация может моделироваться либо двумя различными потоками, либо одним – двунаправленным.

На диаграммах IDEF0 потоки данных соответствуют входам и выходам, но в отличие от IDEF0 стрелки потоков на DFD могут отображаться входящими и выходящими из любой грани внешней сущности, процесса или накопителя данных.

**Процесс** (в IDEF0 – функция, работа) представляет собой преобразование входных потоков данных в выходные в соответствии с определенным алгоритмом.

Каждый процесс должен иметь имя в виде предложения с глаголом в неопределенной форме (вычислить, рассчитать, проверить, определить, создать, получить), за которым следуют существительные в винительном падеже, например:

- «Ввести сведения о клиентах»;
- «Рассчитать допустимую скорость»;
- «Сформировать ведомость допустимых скоростей».

Номер процесса служит для его идентификации и ставится с учетом декомпозиции. Вложенность процессов обозначается через точку.

Преобразование информации может показываться как с точки зрения процессов, так и с точки зрения систем и подсистем. Если вместо имени процесса «Рассчитать допустимую скорость» написать «Подсистема расчета допустимых скоростей», тогда этот блок на диаграмме стоит рассматривать, как подсистему.

**Накопитель (хранилище) данных** представляет собой абстрактное устройство для хранения информации, которую можно в любой момент поместить в накопитель и через некоторое время извлечь, причем способы помещения и извлечения могут быть любыми.

Накопитель данных может быть реализован физически в виде ящика в картотеке, области в оперативной памяти, файла на магнитном носителе и т.д.

Накопителю обязательно должно даваться уникальное имя и номер в пределах всей модели (всего набора диаграмм). Имя накопителя выбирается из соображения наибольшей информативности для разработчика. Например, если в качестве накопителей выступают таблицы проектируемой базы данных, тогда в качестве имен накопителей рекомендуется использовать имена таблиц. Таким образом, накопитель данных может представлять собой всю базу данных целиком, совокупность таблиц или отдельную таблицу. Такое представление накопителей в дальнейшем облегчит построение информационной модели системы.

**Внешняя сущность** (терминатор) представляет собой материальный объект или физическое лицо, выступающие как источник или приемник информации (например, заказчики, персонал, программа, склад, инструкция). Внешние сущности на DFD по смыслу соответствуют управлению и механизмам, отображаемым на контекстной диаграмме IDEF0.

Определение некоторого объекта, субъекта или системы в качестве внешней сущности указывает на то, что она находится за пределами границ проектируемой информационной системы. В связи с этим внешние сущности, как правило, отображаются только на контекстной диаграмме DFD. В процессе анализа и проектирования некоторые внешние сущности могут быть перенесены на диаграммы декомпозиции, если это необходимо, или, наоборот, часть процессов (подсистем) может быть представлена как внешняя сущность.

### **Методика выполнения**

#### **Функциональная диаграмма IDEF0**

В качестве примера рассматривается процесс выполнения студентом курсовой работы (курсового проекта).

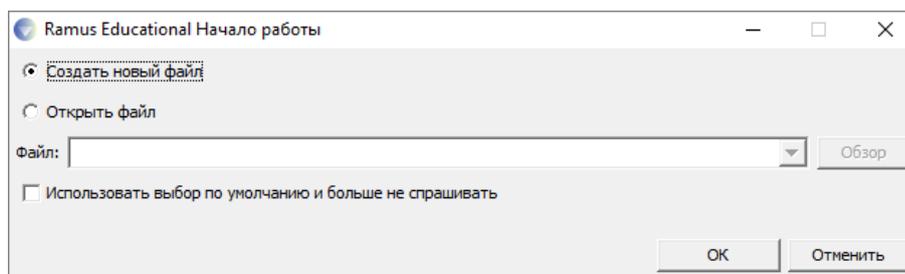
Программное обеспечение «Ramus» предназначено для использования в проектах, в которых необходимо описание бизнес-процессов предприятия. «Ramus» поддерживает методологии моделирования бизнес-процессов IDEF0 и DFD, а также имеет ряд дополнительных возможностей, призванных удовлетворить потребности команд разработчиков систем управления предприятиями.

«Ramus» обладает гибкими возможностями построения отчетности по графическим моделям, позволяющие создавать отчеты в форме документов, регламентирующих деятельность предприятия.

Ramus Educational имеет достаточно интуитивный интерфейс пользователя, позволяющий быстро и просто создавать сложные модели.

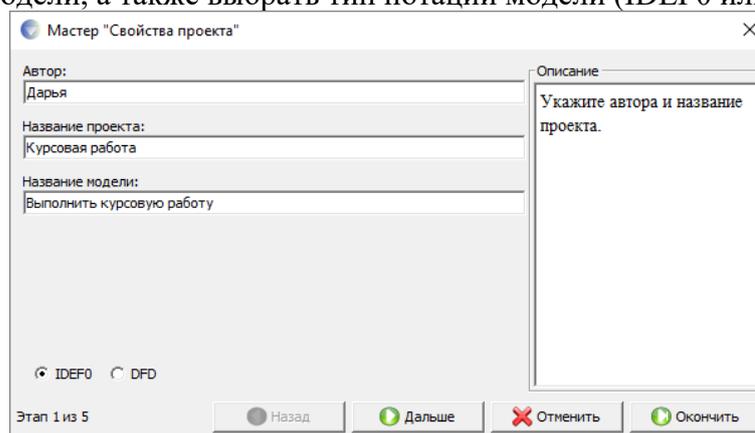
#### **1 Начало работы**

1. Запустите программу Ramus Educational. В появившемся окне предлагается создать новый проект или открыть уже существующий.



2. После нажатия на кнопку «ОК» осуществляется запуск мастера проекта.

– На первом шаге (рис. 5) в соответствующие поля необходимо внести сведения об авторе, названии проекта и модели, а также выбрать тип нотации модели (IDEF0 или DFD).



– На втором шаге вводится название организации, использующей данный проект.

– На третьем – дается краткое описание будущего проекта.

– Четвертый шаг позволяет создать несколько основных классификаторов (в данном случае можно пропустить этот шаг). Так как модели процессов реальных предприятий могут содержать значительное количество объектов (документы, персонал, функции и т.д.), то в Ramus предусмотрена возможность упорядочено хранить информацию об этих объектах в виде системы классификаторов. Классификация объектов упрощает поиск и обработку информации об объектах модели, а также и об объектах непосредственно не представленных на диаграммах процессов, но относящихся к процессам предприятия.

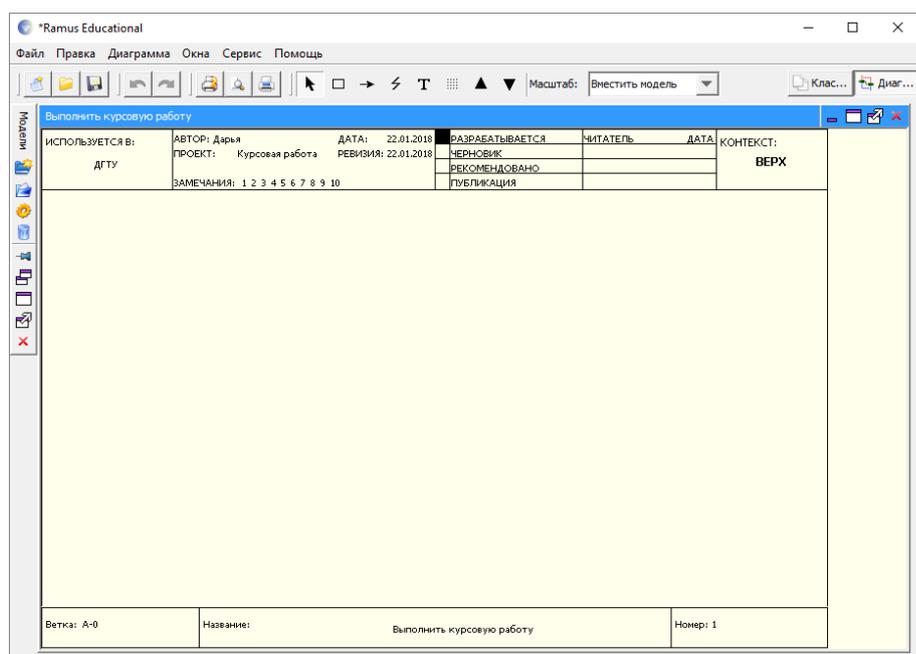
– На пятом, заключительном, предлагается выбрать те из созданных классификаторов, элементы которых будут содержаться в перечне собственников процессов (пропустить данный шаг).

При необходимости можно завершить работу мастера, нажав кнопку «Окончить».

После завершения работы мастера, откроется рабочее пространство «Диаграммы», в котором можно приступить к рисованию графической модели (рис. 6). В верхней части приводятся сведения о проекте, введенные пользователем посредством мастера диаграмм.

Программа Ramus Educational обладает гибким графическим интерфейсом, который можно настроить под нужды и предпочтения конкретного пользователя: ненужные окна можно закрыть/свернуть; можно менять их размеры и месторасположение; также можно группировать два и более окон в одном, при этом содержимое вложенных окон будет размещено на вкладках общего окна (данный функционал возможен не для всех комбинаций окон).

3. Сохраните созданную модель, выбрав опцию меню «Файл» – «Сохранить как».



## 2 Создание контекстной диаграммы

1. На панели инструментов выберите пиктограмму функции (  ) и мышью укажите месторасположение на рабочем пространстве.

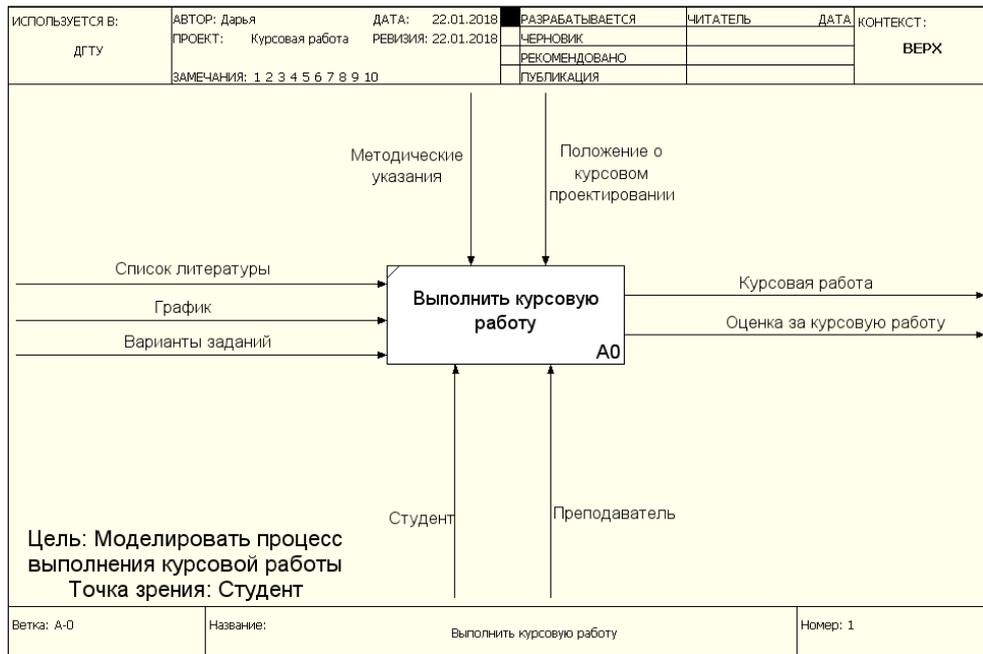
1. Дайте данному функциональному блоку имя «Выполнить курсовую работу». Для этого дважды щелкните внутри блока.

2. Используя пиктограмму панели инструментов (  ), создайте стрелки на контекстной диаграмме согласно Таблица 1.

Таблица 2 – Контекстная диаграмма

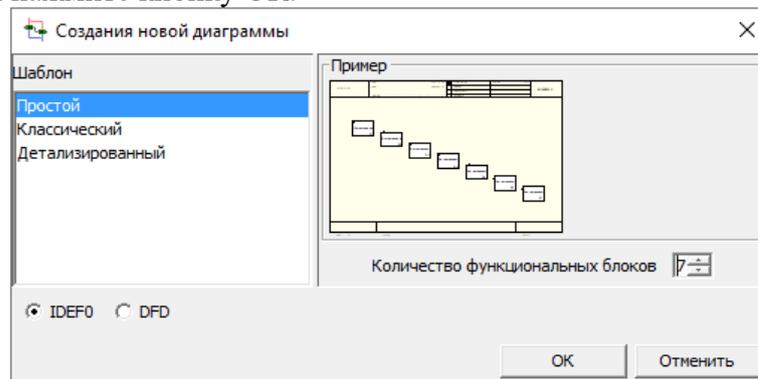
Имя стрелки (Arrow Name)	Определение стрелки (Arrow Definition)	Тип стрелки (Arrow Type)
График	График консультаций и сроки сдачи	Input
Список литературы	Источники информации для выполнения курсовой работы	Input
Варианты заданий	Список заданий на курсовую работу, подлежащий распределению между студентами	Input
Методические указания	Документ, содержащий указания по выполнению курсовой работы, описывающий содержание ее частей и основные требования	Control
Положение о курсовом проектировании	Документ, отражающий организационные требования по выполнению и сдаче курсовой работы	Control
Курсовая работа	Документ, являющийся основанием для получения оценки	Output
Оценка за курсовую работу	Результат выполнения курсовой работы	Output
Студент	Тот, кто выполняет курсовую работу	Mechanism

3. В результате должна получиться контекстная диаграмма.

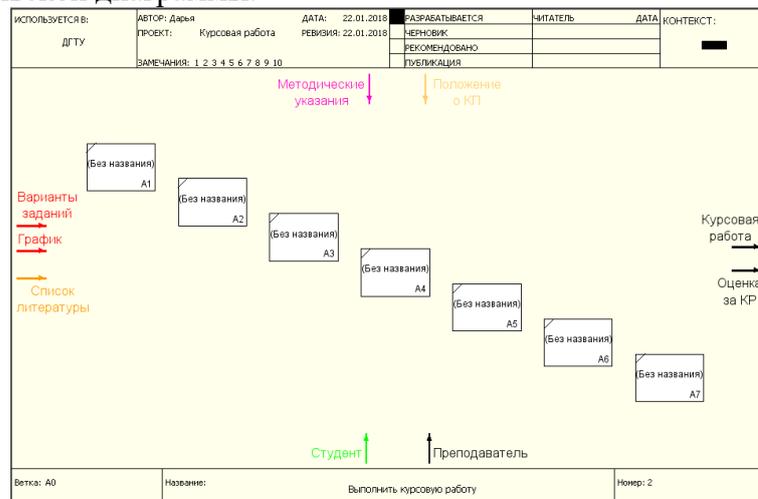


### 3 Создание диаграммы декомпозиции

1. Выберите в палитре инструментов кнопку перехода на нижний уровень , в диалоговом окне «Создание новой диаграммы» установите количество функциональных блоков 7, укажите тип диаграммы (IDEF0) и нажмите кнопку ОК.

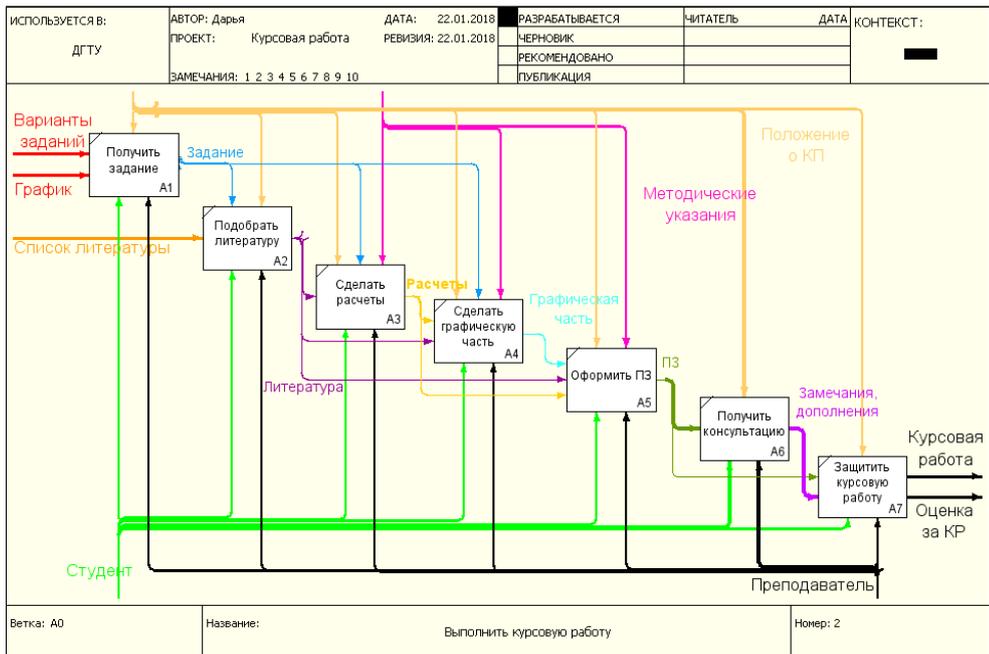


2. Автоматически будет создана диаграмма первого уровня декомпозиции с перенесенными в нее потоками родительской диаграммы.



3. Выделите первую работу (функциональный блок), затем двойным щелчком мыши или, выбрав в контекстном меню пункт «Редактировать активный элемент», откройте окно свойств и внесите имя работы. Повторите операцию для оставшихся работ.

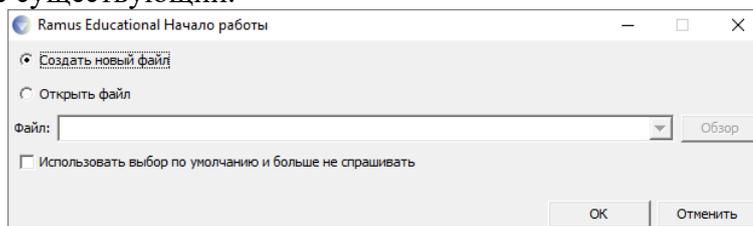
4. Выделив необходимый поток (стрелку) и, удерживая левую клавишу мыши, соедините его требуемым образом (через вход, управление, механизм или выход) с соответствующим функциональным блоком. В результате должна получиться детализирующая диаграмма.



## Диаграмма потоков данных DFD

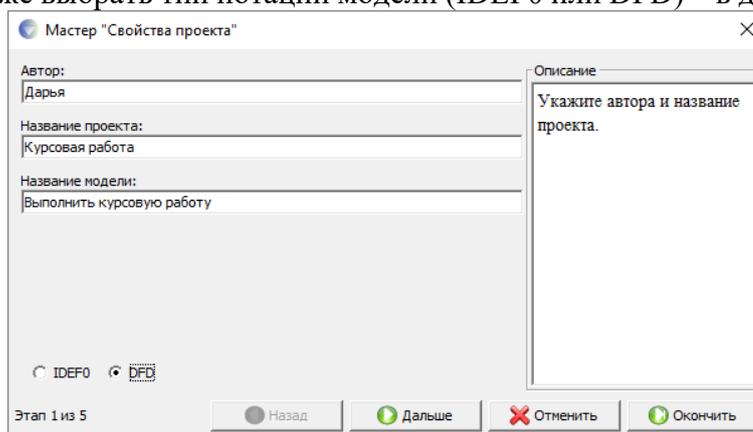
### 1 Начало работы

1. Запустите программу Ramus Educational. В появившемся окне предлагается создать новый проект или открыть уже существующий.



2. После нажатия на кнопку «ОК» осуществляется запуск мастера проекта.

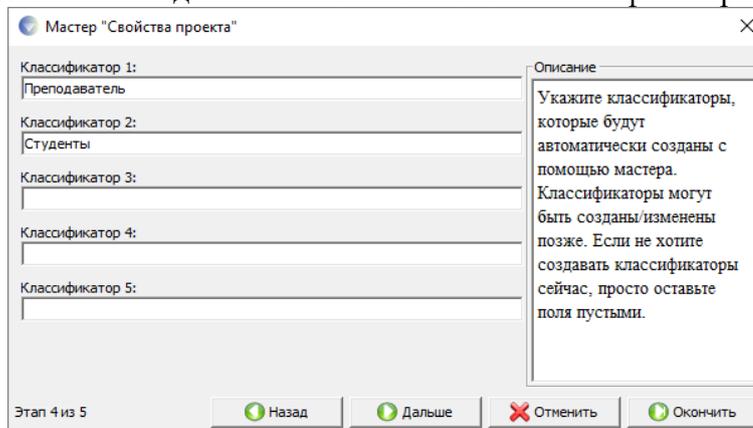
– На первом шаге в соответствующие поля необходимо внести сведения об авторе, названии проекта и модели, а также выбрать тип нотации модели (IDEF0 или DFD) – в данном случае – DFD.



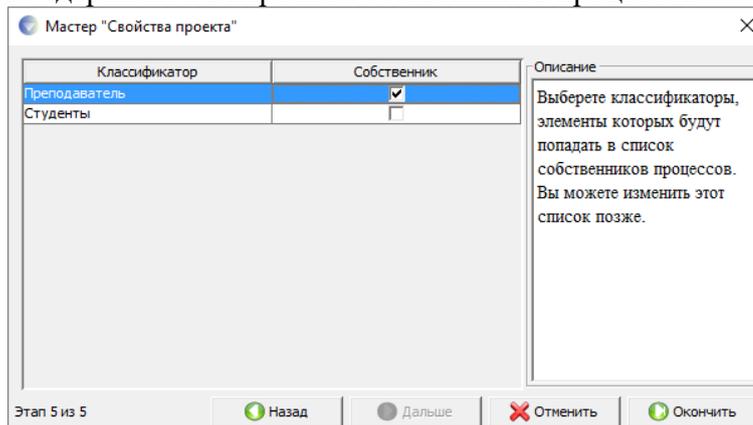
– На втором шаге вводится название организации, использующей данный проект.

– На третьем – дается краткое описание будущего проекта.

– Четвертый шаг позволяет создать несколько основных классификаторов.

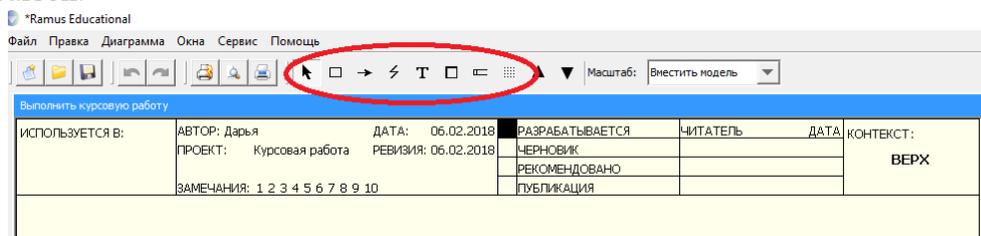


– На пятом, заключительном, предлагается выбрать те из созданных классификаторов, элементы которых будут содержаться в перечне собственников процессов.



При необходимости можно завершить работу мастера, нажав кнопку «Окончить».

После завершения работы мастера, откроется рабочее пространство «**Диаграммы**», в котором можно приступить к построению графической модели. На панели инструментов, в верхней части окна рабочего пространства программы, содержатся элементы диаграммы потоков данных в нотации Gane-Sarson.

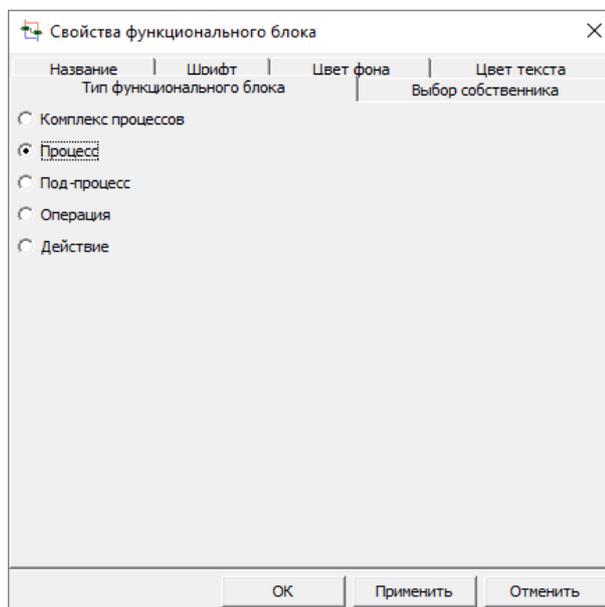


3. Сохраните созданную модель, выбрав опцию меню «**Файл**» – «**Сохранить как**».

## 2 Создание контекстной диаграммы

2. На панели инструментов выберите инструмент создания процесса () и мышью укажите месторасположение на рабочем пространстве нового процесса.

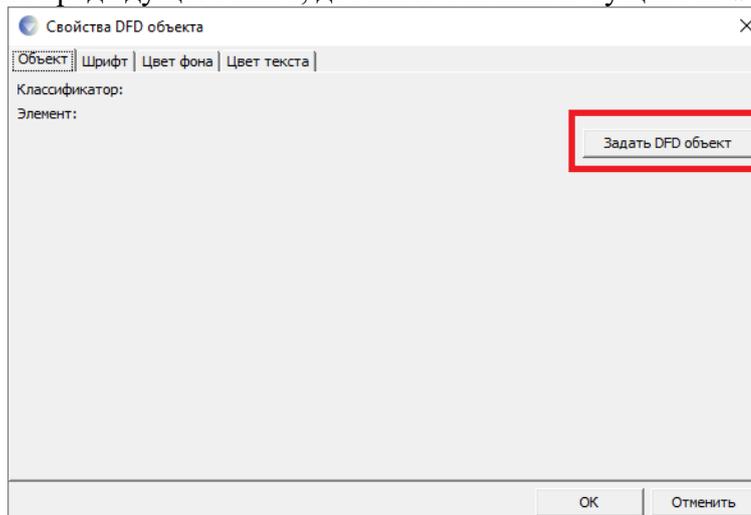
3. Выделив процесс, выберите в контекстном меню опцию «**Редактировать активный элемент**». В появившемся диалоговом окне на вкладке «**Название**» присвойте процессу имя «*Выполнить курсовую работу*»; на вкладке «**Тип функционального блока**» укажите тип элемента – «**Процесс**».

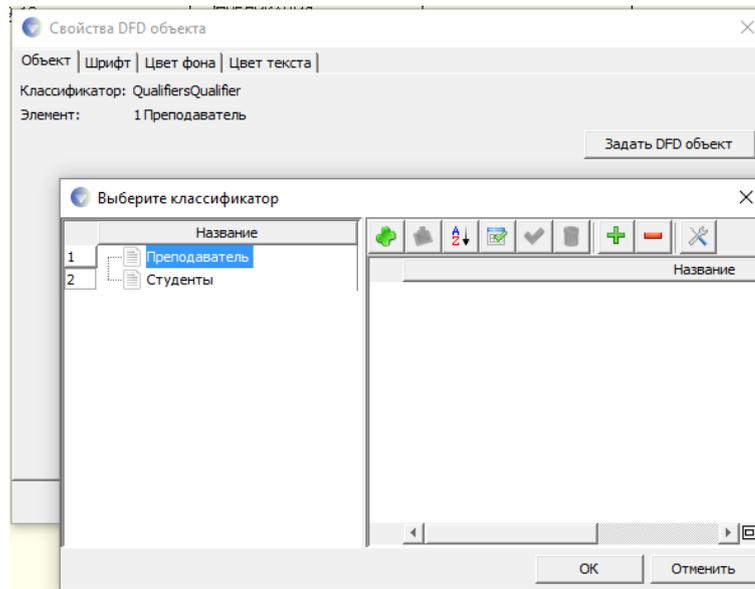


4. На панели инструментов выберите инструмент создания внешней сущности () и мышью укажите произвольное ее месторасположение в области построения.

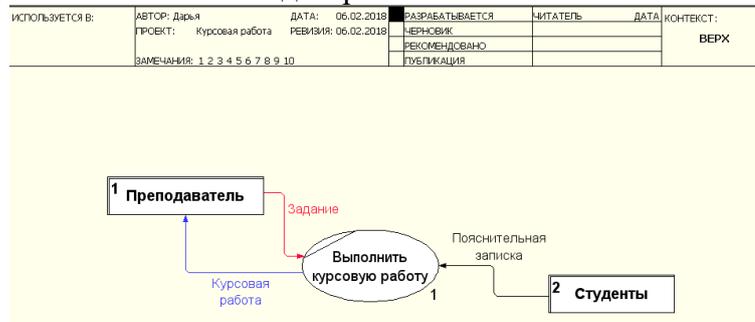
5. В контекстном меню созданной внешней сущности выберите опцию «**Редактировать активный элемент**», на вкладке «**Объект**» нажмите «**Задать DFD объект**», после чего, в появившемся окне выделите классификатор «*Преподаватель*» и нажмите «**ОК**».

6. Повторяя действия предыдущего шага, добавьте внешнюю сущность «*Студенты*».





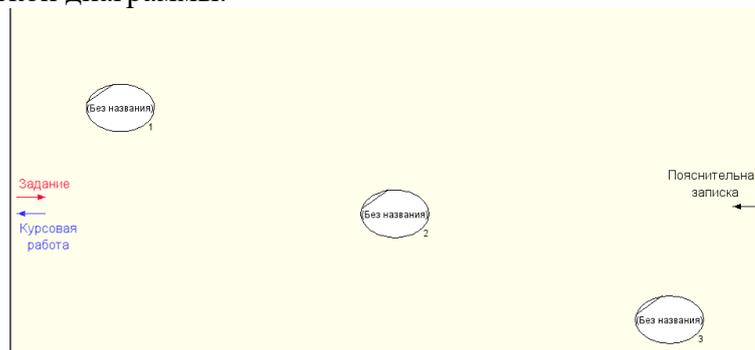
7. Выбра на панели инструментов элемент , создайте стрелки на контекстной диаграмме. В результате должна получиться контекстная диаграмма.



### 3 Создание диаграммы декомпозиции

1. Выберите в палитре инструментов кнопку перехода на нижний уровень , в диалоговом окне «Создание новой диаграммы» установите количество функциональных блоков 3, укажите тип диаграммы (DFD) и нажмите кнопку ОК.

2. Автоматически будет создана диаграмма первого уровня декомпозиции с перенесенными в нее потоками родительской диаграммы.



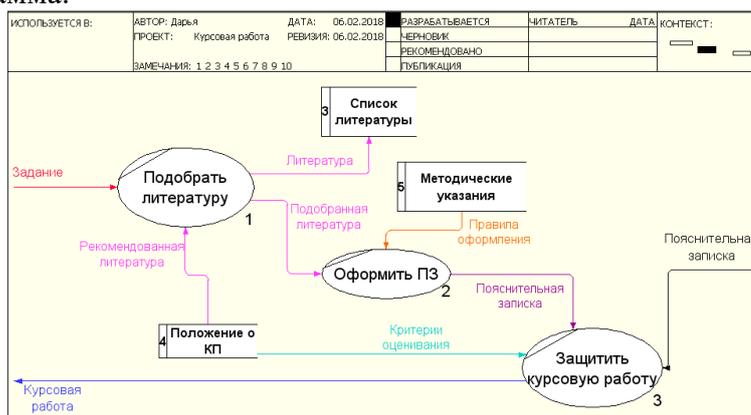
3. Двойным щелчком мыши на одном из процессов, или, выбрав в контекстном меню процесса пункт «**Редактировать активный элемент**», откройте окно редактирования свойств и задайте процессу имя. Повторите операцию для оставшихся процессов.

4. Добавьте недостающие классификаторы для задания DFD объектов хранилищам данным. Для этого в меню выберите Окна -> Показать окно -> Классификаторы.

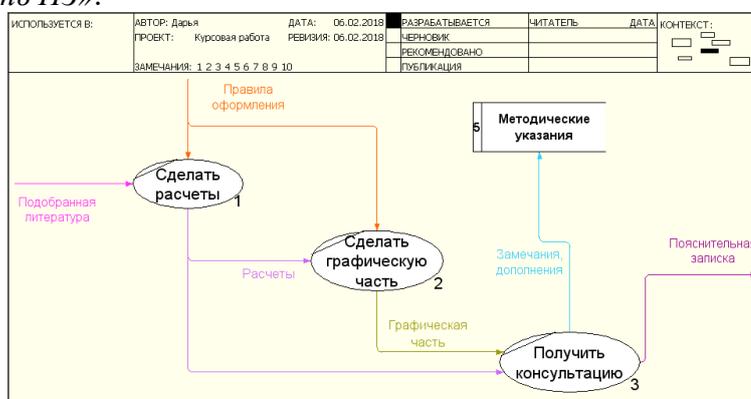
После этого нажмите на кнопку . Название классификатора можно ввести в созданную строку, дважды, медленно кликнув мышью по строке, или же нажав клавишу F2, предварительно выделив нужную строку мышью.

5. Добавьте хранилища данных, воспользовавшись кнопкой  палитры инструментов.

6. Выделив необходимый поток (стрелку) и, удерживая левую клавишу мыши, соедините его требуемым образом с соответствующим процессом. В результате должна получиться детализирующая диаграмма.



7. На основе описанных выше действий постройте диаграмму декомпозиций второго уровня для процесса «Оформить ПЗ».



### Задание практической работы

По образцу построить диаграммы IDEF0 и DFD.

### Задание самостоятельной работы

В соответствии с индивидуальным вариантом, построить диаграммы IDEF0 и DFD.

Перечень индивидуальных вариантов приведен в приложении А.

Отчет по практическому занятию выполняется в формате MS Word, который содержит экранные формы моделей согласно заданию.

### Требования к построению модели

1. На контекстной диаграмме необходимо указать точку зрения и цель моделирования.
2. Количество блоков любой декомпозиции не менее 3-х и не более 9.
3. Количество декомпозиций – 3 уровня декомпозиции.

### Контрольные вопросы

1. Каковы цели функционального моделирования?
2. Назовите основные компоненты функциональной модели.
3. Какие виды интерфейсных дуг различают в IDEF0?
4. Для чего нужна цель и точка зрения?
5. Что такое функциональный блок?
6. Какие виды диаграмм может содержать функциональная модель?
7. Каково назначение стандарта DFD?
8. В чем основные отличия стандартов IDEF0 и DFD?
9. Каким образом в MS Visio создается схема DFD? Какие для этого используются нотации?
10. Какова роль основных элементов в стандарте DFD?

**Форма отчета:** тетрадь, защита работы.

## Практическое занятие № 11-12

**Тема:** Построение диаграммы Вариантов использования и диаграммы Классов

**Цель:** изучение основ создания диаграмм прецедентов (вариантов использования) на языке UML. Изучение основ создания диаграмм классов на языке UML, получение навыков построения

диаграмм классов, применение приобретенных навыков для построения объектно-ориентированных моделей определенной предметной области.

**Оборудование:** тетрадь, ручка, ПК

**Методические указания:** ознакомьтесь с теоретической частью, выполните задания, ответьте на контрольные вопросы

**Ход выполнения:**

**Теоретические сведения**

**Построение диаграммы Вариантов использования**

### **1. Общие сведения о языке UML**

Язык UML представляет собой общецелевой язык визуального моделирования, который разработан для спецификации, визуализации, проектирования и документирования компонентов программного обеспечения, бизнес-процессов и других систем.

Язык UML одновременно является простым и мощным средством моделирования, который может быть эффективно использован для построения концептуальных, логических и графических моделей сложных систем самого различного целевого назначения.

В языке UML используется **четыре основных вида графических конструкций:**

– **Значки или пиктограммы.** Значок представляет собой графическую фигуру фиксированного размера и формы. Примерами значков могут служить окончания связей элементов диаграмм или некоторые другие дополнительные обозначения.

– **Графические символы на плоскости.** Такие двумерные символы изображаются с помощью некоторых геометрических фигур и могут иметь различную высоту и ширину с целью размещения внутри этих фигур других конструкций языка UML.

Наиболее часто внутри таких символов помещаются строки текста, которые уточняют семантику или фиксируют отдельные свойства соответствующих элементов языка UML. Информация, содержащаяся внутри фигур, имеет важное значение для конкретной модели проектируемой системы, поскольку регламентирует реализацию соответствующих элементов в программном коде.

– **Пути,** которые представляют собой последовательности из отрезков линий, соединяющих отдельные графические символы. При этом концевые точки отрезков линий должны обязательно соприкасаться с геометрическими фигурами, служащими для обозначения вершин диаграмм, как принято в теории графов. С концептуальной точки зрения путям в языке UML придается особое значение, поскольку они являются простыми топологическими сущностями.

– **Строки текста.** Служат для представления различных видов информации в некоторой грамматической форме. Предполагается, что каждое использование строки текста должно соответствовать синтаксису в нотации языка UML, посредством которого может быть реализован грамматический разбор этой строки.

При графическом изображении диаграмм следует придерживаться следующих **основных рекомендаций:**

1. Каждая диаграмма должна служить законченным представлением соответствующего фрагмента моделируемой предметной области.

2. Все сущности на диаграмме модели должны быть одного концептуального уровня. Здесь имеется в виду согласованность не только имен одинаковых элементов, но и возможность вложения отдельных диаграмм друг в друга для достижения полноты представлений.

3. Вся информация о сущностях должна быть явно представлена на диаграммах. Речь идет о том, что, хотя в языке UML при отсутствии некоторых символов на диаграмме могут быть использованы их значения по умолчанию (например, в случае неявного указания видимости атрибутов и операций классов), необходимо стремиться к явному указанию свойств всех элементов диаграмм.

4. Диаграммы не должны содержать противоречивой информации. Противоречивость модели может служить причиной серьезных проблем при ее реализации и последующем использовании на практике. Например, наличие замкнутых путей при изображении отношений агрегирования или композиции приводит к ошибкам в программном коде, который будет реализовывать соответствующие классы. Наличие элементов с одинаковыми именами и различными атрибутами свойств в одном пространстве имен также приводит к неоднозначной интерпретации и может служить источником проблем.

5. Диаграммы не следует перегружать текстовой информацией. Принято считать, что визуализация модели является наиболее эффективной, если она содержит минимум пояснительного текста.

6. Каждая диаграмма должна быть самодостаточной для правильной интерпретации всех ее элементов и понимания семантики всех используемых графических символов.

7. Количество типов диаграмм для конкретной модели приложения не является строго фиксированным.

## **2. Диаграмма вариантов использования (usecase diagram)**

Разработка диаграммы вариантов использования преследует цели:

1. Определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы.

2. Сформулировать общие требования к функциональному поведению проектируемой системы.

3. Разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей.

4. Подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью так называемых вариантов использования.

При этом **актером (actor)** или действующим лицом называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик.

В свою очередь, **вариант использования (usecase)** служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой.

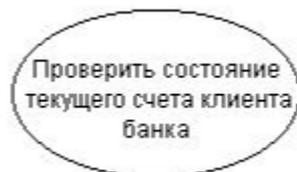
В самом общем случае, диаграмма вариантов использования представляет собой граф специального вида, который является графической нотацией для представления конкретных вариантов использования, актеров, возможно некоторых интерфейсов, и отношений между этими элементами.

**Вариант использования (use case)** – конструкция или стандартный элемент языка UML, который применяется для спецификации общих особенностей поведения системы или любой другой сущности предметной области без рассмотрения внутренней структуры этой сущности. Каждый вариант использования определяет последовательность действий, которые должны быть выполнены проектируемой системой при взаимодействии ее с соответствующим актером.

Диаграмма вариантов может дополняться пояснительным текстом, который раскрывает смысл или семантику составляющих ее компонентов.

Такой пояснительный текст получил название **примечания или сценария**.

Отдельный вариант использования обозначается на диаграмме эллипсом, внутри которого содержится его краткое название или имя в форме глагола с пояснительными словами.



**Актер (actor)** представляет собой любую внешнюю по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач. При этом актеры служат для обозначения согласованного множества ролей, которые могут играть пользователи в процессе взаимодействия с проектируемой системой. Каждый актер может рассматриваться как некая отдельная роль относительно конкретного варианта использования. Стандартным графическим обозначением актера на диаграммах является фигурка «человечка», под которой записывается конкретное имя актера.



**Интерфейс (interface)** служит для спецификации параметров модели, которые видимы извне без указания их внутренней структуры. В языке UML интерфейс является классификатором и характеризует только ограниченную часть поведения моделируемой сущности. Применительно к диаграммам вариантов использования, интерфейсы определяют совокупность операций, которые обеспечивают необходимый набор сервисов или функциональности для актеров. Интерфейсы не могут содержать ни атрибутов, ни состояний, ни направленных ассоциаций. Они содержат только операции без указания особенностей их реализации.

На диаграмме вариантов использования интерфейс изображается в виде маленького круга, рядом с которым записывается его имя.



В качестве имени может быть существительное, которое характеризует соответствующую информацию или сервис (например, «датчик», «сирена», «видеокамера»), но чаще строка текста (например, «запрос к базе данных», «форма ввода», «устройство подачи звукового сигнала»).

Если имя записывается на английском, то оно должно начинаться с заглавной буквы I, например, ISecureInformation, ISensor.

Графический символ отдельного интерфейса может соединяться на диаграмме сплошной линией с тем вариантом использования, который его поддерживает. Сплошная линия в этом случае указывает на тот факт, что связанный с интерфейсом вариант использования должен реализовывать все операции, необходимые для данного интерфейса, а возможно и больше.

Кроме этого, интерфейсы могут соединяться с вариантами использования пунктирной линией со стрелкой, означающей, что вариант использования предназначен для спецификации только того сервиса, который необходим для реализации данного интерфейса.



**Примечания (notes)** в языке UML предназначены для включения в модель произвольной текстовой информации, имеющей непосредственное отношение к контексту разрабатываемого проекта. В качестве такой информации могут быть комментарии разработчика (например, дата и версия разработки диаграммы или ее отдельных компонентов), ограничения (например, на значения отдельных связей или экземпляры сущностей) и помеченные значения.

Применительно к диаграммам вариантов использования примечание может носить самую общую информацию, относящуюся к общему контексту системы.

Графически примечания обозначаются прямоугольником с «загнутым» верхним правым уголком. Внутри прямоугольника содержится текст примечания. Примечание может относиться к любому элементу диаграммы, в этом случае их соединяет пунктирная линия. Если примечание относится к нескольким элементам, то от него проводятся, соответственно, несколько линий. Разумеется, примечания могут присутствовать не только на диаграмме вариантов использования, но и на других канонических диаграммах.



### 3. Отношения на диаграмме вариантов использования

Между компонентами диаграммы вариантов использования могут существовать различные отношения, которые описывают взаимодействие экземпляров одних актеров и вариантов использования с экземплярами других актеров и вариантов.

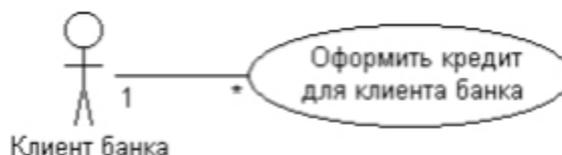
Один актер может взаимодействовать с несколькими вариантами использования. В этом случае этот актер обращается к нескольким сервисам данной системы. В свою очередь один вариант использования может взаимодействовать с несколькими актерами, предоставляя для всех них свой сервис. Следует заметить, что два варианта использования, определенные для одной и той же сущности, не могут взаимодействовать друг с другом, поскольку каждый из них самостоятельно описывает законченный вариант использования этой сущности. Более того, варианты использования всегда предусматривают некоторые сигналы или сообщения, когда взаимодействуют с актерами за пределами системы. В то же время могут быть определены другие способы для взаимодействия с элементами внутри системы.

В языке UML имеется несколько стандартных **видов отношений между актерами и вариантами использования**:

#### 1. Отношение ассоциации (association relationship)

Отношение ассоциации является одним из фундаментальных понятий в языке UML и в той или иной степени используется при построении всех графических моделей систем в форме канонических диаграмм.

Применительно к диаграммам вариантов использования оно служит для обозначения специфической роли актера в отдельном варианте использования. Другими словами, ассоциация специфицирует семантические особенности взаимодействия актеров и вариантов использования в графической модели системы. Таким образом, это отношение устанавливает, какую конкретную роль играет актер при взаимодействии с экземпляром варианта использования. На диаграмме вариантов использования, так же, как и на других диаграммах, отношение ассоциации обозначается сплошной линией между актером и вариантом использования. Эта линия может иметь дополнительные условные обозначения, такие, например, как имя и кратность.



#### 2. Отношение расширения (extend relationship)

Отношение расширения определяет взаимосвязь экземпляров отдельного варианта использования с более общим вариантом, свойства которого определяются на основе способа совместного объединения данных экземпляров.

Так, если имеет место отношение расширения от варианта использования А к варианту использования В, то это означает, что свойства экземпляра варианта использования В могут быть дополнены благодаря наличию свойств у расширенного варианта использования А.

Отношение расширения между вариантами использования обозначается пунктирной линией со стрелкой (вариант отношения зависимости), направленной от того варианта использования, который является расширением для исходного варианта использования. Данная линия со стрелкой помечается ключевым словом «extend» («расширяет»)



Отношение расширения отмечает тот факт, что один из вариантов использования может присоединять к своему поведению некоторое дополнительное поведение, определенное для другого варианта использования.

Один из вариантов использования может быть расширением для нескольких базовых вариантов, а также иметь в качестве собственных расширений несколько других вариантов. Базовый вариант использования может дополнительно никак не зависеть от своих расширений.

### 3. Отношение обобщения (generalization relationship)

Отношение обобщения служит для указания того факта, что некоторый вариант использования А может быть обобщен до варианта использования В.

В этом случае вариант А будет являться специализацией варианта В. При этом В называется предком или родителем по отношению А, а вариант А – потомком по отношению к варианту использования В. Следует подчеркнуть, что потомок наследует все свойства и поведение своего родителя, а также может быть дополнен новыми свойствами и особенностями поведения.

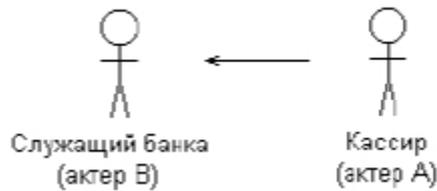
Графически данное отношение обозначается сплошной линией со стрелкой в форме незакрашенного треугольника, которая указывает на родительский вариант использования. Эта линия со стрелкой имеет специальное название – стрелка «обобщение».



Отношение обобщения между вариантами использования применяется в том случае, когда необходимо отметить, что дочерние варианты использования обладают всеми атрибутами и особенностями поведения родительских вариантов. При этом дочерние варианты использования участвуют во всех отношениях родительских вариантов. В свою очередь, дочерние варианты могут наделяться новыми свойствами поведения, которые отсутствуют у родительских вариантов использования, а также уточнять или модифицировать наследуемые от них свойства поведения.

Между отдельными актерами также может существовать отношение обобщения. Данное отношение является направленным и указывает на факт специализации одних актеров относительно других. Например, отношение обобщения от актера А к актеру В отмечает тот факт, что каждый экземпляр актера А является одновременно экземпляром актера В и обладает всеми его свойствами. В этом случае актер В является родителем по отношению к актеру А, а актер А, соответственно, потомком актера В. При этом актер А обладает способностью играть такое же множество ролей, что и актер В.

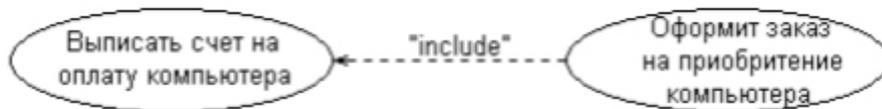
Графически данное отношение также обозначается стрелкой обобщения, т. е. сплошной линией со стрелкой в форме незакрашенного треугольника, которая указывает на родительского актера.



#### 4. Отношение включения (include relationship)

Отношение включения между двумя вариантами использования указывает, что некоторое заданное поведение для одного варианта использования включается в качестве составного компонента последовательность поведения другого варианта использования. Данное отношение является направленным бинарным отношением в том смысле, что пара экземпляров вариантов использования всегда упорядочена в отношении включения.

Отношение включения, направленное от варианта использования А к варианту использования В, указывает, что каждый экземпляр варианта А включает в себя функциональные свойства, заданные для варианта В. Эти свойства специализируют поведение соответствующего варианта А на данной диаграмме. Графически данное отношение обозначается пунктирной линией со стрелкой (вариант отношения зависимости), направленной от базового варианта использования к включаемому. При этом данная линия со стрелкой помечается ключевым словом «include» («включает»)



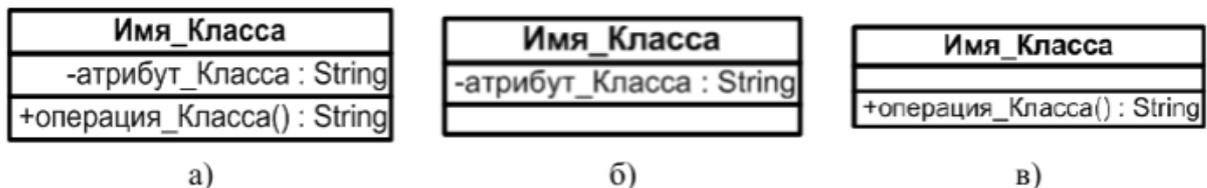
#### Построение диаграммы Классов

Диаграмма классов является одной из канонических диаграмм UML, создаваемой для визуализации структурированной статической модели предметной области. Этот вид диаграмм представляет собой графическое изображение объектов – классов с присущими им атрибутами, операциями и различных отношений между классами.

##### 1. Классы

Класс (class) служит для обозначения множества объектов, обладающих функциональным набором одинаково описывающих параметров (атрибутов), реализуемых операций и однотипными отношениями с объектами других классов.

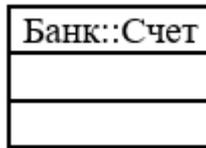
На диаграмме класс изображается габаритной прямоугольной рамкой, которая дополнительно может быть разделена горизонтальными линиями на секции, каждая из которых предназначена для указания имени, атрибутов (свойств) и реализуемых операций объектов данного класса.



Имя класса является обязательным элементом в его обозначении и должно быть уникальным (хотя бы в пределах пакета), имеющее непосредственное отношение к контексту моделируемой предметной области.

В соответствии с принятым в языке UML общим соглашением в качестве имени класса используются существительные и прилагательные, каждое из которых начинается с заглавной буквы, записанные без пробелов. Например, в качестве имен классов могут быть использованы профессиональные термины: «Сотрудник», «Компания», «Руководитель», «Клиент», «Продавец», «Менеджер», «Офис», «Покупатель», «Датчик\_Температуры» и др. Такие имена классов являются простыми.

Иногда возникает необходимость в явном указании пакета, к которому относится данный класс. С этой целью в условном обозначении перед именем класса указывается имя пакета и специальный символ разделитель – двойное двоеточие "::". Такое имя класса является квалифицированным. Текстовая строка имени класса в этом случае записывается в формате <Имя\_пакета>::<Имя\_класса>. Например, если определен пакет с именем «Банк», то имя класса «Счет» может быть записано так.

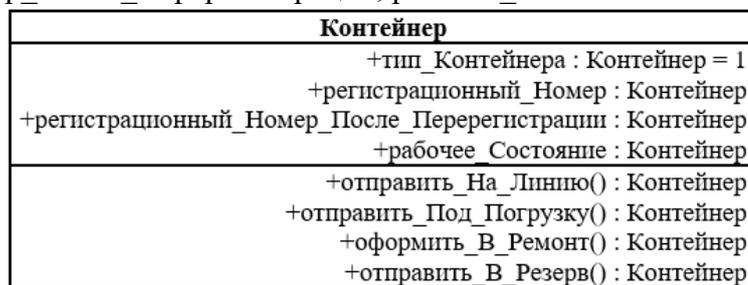


## 2. Атрибуты классов

Содержательной характеристикой класса является атрибут, содержащий множество значений, которые могут принимать отдельные объекты этого класса. При этом, класс может иметь любое число атрибутов или не иметь ни одного. Так, например, атрибутами класса «Усилитель» являются частотный диапазон, выходная мощность, коэффициент нелинейных искажений, уровень шума и т. д.

Запись атрибута также представляет собой отдельную строку текста, содержащую обязательное имя, в котором обычно каждое слово пишется с заглавной буквы, за исключением первого, например, name (имя) или birth\_Date (дата\_Рождения).

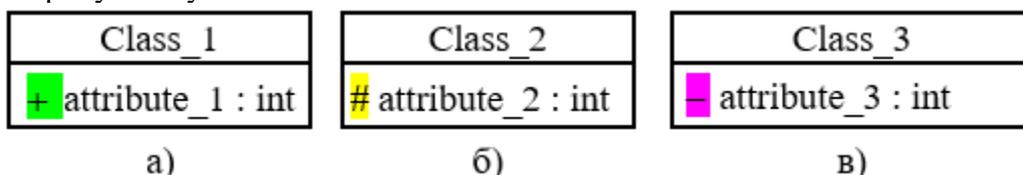
Например, указаны атрибуты класса Контейнер, в качестве которых выступают атрибут тип\_Контейнера, атрибут регистрационный\_Номер\_Контейнера, регистрационный\_Номер\_После\_Перерегистрации, рабочее\_Состояние.



Качественной характеристикой описания элементов класса является квантор видимости атрибута – потенциальная возможность других объектов модели оказывать влияние на отдельные аспекты поведения данного класса.

Эта характеристика может принимать одно из трех возможных значений и, соответственно, отображается при помощи специальных символов: символ "+" (public) обозначает атрибут с областью видимости типа общедоступный; атрибут с этой областью видимости доступен или виден из любого другого класса пакета, в котором определена диаграмма; например, для класса Class\_1 указан атрибут общедоступного типа, символ "#" (protected) обозначает атрибут с областью видимости типа защищенный; атрибут с этой областью видимости недоступен или невиден для всех классов, за исключением подклассов данного класса; например, для класса Class\_2 указан атрибут защищенного типа; символ "-" (private) обозначает атрибут с областью видимости типа закрытый; атрибут с этой областью видимости недоступен или невиден для всех классов без исключения; например, для класса Class\_3 указан атрибут защищенного типа;

Квантор видимости при описании атрибутов может быть опущен, что будет означать тот факт, что видимость атрибута не указывается.



## 3. Операции классов

Операция (operation) класса – это реализация услуги, которая может быть запрошена у любого объекта данного класса, чтобы вызвать определенное его поведение. Класс может иметь любое число операций либо не иметь ни одной. Так автомобиль может перемещаться по грунту, корабль – перемещаться по воде, компьютер – производить вычисления.

Представление полного синтаксиса записи операций класса также подчиняется определенным синтаксическим правилам: каждой операции класса соответствует отдельная строка, которая состоит из квантора видимости операции, обязательного имени операции, выражения типа возвращаемого операцией значения и, возможно, строки-свойства данной операции:

< квантор видимости >< имя операции > (список параметров) : < выражение типа возвращаемого значения >{строка-свойство}

Квантор видимости, как и в случае атрибутов класса, может принимать одно из трех возможных значений и, соответственно, также отображается при помощи специального символа.

Для именованной операции используются короткие глагольные конструкции, описывающие некоторое поведение класса, которому принадлежит операция. Обычно каждое слово в имени операции пишется с заглавной буквы, за исключением первого.

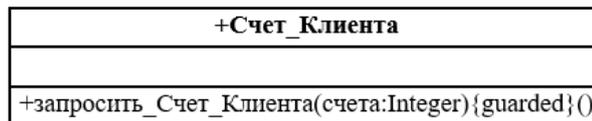
Например, запись +создать() – может обозначать абстрактную операцию по созданию отдельного объекта класса, которая является общедоступной и не содержит формальных параметров, запись +нарисовать(форма: Многоугольник = прямоугольник, цвет\_заливки: Color = (0, 0, 255)) – может обозначать операцию по изображению на экране монитора прямоугольной области синего цвета, если не указываются другие значения в качестве аргументов данной операции.

*(список-параметров) содержит необязательные аргументы, синтаксис которых совпадает с синтаксисом атрибутов;*

*< выражение типа возвращаемого значения > является необязательной спецификацией и зависит от конкретного языка программирования;*

*{строка-свойство} показывает значения свойств, которые применяются к данной операции.*

Например, запись запросить\_Счет\_Клиента(номер\_счета:Integer) – обозначает операцию по установлению наличия средств на текущем счете клиента банка. При этом аргументом данной операции является номер счета клиента, который записывается в виде целого числа (например, «123456»).



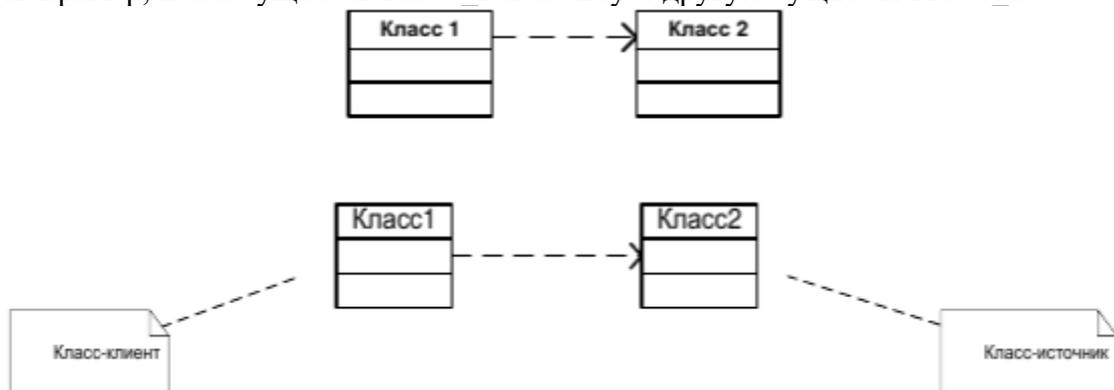
Квантор видимости для операции может быть опущен. В этом случае его отсутствие означает, что видимость операции не указывается.

#### **4. Отношения между классами**

Классы на диаграмме связываются различными типами отношений. При этом совокупность типов таких отношений фиксирована в языке UML и предопределена их семантикой.

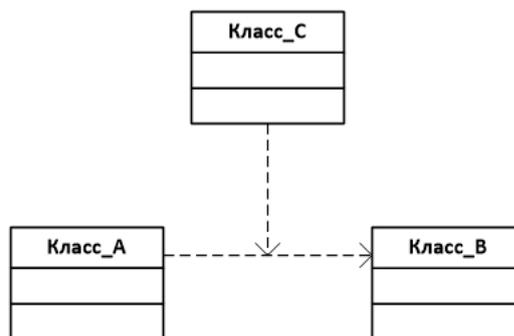
##### **4.1. Отношение зависимости**

**Отношением зависимости (dependency relationship)** называют связь по использованию, когда изменение в спецификации одного класса может повлиять на поведение другого. Отношение зависимости используется в такой ситуации, когда некоторое изменение одного элемента модели может потребовать изменения другого зависящего от него элемента модели. Отношение зависимости графически изображается пунктирной линией между соответствующими элементами со стрелкой на одном из ее концов, направленной к той сущности, от которой зависит данная сущность. Например, некая сущность Класс\_2 использует другую сущность Класс\_1.



В качестве класса-клиента и класса-источника зависимости может выступать множество элементов модели. В этом случае одна линия со стрелкой, выходящая от источника зависимости, расщепляется в некоторой точке на несколько отдельных линий, каждая из которых имеет отдельную стрелку для класса-клиента.

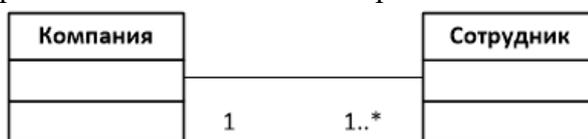
Например, если функционирование сущности Класс\_С зависит от особенностей реализации сущностей Класс\_А и Класс\_Б, то данная зависимость может быть изображена следующим образом.



#### 4.2. Отношение ассоциации

**Ассоциацией (association relationship)** называется структурная связь, показывающая, что объекты одного класса некоторым образом связаны с объектами другого или того же самого класса.

Ассоциация может отображаться графически линией со стрелкой (маркером в виде треугольника), показывающей направление следования классов и кратность – количество объектов, связанных отношением. Отсутствие стрелки рядом с именем ассоциации означает, что порядок следования классов в рассматриваемом отношении не определен



Так, в примере кратность «1» для класса «Компания» означает, что каждый сотрудник может работать только в одной компании. Кратность «1..\*» для класса «Сотрудник» означает, что в каждой компании могут работать несколько сотрудников, общее число которых заранее неизвестно и ничем не ограничено.

Специальной формой или частным случаем отношения ассоциации является отношение агрегации, которое, в свою очередь, тоже имеет специальную форму – отношение композиции (см. пункт 3.4.4).

#### 4.3. Отношение агрегации

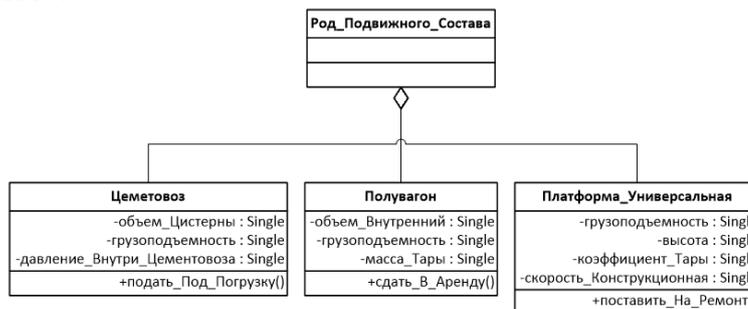
**Отношение агрегации (generalization relationship)** имеет место между несколькими классами в том случае, если один из классов представляет собой некоторую сущность, включающую в себя в качестве составных частей другие сущности.

Данное отношение имеет фундаментальное значение для описания структуры сложных систем, поскольку применяется для представления системных взаимосвязей типа «часть – целое».

Это отношение по своей сути описывает декомпозицию или разбиение сложной системы на более простые составные части, которые также могут быть подвергнуты декомпозиции, если в этом возникнет необходимость в последующем.

Так, автомобиль состоит из кузова, двигателя, трансмиссии и т.п., а в состав приемопередающего устройства входят передатчик, приемник и антенно-фидерное устройство.

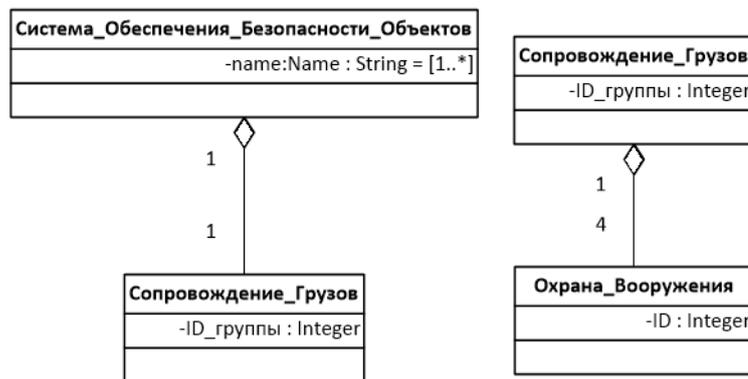
Графически отношение агрегации изображается сплошной линией, один из концов которой представляет собой геометрическую фигуру – ромб. Этот ромб указывает на тот из классов, который представляет собой «целое».



Примером отношения агрегации может служить деление класса Аналитическая\_информация на составные части: Отчет\_по\_грузу, Отчет\_по\_контейнерам, Отчет\_по\_тарифам.



Отношение агрегации обладает кратностью. Так, класс Система\_обеспечения\_безопасности\_объектов может содержать одну подсистему Сопровождение\_грузов, которая в свою очередь может содержать, например, четыре класса Охрана\_вооруженная, каждый из которых может принадлежать лишь одному классу Сопровождение\_грузов.



#### 4.4. Отношение композиции

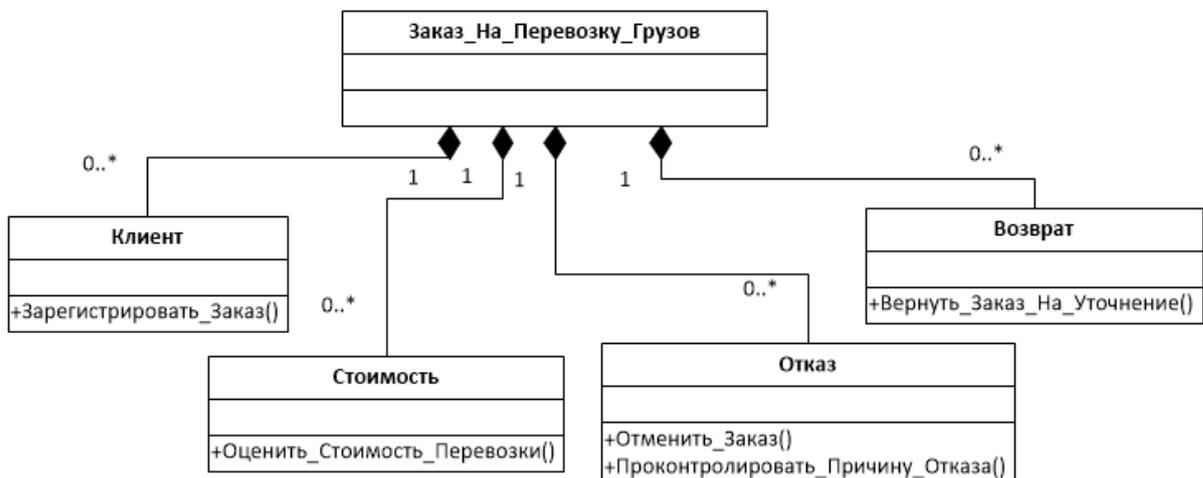
**Отношение композиции (realization relationship)** служит для выделения специальной формы отношения «часть-целое», при которой составляющие части в некотором смысле находятся внутри целого.

Специфика взаимосвязи между ними заключается в том, что части не могут выступать в отрыве от целого, т. е. с уничтожением целого уничтожаются и все его составные части.

Графически отношение композиции изображается сплошной линией, один из концов которой представляет собой закрашенный внутри ромб. Этот ромб указывает на тот из классов, который представляет собой класс-композицию или «целое».



Применительно к классу Заказ\_на\_перевозку\_грузов отношение композиции может иметь следующий вид.



#### 4.5. Отношение обобщения

**Отношение обобщения (генерализация)** является обычным таксономическим отношением между более общим элементом (класс-предок) и более частным или специальным элементом (класс-потомок).

Применительно к диаграмме классов данное отношение описывает иерархическое строение классов и наследование их свойств и поведения. При этом предполагается, что класс-потомок обладает всеми свойствами и поведением класса-предка, а также имеет свои собственные свойства и поведение, которые отсутствуют у класса-предка.

На диаграммах отношение обобщения обозначается сплошной линией с треугольной стрелкой на одном из концов, направленной на более общий класс (класс-предок или суперкласс) от более специального класса (класса-потомка или подкласса).

Как правило, на диаграмме может указываться несколько линий для одного отношения обобщения, что отражает его таксономический характер. В этом случае более общий класс разбивается на подклассы одним отношением обобщения, например, так, как показано на рис. 14.

В этом случае данные отдельные линии изображаются сходящимися к единственной стрелке, имеющей с ними общую точку пересечения. Родительский Класс Отчет\_По\_Заказам\_На\_Перевозку имеет три потомка Отчет\_По\_Количеству\_Заказов, Отчет\_По\_Клиентам, Отчет\_За\_Период, которые наследуют структуру и поведение родительского класса.



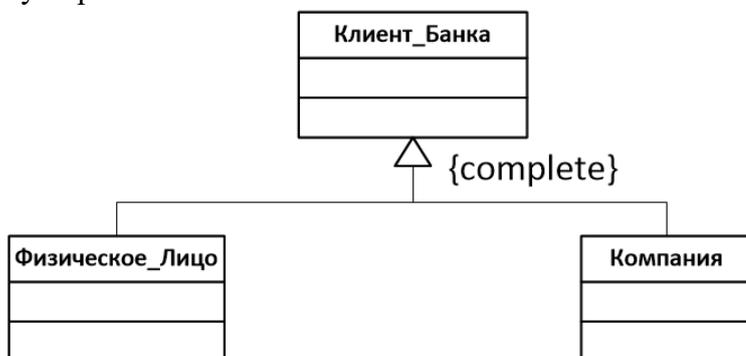
Для связей обобщения язык UML содержит ограничения. В большинстве случаев ограничение размещается рядом с элементом и заключается в фигурные скобки, например {complete}.

**В качестве ограничений** могут быть использованы следующие ключевые слова языка UML:

1. {complete} означает, что в данном отношении обобщения специфицированы все классы-потомки, и других классов-потомков у данного класса-предка быть не может.

Например, класс Клиент\_банка является предком для двух классов: Физическое\_лицо и Компания, и других классов-потомков он не имеет.

На соответствующей диаграмме классов это можно указать явно, записав рядом с линией обобщения данную строку-ограничение.



2. {incomplete} означает тот факт, что на диаграмме указаны в обобщении не все классы-потомки. В последующем, возможно, восполнить их перечень, не изменяя уже построенную диаграмму.

3. {disjoint} означает тот факт, что классы-потомки не могут содержать объектов, одновременно являющихся экземплярами двух или более классов.

В приведенном выше примере это условие также выполняется, поскольку предполагается, что никакое конкретное физическое лицо не может являться одновременно и конкретной компанией. В этом случае рядом с линией обобщения можно записать данную строку-ограничение.

4. {overlapping} означает, что отдельные экземпляры классов-потомков могут принадлежать одновременно нескольким классам.

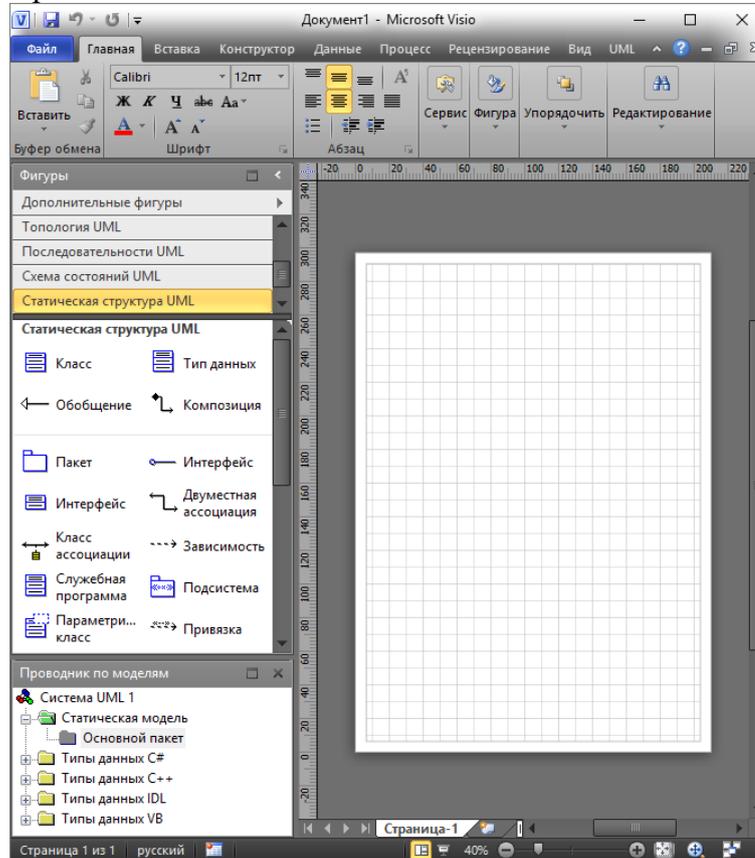
Например, класс Транспорт может быть специализирован путем создания подклассов Наземный\_Транспорт и Водный\_Транспорт, автомобиль – амфибия относится к обоим классам.

## Методика выполнения

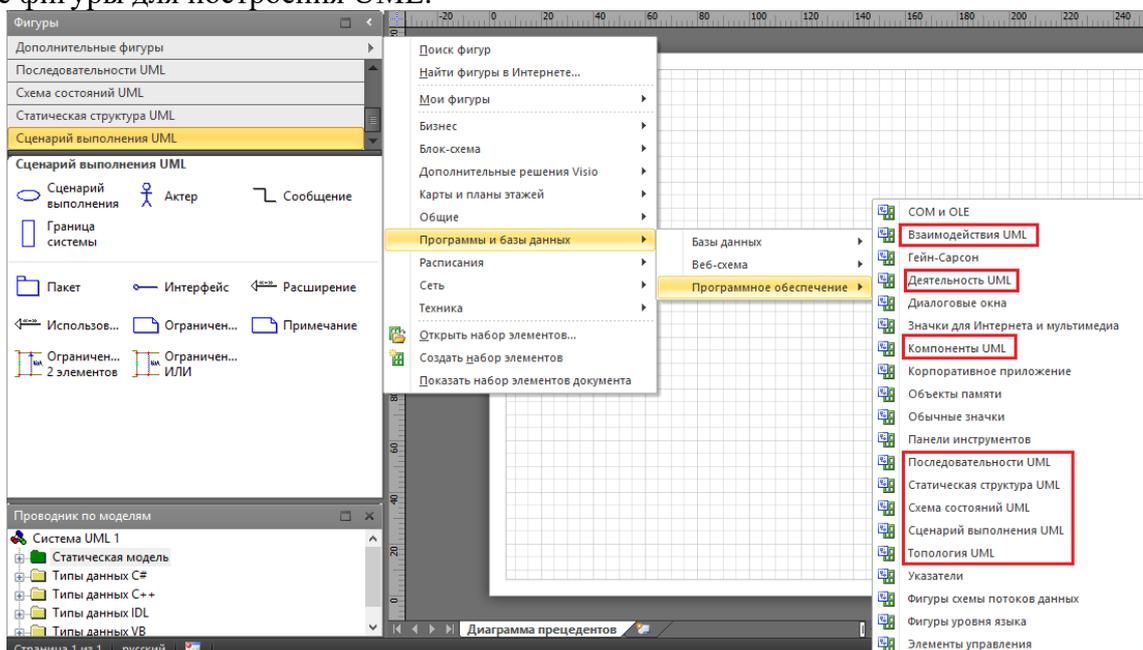
### Построение диаграммы Вариантов использования

В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

1. Запустите MS Visio.
2. На экране выбора шаблона выберите категорию *Программы и БД* и в ней элемент *Схема модели UML*. Нажмите кнопку *Создать* в правой части экрана.
3. Окно программы примет вид.



4. Далее необходимо открыть все фигуры, необходимые для построения UML-диаграмм. Для этого в левой части экрана необходимо нажать кнопку *Дополнительные фигуры*. В открывшемся вспомогательном меню выбрать *Программы и БД* -> *Программное обеспечение* и выбрать все доступные фигуры для построения UML.

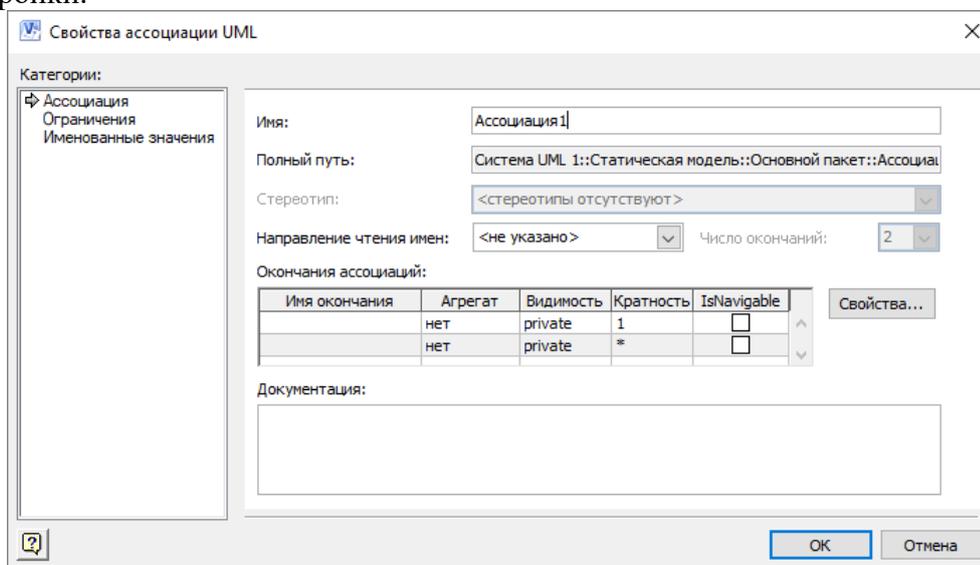


5. После этого необходимо провести следующие этапы моделирования.

#### 5.1. Выбор актеров.

В качестве актеров данной системы могут выступать два субъекта, один из которых является продавцом, а другой – покупателем. Каждый из этих актеров взаимодействует с рассматриваемой системой продажи товаров по каталогу и является ее пользователем, т. е. они оба обращаются к соответствующему сервису «Оформить заказ на покупку товара». Как следует из существа выдвигаемых к системе требований, этот сервис выступает в качестве варианта использования разрабатываемой диаграммы, первоначальная структура которой может включать в себя только двух указанных актеров и единственный вариант использования (рис. 14).

- В группе фигур *Сценарий выполнения UML* выбрать блок *Граница системы* и добавить его на лист.
- Внутри границы системы добавить блок *Сценарий выполнения* и добавить к нему название, дважды щелкнув внутри блока.
- Добавить два блока *Актер* – покупатель и продавец.
- С помощью блока *Сообщение* установите связь актеров и варианта использования. Двойным щелчком правой кнопки мыши по блоку *Сообщение* откройте окно *Свойств ассоциации UML*, проведите настройки.

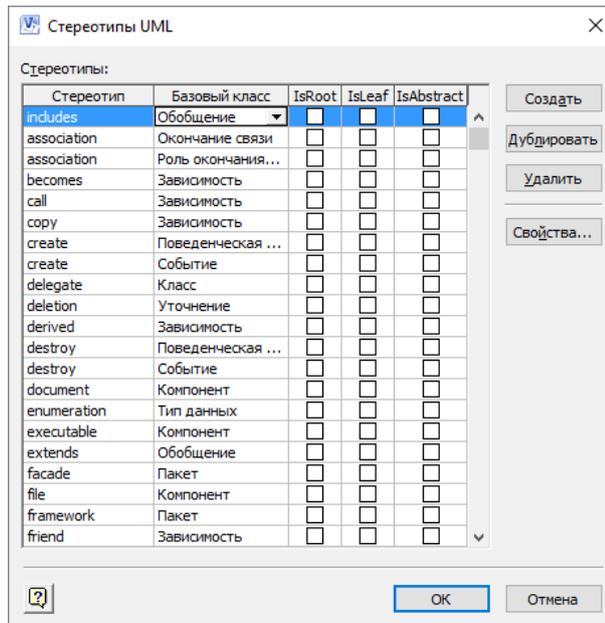


## 5.2. Выделение дополнительных вариантов использования.

Детализировать вариант использования «Оформить заказ на продажу товара» можно выделив следующие дополнительные варианты использования:

- обеспечить покупателя информацией – является отношением включения;
- согласовать условия оплаты – является отношением включения;
- заказать товар со склада – является отношением включения;
- запросить каталог товаров – является отношением расширения.

Так как в MS Visio отсутствует отношение включения, его необходимо добавить самостоятельно. Для этого перейти на вкладку *UML* -> в группе *Модель* выбрать пункт *Стереотипы*. В открывшемся окне нажать кнопку *Создать* и настроить стереотип



Далее на новом листе необходимо добавить границу системы и все варианты использования. После чего соединить варианты использования с помощью блока *Расширение*.

Для того, чтобы изменить тип отношения дважды щелкните по стрелке и окне свойств задайте необходимые параметры.

Дополненная диаграмма вариантов использования примет вид.



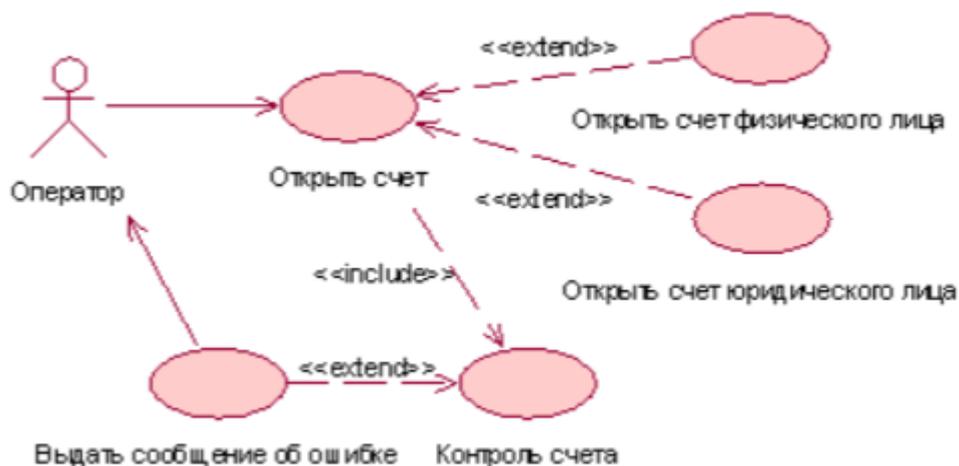
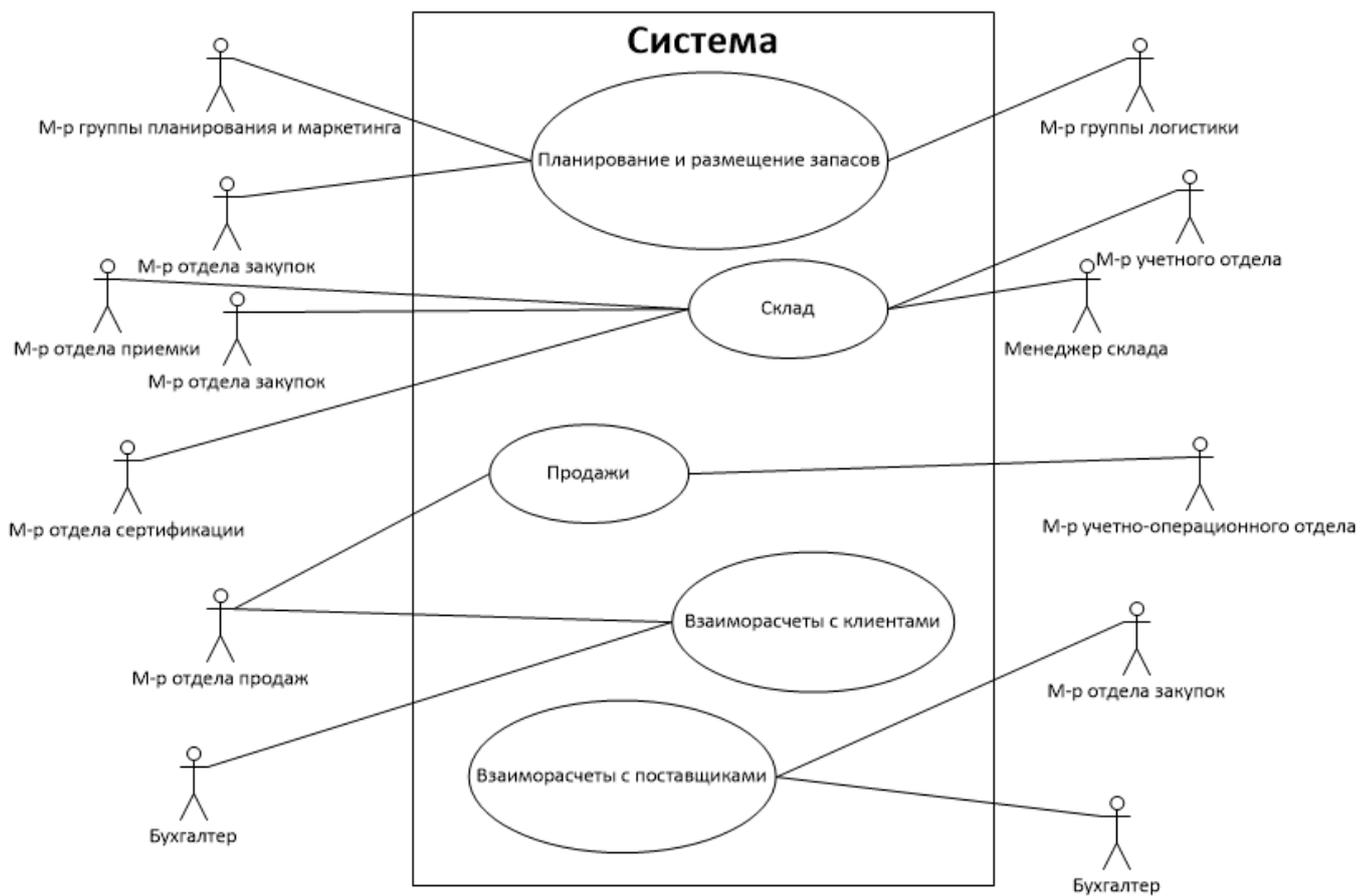
### 5.3. *Написание описательной спецификации для каждого варианта использования.*

Спецификация для варианта использования «Оформить заказ на покупку компьютера» приведена в таблице.

Раздел	Описание
Краткое описание	Покупатель желает оформить заказ на покупку компьютера, который он выбрал в каталоге товаров. При условии, что клиент зарегистрирован и выбранный компьютер есть в наличии оформляется заказ. Если клиент не зарегистрирован, то предлагается ему пройти регистрацию, и после этого заказать выбранный компьютер. Если компьютера нет в наличии, то предлагается заказать товар со склада в течении заданного срока поставки.
Субъекты	Продавец, Покупатель
Предусловия	В каталоге товаров имеются компьютеры, которые можно заказать. У покупателей есть доступ к системе для регистрации. Продавцы умеют пользоваться рассматриваемой системой продажи. У покупателя есть бонусы.
Основной поток	<p>Зарегистрированный покупатель имеет возможность заказать любой компьютер из каталога товаров. В случае наличия выбранного компьютера оформляется заказ с присвоением ему уникального номера. После этого покупателю предлагается выбрать способ оплаты и способ получения компьютера.</p> <p>В случае отсутствия компьютера в наличии предлагается оформить заказ со склада и ожидания его поставки в рамках указанного срока или выбрать другой компьютер.</p>
Альтернативный поток	<p>Покупатель не зарегистрирован. В этом случае, прежде чем оформить заказ на компьютер, ему предлагается пройти регистрацию.</p> <p>Попытка заказать товар, который отсутствует на складе</p> <p>Начисление бонусов</p>
Постусловия	Заказ оформлен и определен срок поставки компьютера и место его получения

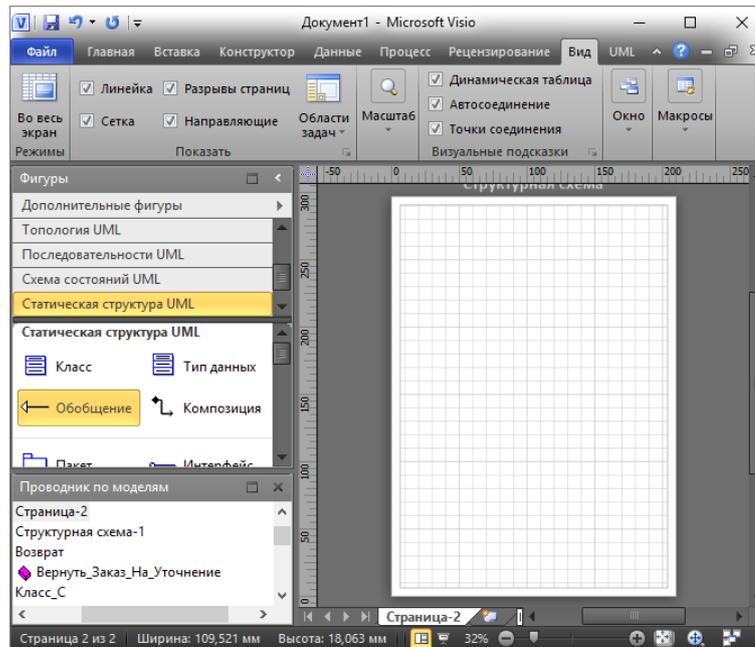
На рисунках приведены примеры диаграмм вариантов использования для различных систем.





### Построение диаграммы Классов

1. Запустите MS Visio.
2. На экране выбора шаблона выберите категорию *Программы и БД* и в ней элемент *Схема модели UML*. Нажмите кнопку *Создать* в правой части экрана.
3. Ознакомьтесь с элементами графического интерфейса и найдите обязательные панели инструментов **Фигуры**, содержащие категории **Деятельность UML**, **Взаимодействия UML**, **Компоненты UML**, **Топология UML**, **Последовательности UML**, **Схема состояний UML**, **Статическая структура UML**, **Сценарий выполнения UML**, **Проводник по моделям**, содержащий иерархическую структуру объектов Системы UML1, **Рабочую область**, ярлык **Страница\_1**, горизонтальную и вертикальную линейки



4. Установите следующие параметры страницы: **Ориентация** – Альбомная, **Автоподбор размера** – выключен, **Имя страницы** – *Диаграмма классов для системы продажи товаров по каталогу*.

5. Перейдите в категорию **Статическая структура UML**, ознакомьтесь с содержимым этой категории и найдите элементы: Класс, Пакет, Подсистема, Интерфейс, Метакласс, Двусторонняя ассоциация, Обобщение, Композиция, Примечание, Ограничение и др.

6. Создайте поэтапно статическую структуру классов UML, с помощью которой может быть сформирована некоторая функциональная часть системы, например, *Система продажи товаров по каталогу*. Для чего:

– Выберите структурные элементы (идентифицируйте классы), участвующие в организации продаж, например, *Продавец, Товар, Заказ, Заказ\_Оплата, Клиент, Корпоративный\_Клиент, Частный\_Клиент* и создайте предварительный вариант совокупности классов с указанием имен

– Установите для каждого класса атрибуты в соответствии с перечнем и содержательным описанием бизнес-процессов:

например, для класса *Продавец* в качестве атрибутов могут выступать данные: *фамилия, имя, отчество, телефон*. В данном случае все атрибуты видимы, принадлежат основному пакету *Продавец*.

для класса *Товар* в качестве атрибутов могут выступать данные: тип, марка, артикул.



Товар
-Тип : String
-Марка : String
-Атрибут : String

для класса *Заказ* в качестве атрибутов могут выступать данные: количество, цена, статус, а в качестве операций – сформировать заказ.

Заказ
-Количество : Integer
-Цена : Double
-Статус : String
+Сформировать_Заказ()

для класса *Заказ\_Оплата* в качестве атрибутов могут выступать данные: дата получения, проплачен, номер, цена, а в качестве операций – отправить, закрыть.

Заказ_Оплата
-Дата_Получения : Date
-Проплачен : Boolean
-Номер : String
-Цена : Double
+Отправить()
+Закрыть()

для класса *Клиент* в качестве атрибутов могут выступать данные: имя, адрес, а в качестве операций – кредитный рейтинг.

Клиент
-Имя : String
-Адрес : String
+Кредитный_Рейтинг() : String

для класса *Корпоративный\_Клиент* в качестве атрибутов могут выступать данные: контактное имя, кредитный рейтинг, кредитный лимит, а в качестве операций – сделать, напоминание, счет за месяц.

Корпоративный_Клиент
-Контактное_Имя : String
-Кредитный_Рейтинг : Integer
-Кредитный_Лимит : Double
+Сделать()
+Напоминание()
+Счет_За_Месяц() : Integer

для класса *Частный\_Клиент* в качестве атрибутов могут выступать данные: номер кредитной карты.

Частный_Клиент
-Номер_Кредитной_Карты : String

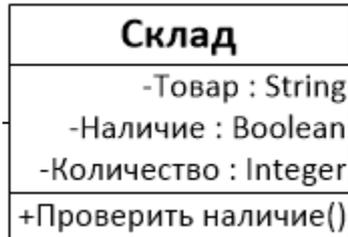
для класса *Вариант\_Оплаты* в качестве атрибутов могут выступать данные: тип оплаты, а в качестве операций – выбор варианта оплаты.

Вариант_Оплаты
-Тип_Оплаты : String
+Выбор_Варианта_Оплаты()

для класса *Каталог\_Товаров* в качестве атрибутов могут выступать данные: тип, марка, артикул, а в качестве операций – проверить наличие.

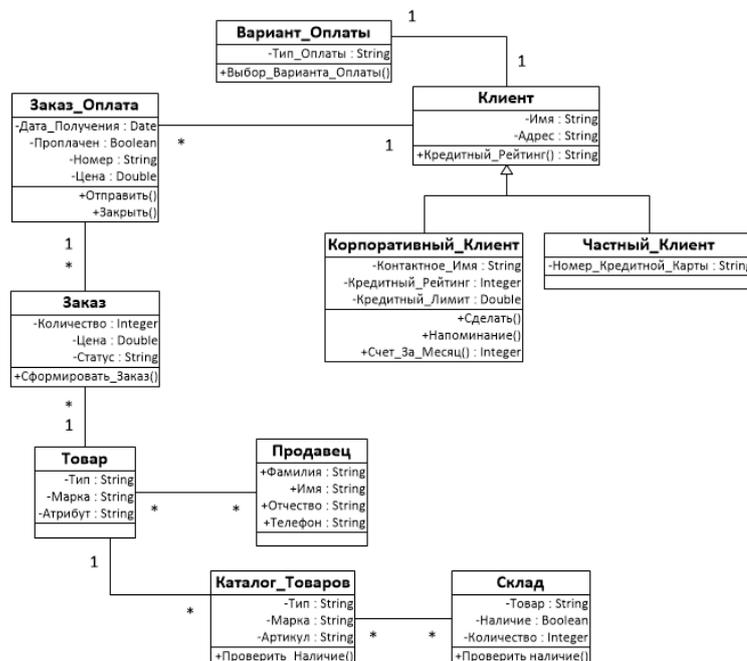


для класса *Склад* в качестве атрибутов могут выступать данные: товар, наличие, количество, а в качестве операций – Проверить наличие.



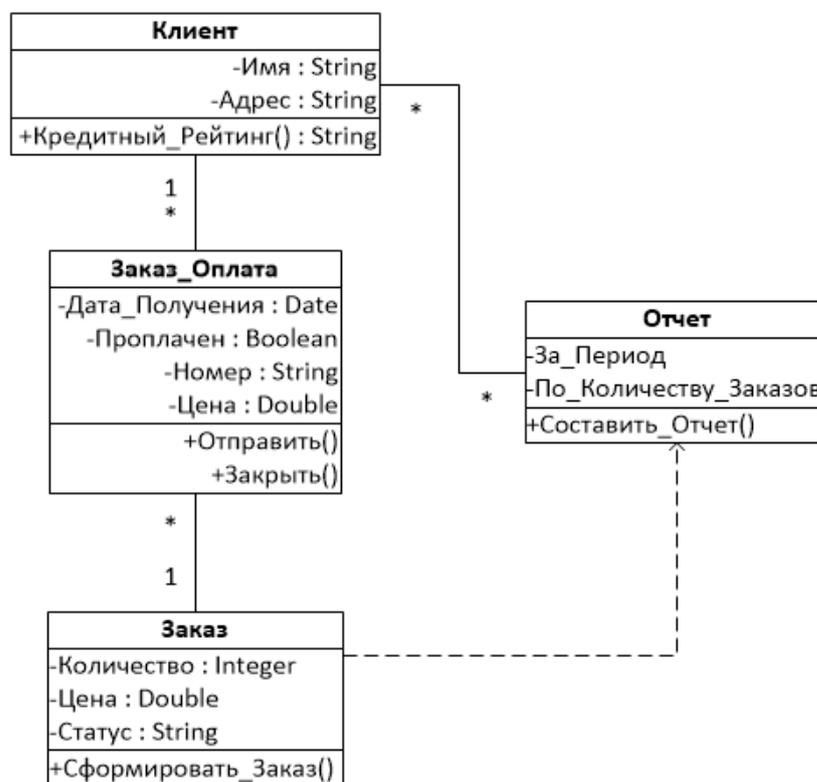
– Убедитесь, что все элементы наполнены адекватным содержанием и расположите все структурные элементы диаграммы наиболее оптимально на странице для установления отношений между ними.

В качестве примера на рис. 26 показан набор классов, описывающих реализацию системы продаж товаров по каталогу. Акцент сделан на классе *Клиент*, с которым связан класс *Заказ\_Оплата* посредством двусторонней ассоциации «один-ко-многим», *Вариант\_Оплаты* – двусторонней ассоциацией «один-к-одному» и классы *Корпоративный\_Клиент* и *Частный\_Клиент* посредством отношения обобщения. Классы *Заказ\_Оплата* и *Товар* связаны с классом *Заказ* посредством двусторонней ассоциации «один-ко-многим». Класс *Товар* связан с классом *Продавец* двусторонней ассоциацией «многие-ко-многим» и классом *Каталог\_Товаров* двусторонней ассоциацией «один-ко-многим». Класс *Каталог\_Товаров* связан посредством двусторонней ассоциации «многие-ко-многим» с классом *Склад*.



7. Создайте новую страницу с именем *Диаграмма классов учета клиентов*, и установите следующие опции: **Ориентация** – Альбомная, **Автоподбор размера** – выключен.

8. Идентифицируйте классы учета клиентов, осуществляющих заказы и создайте диаграмму классов с указанием их имен, атрибутов, операций, например.



### Задание практической работы

По образцу построить диаграммы вариантов использования и классов.

### Задание самостоятельной работы

В соответствии с индивидуальным вариантом, построить диаграммы вариантов использования и классов.

Перечень индивидуальных вариантов приведен в приложении А.

Отчет по практическому занятию выполняется в формате MS Word, который содержит экранные формы моделей согласно заданию.

### Контрольные вопросы

1. Для чего используется язык UML?
2. Назначение диаграммы вариантов использования?
3. Что такое «актер»?
4. Что такое «вариант использования»?
5. Что такое «интерфейс»?
6. Что такое «примечание»?
7. Перечислить виды отношений между актерами и вариантами использования, охарактеризовать каждое из них?
8. Каково назначение диаграммы классов?
9. Назовите основные элементы диаграммы классов.
10. Какие виды связей доступны в диаграмме классов?
11. Для чего используется каждый вид связи?
12. Как создать диаграмму классов в VISIO?

**Форма отчета:** тетрадь, защита работы.

## Практическое занятие № 13-14

**Тема:** Построение диаграммы Состояний

**Цель:** изучение основ создания диаграмм состояний на языке UML, получение навыков построения диаграмм состояний, применение приобретенных навыков для построения объектно-ориентированных моделей определенной предметной области.

**Оборудование:** тетрадь, ручка, ПК

**Методические указания:** ознакомьтесь с теоретической частью, выполните задания, ответьте на контрольные вопросы

**Ход выполнения:**

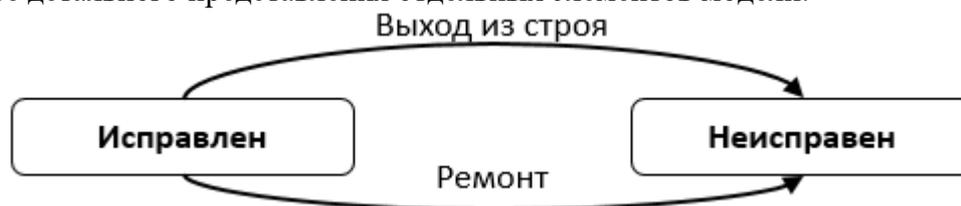
**Краткие теоретические сведения**

**Диаграмма состояний** описывает процесс изменения состояний только одного класса, а точнее – одного экземпляра определенного класса, т. е. моделирует все возможные изменения в состоянии конкретного объекта. При этом изменение состояния объекта может быть вызвано внешними воздействиями со стороны других объектов или извне. Именно для описания реакции объекта на подобные внешние воздействия и используются диаграммы состояний.

Диаграмма состояний описывает возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла. Диаграмма состояний представляет динамическое поведение сущностей, на основе спецификации их реакции на восприятие некоторых конкретных событий.

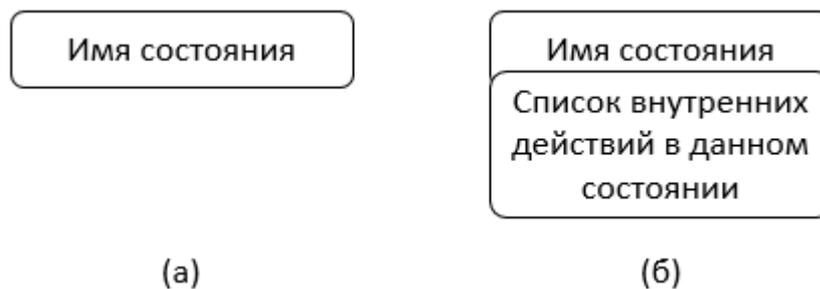
Хотя диаграммы состояний чаще всего используются для описания поведения отдельных экземпляров классов (объектов), но они также могут быть применены для спецификации функциональности других компонентов моделей, таких как варианты использования, актеры, подсистемы, операции и методы.

Диаграмма состояний по существу является графом специального вида, который представляет некоторый автомат. Понятие автомата в контексте UML обладает довольно специфической семантикой, основанной на теории автоматов. Вершинами этого графа являются состояния и некоторые другие типы элементов автомата (псевдосостояния), которые изображаются соответствующими графическими символами. Дуги графа служат для обозначения переходов из состояния в состояние. Диаграммы состояний могут быть вложены друг в друга, образуя вложенные диаграммы более детального представления отдельных элементов модели.



### Состояние

Под состоянием понимается абстрактный метакласс, используемый для моделирования отдельной ситуации, в течение которой имеет место выполнение некоторого условия. Состояние может быть задано в виде набора конкретных значений атрибутов класса или объекта, при этом изменение их отдельных значений будет отражать изменение состояния моделируемого класса или объекта.



Секция «Список внутренних действий» содержит перечень внутренних действий или деятельностей, которые выполняются в процессе нахождения моделируемого элемента в данном состоянии. Каждое из действий записывается в виде отдельной строки и имеет следующий формат:

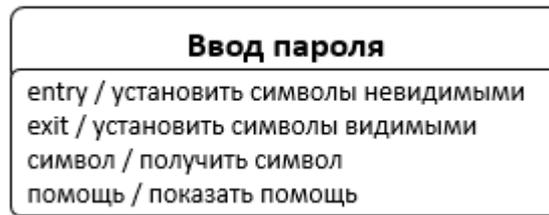
<метка-действия '/' выражение-действия>

**Метка действия** указывает на обстоятельства или условия, при которых будет выполняться деятельность, определенная выражением действия:

- **entry** – эта метка указывает на действие, специфицированное следующим за ней выражением действия, которое выполняется в момент входа в данное состояние (входное действие);
- **exit** – эта метка указывает на действие, специфицированное следующим за ней выражением действия, которое выполняется в момент выхода из данного состояния (выходное действие);
- **do** – эта метка специфицирует выполняющуюся деятельность («doactivity»), которая выполняется в течение всего времени, пока объект находится в данном состоянии, или до тех пор, пока не закончится вычисление, специфицированное следующим за ней выражением действия. В последнем случае при завершении события генерируется соответствующий результат;

– *include* – эта метка используется для обращения к подавтомату, при этом следующее за ней выражение действия содержит имя этого подавтомата.

**Пример: Аутентификация входа**



### **Начальное и конечное состояния**

**Начальное состояние** представляет собой частный случай состояния, которое не содержит никаких внутренних действий (псевдосостояния). В этом состоянии находится объект по умолчанию в начальный момент времени. Оно служит для указания на диаграмме состояний графической области, от которой начинается процесс изменения состояний.

**Конечное (финальное) состояние** представляет собой частный случай состояния, которое также не содержит никаких внутренних действий (псевдосостояния). В этом состоянии будет находиться объект по умолчанию после завершения работы автомата в конечный момент времени.



### **Переход**

**Простой переход (simpletransition)** представляет собой отношение между двумя последовательными состояниями, которое указывает на факт смены одного состояния другим. Пребывание моделируемого объекта в первом состоянии может сопровождаться выполнением некоторых действий, а переход во второе состояние будет возможен после завершения этих действий, а также после удовлетворения некоторых дополнительных условий. В этом случае говорят, что переход срабатывает, Или происходит срабатывание перехода. До срабатывания перехода объект находится в предыдущем от него состоянии, называемым исходным состоянием, или в источнике (не путать с начальным состоянием – это разные понятия), а после его срабатывания объект находится в последующем от него состоянии (целевом состоянии).

Переход осуществляется при наступлении некоторого события: окончания выполнения деятельности (doactivity), получении объектом сообщения или приемом сигнала. На переходе указывается имя события. Кроме того, на переходе могут указываться действия, производимые объектом в ответ на внешние события при переходе из одного состояния в другое. Срабатывание перехода может зависеть не только от наступления некоторого события, но и от выполнения определенного условия, называемого сторожевым условием. Объект перейдет из одного состояния в другое в том случае, если произошло указанное событие и сторожевое условие приняло значение «истина».

На диаграмме состояний **переход изображается** сплошной линией со стрелкой, которая направлена в целевое состояние. Каждый переход может помечен строкой текста, которая имеет следующий общий формат:

<сигнатура события>['<сторожевое условие>'] <выражение действия>

### **Событие**

**Событие** представляет собой спецификацию некоторого факта, имеющего место в пространстве и во времени. Про события говорят, что они «происходят», при этом отдельные события должны быть упорядочены во времени. После наступления некоторого события нельзя уже вернуться к предыдущим событиям, если такая возможность не предусмотрена явно в модели.

События играют роль стимулов, которые инициируют переходы из одних состояний в другие. В качестве событий можно рассматривать сигналы, вызовы, окончание фиксированных промежутков времени или моменты окончания выполнения определенных действий. Имя события идентифицирует каждый отдельный переход на диаграмме состояний и может содержать строку текста, начинающуюся со строчной буквы.

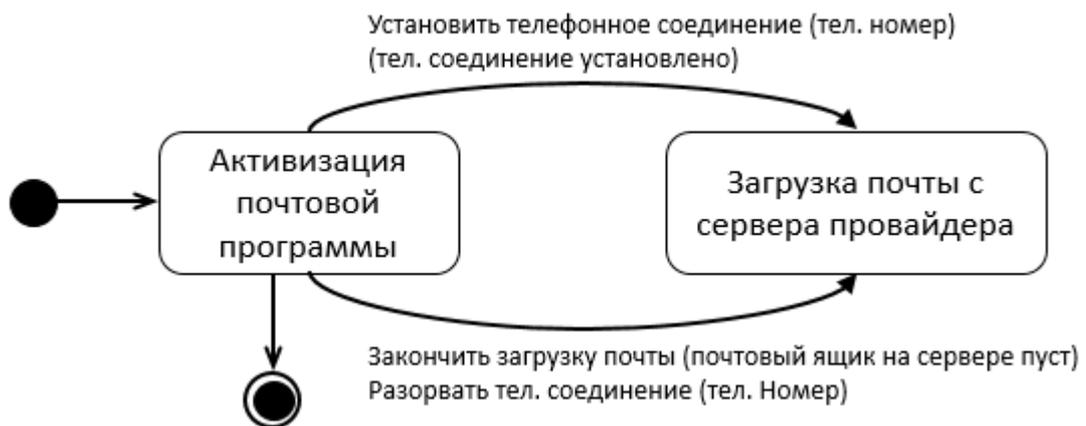
### Сторожевое условие

**Сторожевое условие (guardcondition)**, если оно есть, всегда записывается в прямых скобках после события и представляет собой некоторое булевское выражение.

Если сторожевое условие принимает значение «истина», то соответствующий переход может сработать, в результате чего объект перейдет в целевое состояние. Если же сторожевое условие принимает значение «ложь», то переход не может сработать, и при отсутствии других переходов объект не может перейти в целевое состояние по этому переходу. Однако вычисление истинности сторожевого условия происходит только после возникновения ассоциированного с ним события, инициирующего соответствующий переход.

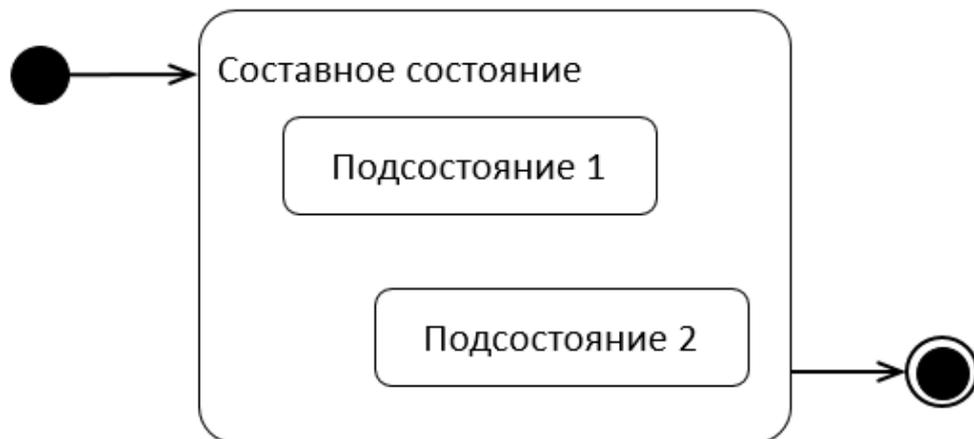
В общем случае из одного состояния может быть несколько переходов с одним и тем же событием-триггером. При этом никакие два сторожевых условия не должны одновременно принимать значение «истина». Каждое из сторожевых условий необходимо вычислять всякий раз при наступлении соответствующего события.

### Пример

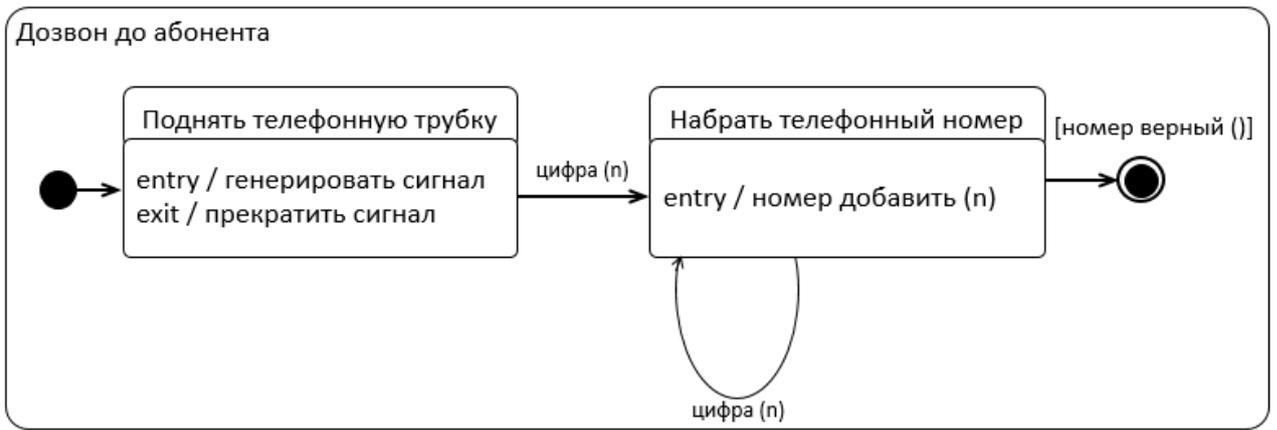


### Составное состояние и подсостояние

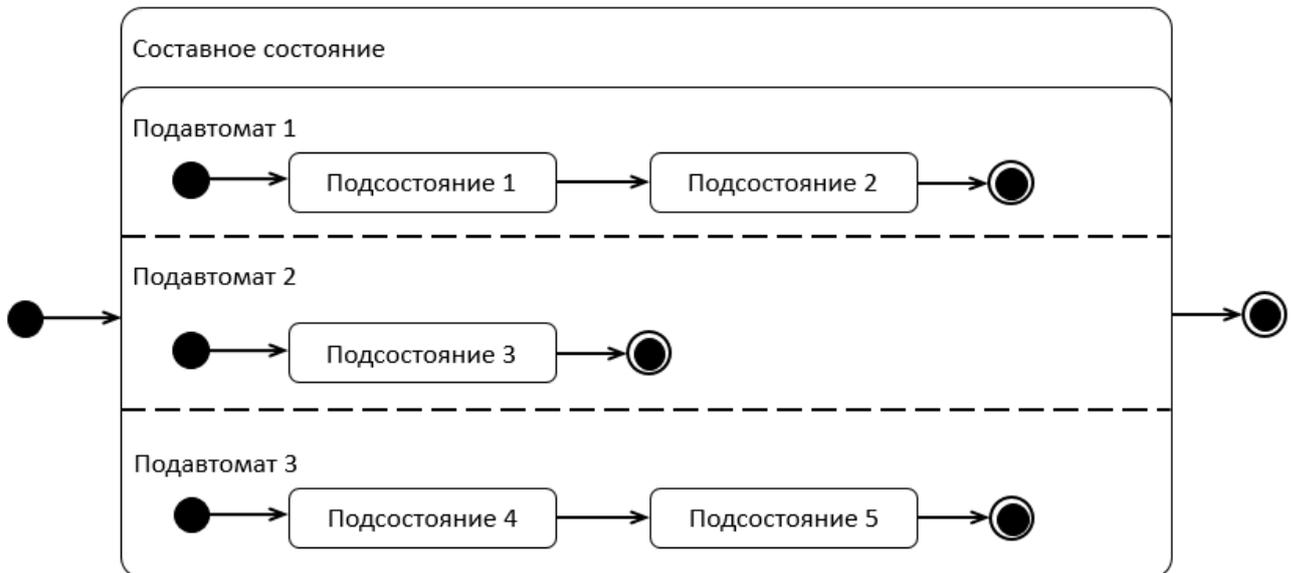
**Составное состояние (compositestate)** – такое сложное состояние, которое состоит из других вложенных в него состояний. Последние будут выступать по отношению к первому как подсостояния (substate).



**Последовательные подсостояния (sequential substates)** используются для моделирования такого поведения объекта, во время которого в каждый момент времени объект может находиться в одном и только одном подсостоянии. Поведение объекта в этом случае представляет собой последовательную смену подсостояний, начиная от начального и заканчивая конечным подсостояниями. Хотя объект продолжает находиться в составном состоянии, введение в рассмотрение последовательных подсостояний позволяет учесть более тонкие логические аспекты его внутреннего поведения.



**Параллельные подсостояния (concurrentsubstates)** позволяют специфицировать два и более подавтомата, которые могут выполняться параллельно внутри составного события. Каждый из подавтоматов занимает некоторую область (регион) внутри составного состояния, которая отделяется от остальных горизонтальной пунктирной линией. Если на диаграмме состояний имеется составное состояние с вложенными параллельными подсостояниями, то объект может одновременно находиться в каждом из этих подсостояний.



### Синхронизирующие состояния

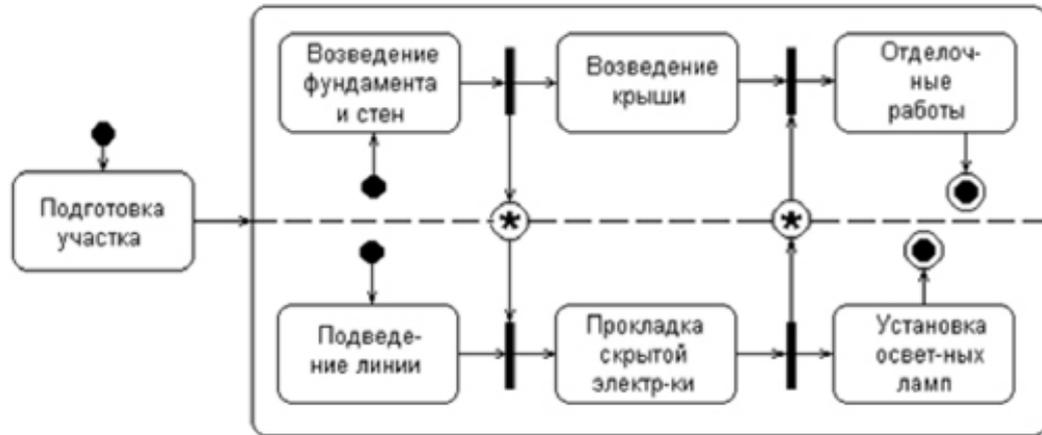
Поведение параллельных подавтоматов независимо друг от друга, что позволяет реализовать многозадачность в программных системах. Однако в отдельных случаях может возникнуть необходимость учесть в модели синхронизацию наступления отдельных событий. Для этой цели в языке UML имеется специальное псевдосостояние, которое называется синхронизирующим состоянием.

**Синхронизирующее состояние (synch state)** обозначается небольшой окружностью, внутри которой помещен символ звездочки "\*". Оно используется совместно с переходом-соединением или переходом-ветвлением для того, чтобы явно указать события в других подавтоматах, оказывающие непосредственное влияние на поведение данного подавтомата.

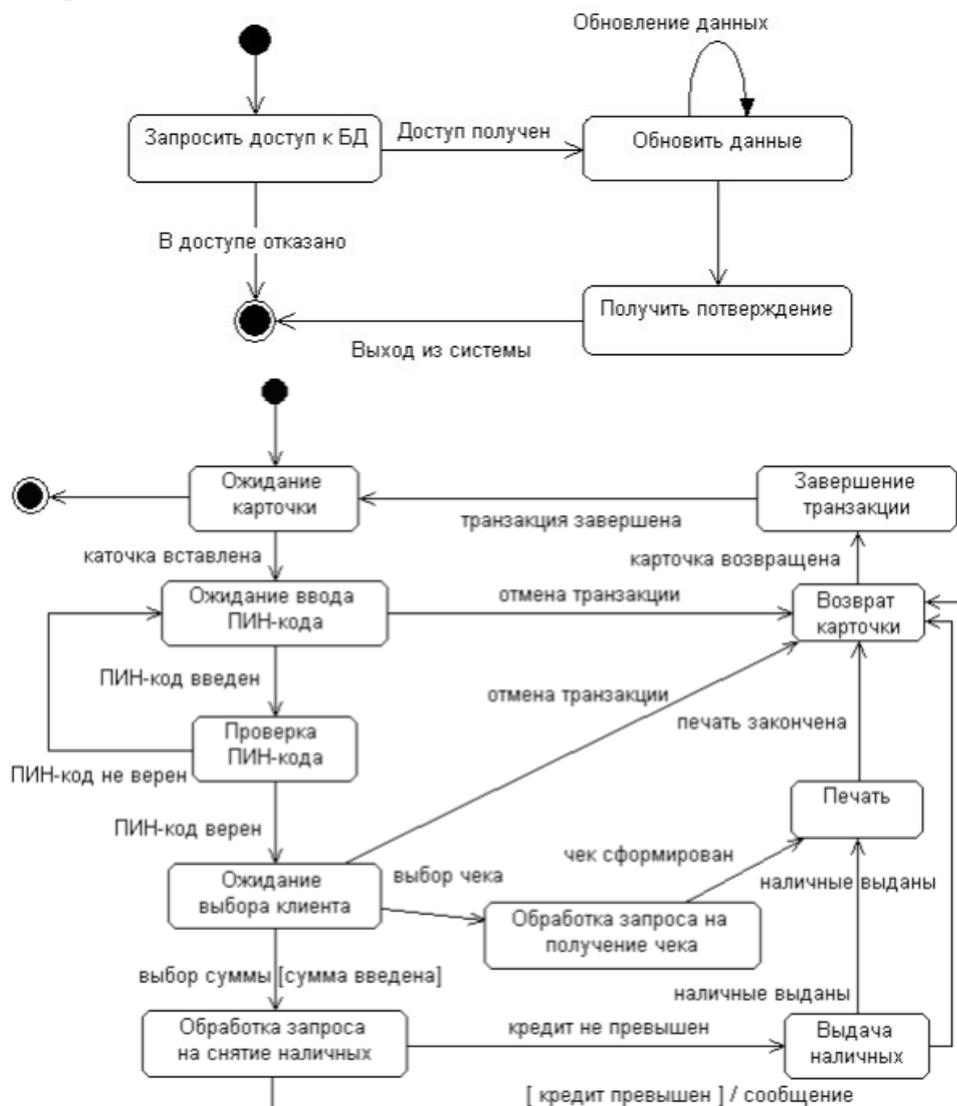
Для иллюстрации использования синхронизирующих состояний рассмотрим упрощенную ситуацию с моделированием процесса постройки дома. Предположим, что постройка дома включает в себя строительные работы (возведение фундамента и стен, возведение крыши и отделочные работы) и работы по электрификации дома (подведение электрической линии, прокладка скрытой электропроводки и установка осветительных ламп). Очевидно, два этих комплекса работ могут выполняться параллельно, однако между ними есть некоторая взаимосвязь.

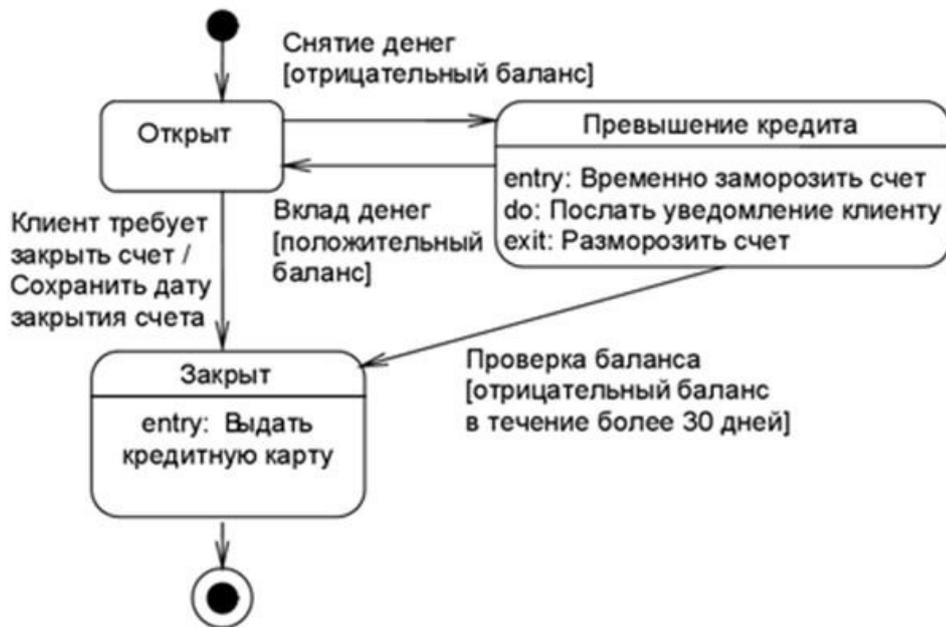
В частности, прокладка скрытой электропроводки может начаться лишь после того, как будет завершено возведение фундамента и стен. А отделочные работы следует начать лишь после того, как будет закончена прокладка скрытой электропроводки. В противном случае отделочные работы придется проводить повторно. Рассмотренные особенности синхронизации этих параллельных

процессов учтены на соответствующей диаграмме состояний с помощью двух синхронизирующих состояний.



Примеры диаграмм состояний.

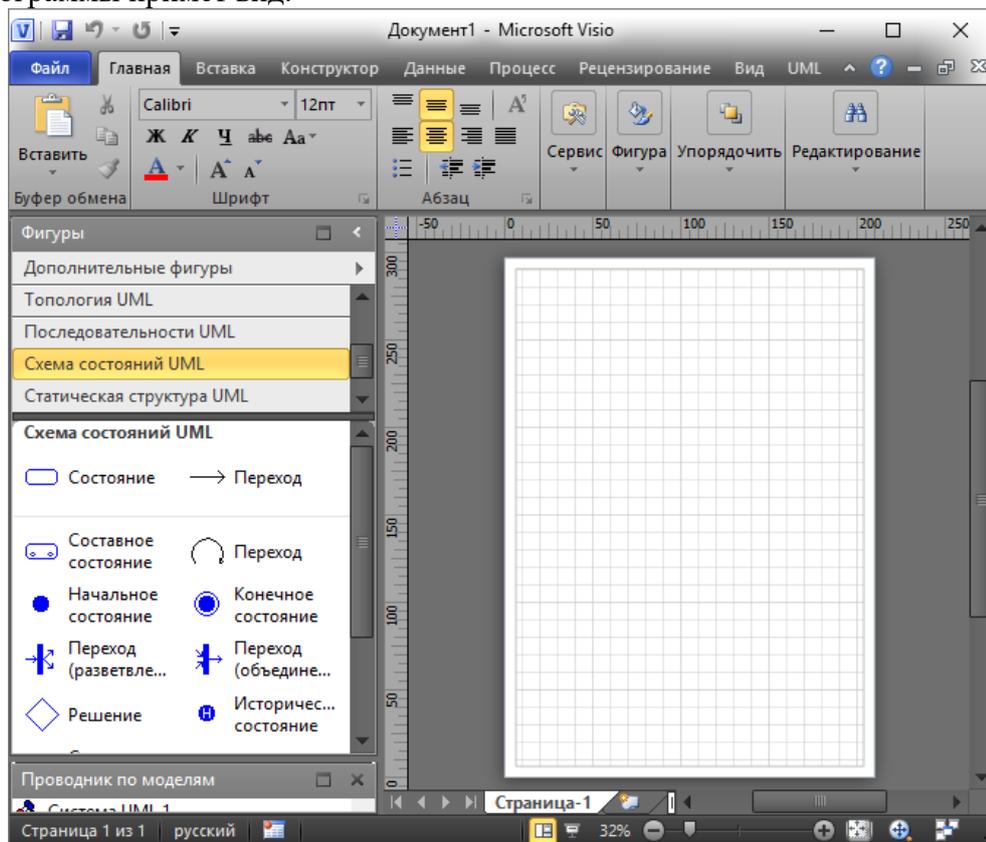




### Методика выполнения

В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

1. Запустите MS Visio.
2. На экране выбора шаблона выберите категорию *Программы и БД* и в ней элемент *Схема модели UML*. Нажмите кнопку *Создать* в правой части экрана.
3. Окно программы примет вид.



Клиент оформляет заказ. Класс Заказ имеет атрибут Статус. Проследим динамику движения заказов в системе с помощью **диаграммы состояний, составленной для класса Заказ**.

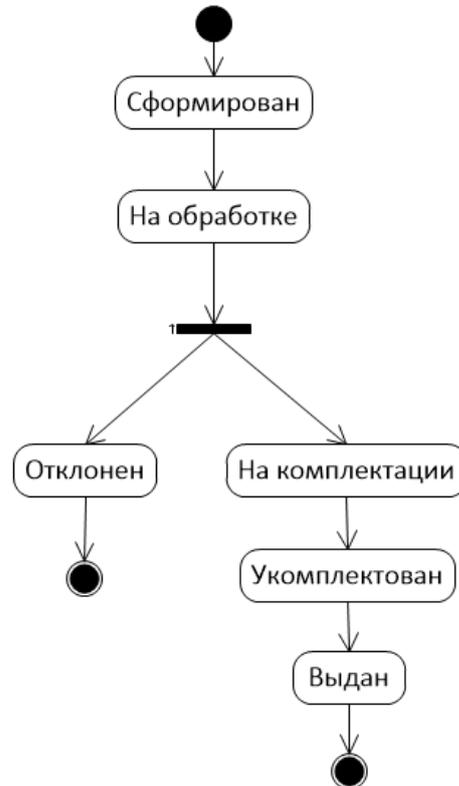
Данные о сформированном заказе поступают продавцу, который проверяет наличие товаров из заказа, проверяет оплату заказа, комплектует его и делает отметку о готовности. После оплаты заказа он выдается клиенту. Продавец делает отметку о том, что заказ выдан.

Если после проверки кредитного рейтинга клиента, он окажется отрицательным, то заказ будет отклонен.

Построим диаграмму состояний для класса Заказ. Для этого, в файле с диаграммой классов, созданной в практическом занятии 8, необходимо проделать следующие действия:

1. Щелкнуть правой кнопкой мыши по *классу Заказ*.
2. В контекстном меню выбрать пункт *Схемы*.
3. Т.к. в настоящее время уже созданных схем нет, нажать кнопку *Создать* и выбрать *Схема состояний*.
4. Переименовать созданный лист в *Схема состояний-Заказ*.
5. Построить диаграмму состояний для класса *Заказ*.

После формирования заказа он должен быть оплачен. Обработка заказа подразумевает проверку наличия товара и проверку оплаты. Переход в одно из состояний. На комплектации, Укомплектован, Выдан означает смену Статуса заказа.



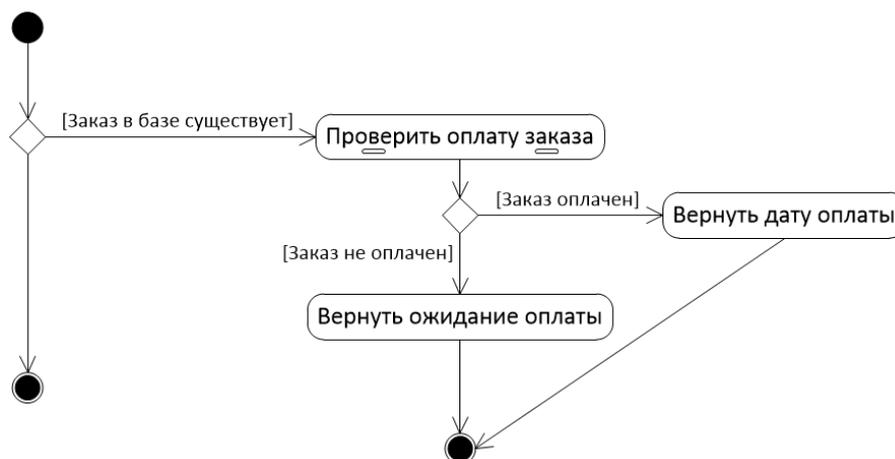
Далее опишем с помощью *диаграммы состояний процесс оплаты заказа клиентом*, которому соответствует класс *ЗаказОплата*.

Построим диаграмму состояний для проверки оплаты заказа.

Чтобы проверить оплату заказа, необходимо определить, существует ли сам заказ. Результатом проверки оплаты заказа является вывод либо сообщения о произведенной оплате с параметрами (дата оплаты), либо сообщения об ожидании оплаты.

Событием, предшествующим проверке оплаты заказа, является занесение информации о заказе в базу данных заказов.

Чтобы построить диаграмму состояний для класса *ЗаказОплата*, необходимо проделать действия, описанные в пунктах 1-4 построения диаграммы состояний для класса *Заказ*. Полученная диаграмма должна иметь вид, изображенный на рис. 15.



На этой диаграмме есть составное состояние **«Проверить оплату заказа»**, т.к. оно включает в себя *проверку кредитного рейтинга клиента и проверку выбора варианта оплаты клиентом*.

Оплату заказа может произвести только клиент с положительным кредитным рейтингом, поэтому необходимым условием проверки оплаты заказа является проверка кредитоспособности клиента. Если клиент имеет отрицательный кредитный рейтинг, то заказ отклоняется, и на этом дальнейшие события не имеют смысла.

Если кредитный рейтинг клиента положительный, то необходимо проверить, выбрал ли клиент вариант оплаты. Событие, которое переводит систему в состояние ожидания выбора варианта оплаты клиентом, является получение сообщения о кредитоспособности клиента.

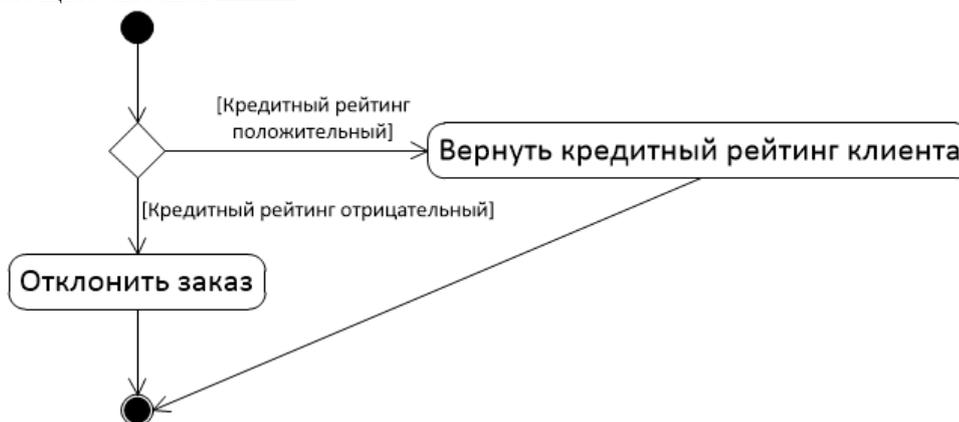
Оплата может быть произведена наличными средствами в магазине или с помощью безналичного расчета. В первом случае необходимо договориться с клиентом о дате и времени его прибытия в магазин. Во втором случае необходимо сообщить клиенту о наличии/поступлении товара. Событие, которое переводит систему в состояние ожидания оплаты, является выбор клиентом варианта оплаты.

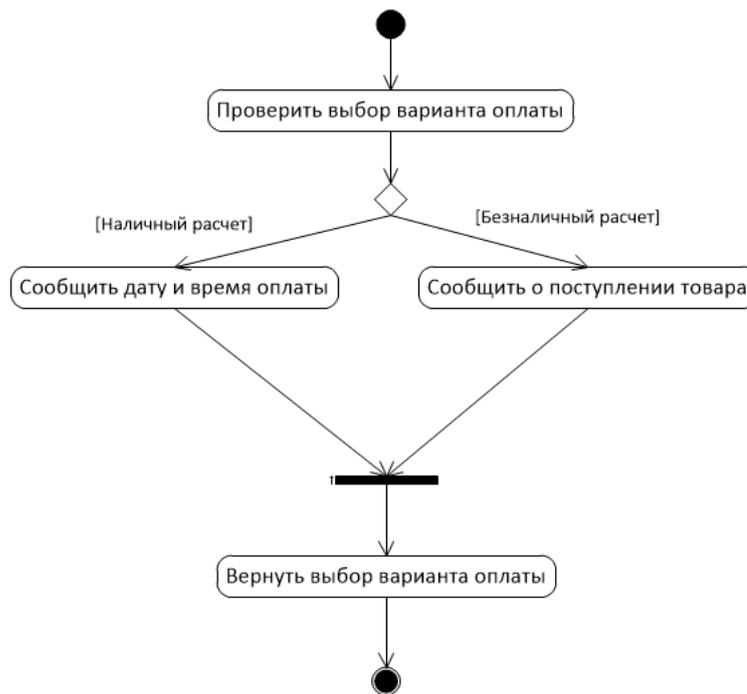
Соответствующие диаграммы состояний имеют вид

Для создания диаграмм состояний, которые входят в состав составного состояния, нужно:

1. Щелкнуть правой кнопкой мыши по Составному состоянию и выбрать пункт Схема.
2. Либо, в Проводнике по моделям выделить название составного состояния и создать новую

страницу с помощью кнопки .





### Задание практической работы

По образцу построить диаграмму состояний.

### Задание самостоятельной работы

В соответствии с индивидуальным вариантом, построить диаграмму состояний.

Перечень индивидуальных вариантов приведен в приложении А.

Отчет по практическому занятию выполняется в формате MS Word, который содержит экранные формы моделей согласно заданию.

### Контрольные вопросы

1. Каково назначение диаграммы состояний?
2. Назовите основные элементы диаграммы состояний.
3. Как создать диаграмму состояний в VISIO?
4. В чем отличие диаграммы классов и состояний?

**Форма отчета:** тетрадь, защита работы.

## Практическое занятие № 15-16

### Тема: Построение диаграммы Деятельности и диаграммы Последовательности

**Цель:** изучение основ создания диаграмм деятельности на языке UML, получение навыков построения диаграмм деятельности, применение приобретенных навыков для построения объектно-ориентированных моделей определенной предметной области. Изучение основ создания диаграмм последовательностей на языке UML, получение навыков построения диаграмм последовательностей, применение приобретенных навыков для построения объектно-ориентированных моделей определенной предметной области.

**Оборудование:** тетрадь, ручка, ПК

**Методические указания:** ознакомьтесь с теоретической частью, выполните задания, ответьте на контрольные вопросы

**Ход выполнения:**

### Краткие теоретические сведения

#### Построение диаграммы Деятельности

При моделировании поведения проектируемой или анализируемой системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и логической реализации выполняемых системой операций. Традиционно для этой цели использовались блок-схемы или структурные схемы алгоритмов. Каждая такая схема акцентирует внимание на последовательности выполнения определенных действий или элементарных операций, которые в совокупности приводят к получению желаемого результата.

Алгоритмические и логические операции, требующие выполнения в определенной последовательности, окружают нас постоянно. Например, чтобы позвонить по телефону, нам предварительно нужно снять трубку или включить его. Для приготовления кофе или заваривания чая необходимо вначале вскипятить воду. Чтобы выполнить ремонт двигателя автомобиля, требуется осуществить целый ряд нетривиальных операций, таких как разборка силового агрегата, снятие генератора и некоторых других.

С увеличением сложности системы строгое соблюдение последовательности выполняемых операций приобретает все более важное значение. Если попытаться заварить кофе холодной водой, то мы можем только испортить одну порцию напитка. Нарушение последовательности операций при ремонте двигателя может привести к его поломке или выходу из строя. Еще более катастрофические последствия могут произойти в случае отклонения от установленной последовательности действий при взлете или посадке авиалайнера, запуске ракеты, регламентных работах на АЭС.

Для моделирования процесса выполнения операций в языке UML используются так называемые **диаграммы деятельности**. Применяемая в них графическая нотация во многом похожа на нотацию диаграммы состояний, поскольку на диаграммах деятельности также присутствуют обозначения состояний и переходов. Отличие заключается в семантике состояний, которые используются для представления не деятельностей, а действий, и в отсутствии на переходах сигнатуры событий. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении этой операции в предыдущем состоянии. Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия, а дугами – переходы от одного состояния действия к другому.

Таким образом, диаграммы деятельности можно считать частным случаем диаграмм состояний. Основным направлением использования диаграмм деятельности является визуализация особенностей реализации операций классов, когда необходимо представить алгоритмы их выполнения. При этом каждое состояние может являться выполнением операции некоторого класса либо ее части, позволяя использовать диаграммы деятельности для описания реакций на внутренние события системы.

В контексте языка UML **деятельность (activity)** представляет собой некоторую совокупность отдельных вычислений, выполняемых автоматом. При этом отдельные элементарные вычисления могут приводить к некоторому результату или действию (action). На диаграмме деятельности отображается логика или последовательность перехода от одной деятельности к другой, при этом внимание фиксируется на результате деятельности. Сам же результат может привести к изменению состояния системы или возвращению некоторого значения.

#### ***Состояние действия и деятельности***

***Состояние деятельности (activity state)*** – состояние в графе деятельности, которое служит для представления процедурной последовательности действий, требующих определенного времени. Переход из состояния деятельности происходит после выполнения специфицированной в нем деятельности, при этом ключевое слово *do* в имени деятельности не указывается. Состояние деятельности не может иметь внутренних переходов, поскольку оно является элементарным.

Состояния деятельности могут быть подвергнуты дальнейшей декомпозиции, вследствие чего выполняемую деятельность можно представить с помощью других диаграмм деятельности. Состояния деятельности не являются атомарными, то есть могут быть прерваны. Предполагается, что для их завершения требуется заметное время.

Состояние деятельности можно представлять себе, как составное состояние, поток управления которого включает только другие состояния деятельности и действий.

***Состояние действия (action state)*** является специальным случаем состояния с некоторым входным действием и, по крайней мере, одним выходящим из состояния переходом. Этот переход неявно предполагает, что входное действие уже завершилось. Состояние действия не может иметь внутренних переходов, поскольку оно является элементарным. Обычное использование состояния действия заключается в моделировании одного шага выполнения алгоритма (процедуры) или потока управления.

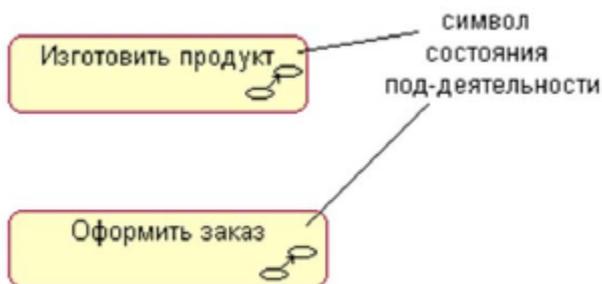
Графически состояние действия изображается фигурой, напоминающей прямоугольник, боковые стороны которого заменены выпуклыми дугами. Внутри этой фигуры записывается

выражение действия (action-expression), которое должно быть уникальным в пределах одной диаграммы деятельности.



Действие может быть записано на естественном языке, некотором псевдокоде или языке программирования. Никаких дополнительных или неявных ограничений при записи действий не накладывается. Рекомендуется в качестве имени простого действия использовать глагол с пояснительными словами. Если же действие может быть представлено в некотором формальном виде, то целесообразно записать его на том языке программирования, на котором предполагается реализовывать конкретный проект.

Иногда возникает необходимость представить на диаграмме деятельности некоторое сложное действие, которое, в свою очередь, состоит из нескольких более простых действий. В этом случае можно использовать специальное обозначение так называемого **состояния поддеятельности (subactivity state)**. Такое состояние является графом деятельности и обозначается специальной пиктограммой в правом нижнем углу символа состояния действия (рис. 2). Эта конструкция может применяться к любому элементу языка UML, который поддерживает «вложенность» своей структуры. При этом пиктограмма может быть дополнительно помечена типом вложенной структуры.



Каждая диаграмма деятельности должна иметь единственное начальное и единственное конечное состояния. Они имеют такие же обозначения, как и на диаграмме состояний. При этом каждая деятельность начинается в начальном состоянии и заканчивается в конечном состоянии. Саму диаграмму деятельности принято располагать таким образом, чтобы действия следовали сверху вниз. В этом случае начальное состояние будет изображаться в верхней части диаграммы, а конечное – в ее нижней части.

### Переходы

При построении диаграммы деятельности используются только такие переходы, которые срабатывают сразу после завершения деятельности или выполнения соответствующего действия. Этот переход переводит деятельность в последующее состояние сразу, как только закончится действие в предыдущем состоянии. На диаграмме такой переход изображается сплошной линией со стрелкой.

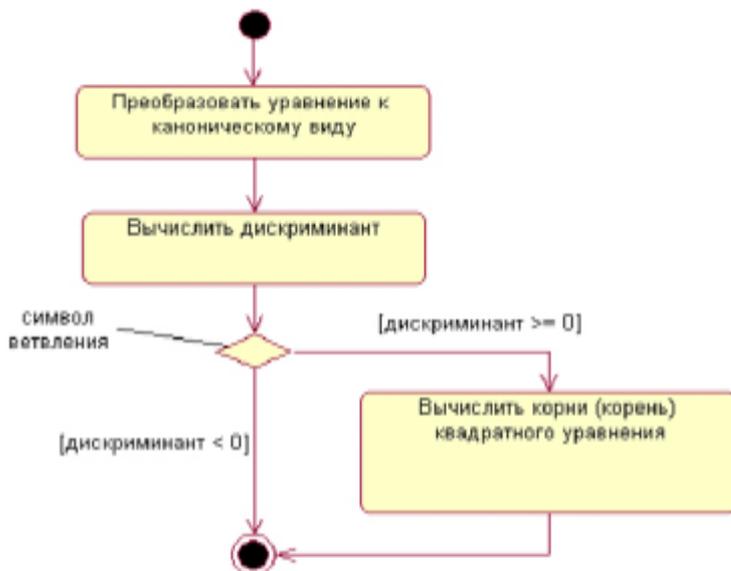
Если из состояния действия выходит единственный переход, то он может быть никак не помечен. Если же таких переходов несколько, то сработать может только один из них. Именно в этом случае для каждого из таких переходов должно быть явно записано сторожевое условие в прямых скобках. При этом для всех выходящих из некоторого состояния переходов должно выполняться требование истинности только одного из них. Подобный случай встречается тогда, когда последовательно выполняемая деятельность должна разделиться на альтернативные ветви в зависимости от значения некоторого промежуточного результата. Такая ситуация получила название ветвления, а для ее обозначения применяется специальный символ.

**Графически ветвление** на диаграмме деятельности обозначается небольшим ромбом, внутри которого нет никакого текста. В этот ромб может входить только одна стрелка от того состояния действия, после выполнения которого поток управления должен быть продолжен по одной из взаимно исключающих ветвей. Принято входящую стрелку присоединять к верхней или левой вершине символа ветвления. Выходящих стрелок может быть две или более, но для каждой из них явно указывается соответствующее сторожевое условие в форме булевого выражения.

В качестве **примера** рассмотрим фрагмент известного алгоритма нахождения корней квадратного уравнения. В общем случае после приведения уравнения второй степени к

каноническому виду:  $a * x * x + b * x + c = 0$  необходимо вычислить его дискриминант. Причем, в случае отрицательного дискриминанта уравнение не имеет решения на множестве действительных чисел, и дальнейшие вычисления должны быть прекращены. При неотрицательном дискриминанте уравнение имеет решение, корни которого могут быть получены на основе конкретной расчетной формулы.

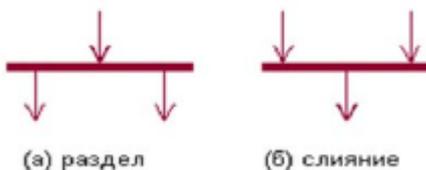
Графически фрагмент процедуры вычисления корней квадратного уравнения может быть представлен в виде диаграммы деятельности с тремя состояниями действия и ветвлением (рис. 3). Каждый из переходов, выходящих из состояния «Вычислить дискриминант», имеет сторожевое условие, определяющее единственную ветвь, по которой может быть продолжен процесс вычисления корней в зависимости от знака дискриминанта. Очевидно, что в случае его отрицательности, мы сразу попадаем в конечное состояние, тем самым завершая выполнение алгоритма в целом.



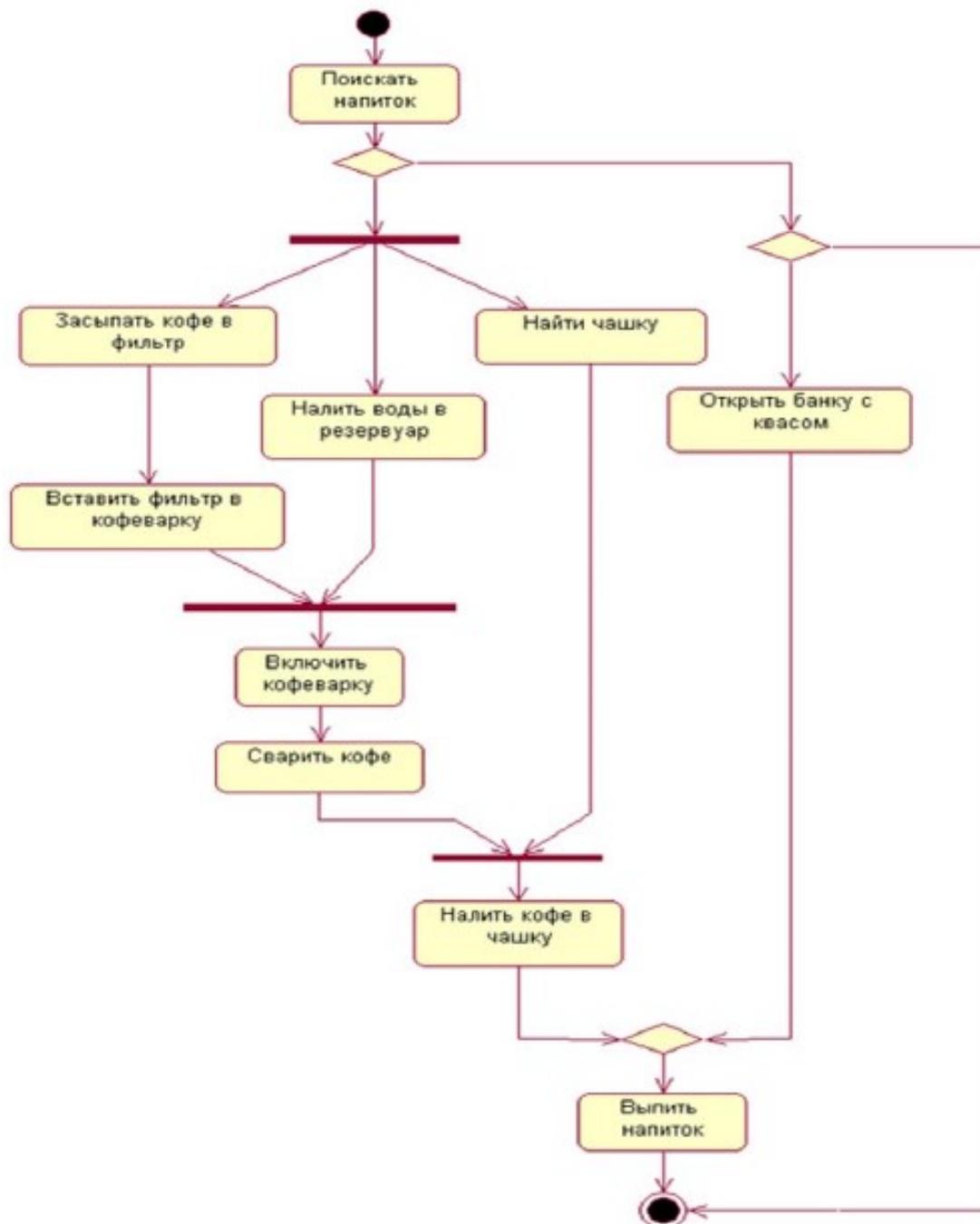
В рассмотренном примере, выполняемые действия соединяются в конечном состоянии. Однако это вовсе не является обязательным. Можно изобразить еще один символ ветвления, который будет иметь несколько входящих переходов и один выходящий.

Один из наиболее значимых недостатков обычных блок-схем или структурных схем алгоритмов связан с проблемой изображения параллельных ветвей отдельных вычислений. Поскольку распараллеливание вычислений существенно повышает общее быстродействие программных систем, необходимы графические примитивы для представления параллельных процессов. В языке UML для этой цели используется специальный символ для разделения и слияния параллельных вычислений или потоков управления. Таким символом является **прямая черточка**.

Такая черточка изображается отрезком горизонтальной линии, толщина которой несколько шире основных сплошных линий диаграммы деятельности. При этом разделение (concurrent fork) имеет один входящий переход и несколько выходящих. Слияние (concurrent join), наоборот, имеет несколько входящих переходов и один выходящий.



Для иллюстрации особенностей параллельных процессов выполнения действий рассмотрим **пример** с приготовлением напитка. Достоинство этого примера состоит в том, что он практически не требует никаких дополнительных пояснений в силу своей очевидности.



Хотя диаграмма деятельности предназначена для моделирования поведения систем, время в явном виде отсутствует на этой диаграмме. Ситуация здесь во многом аналогична диаграмме состояний.

Таким образом, диаграмма деятельности есть не что иное, как специальный случай диаграммы состояний, в которой все или большинство состояний являются действиями или состояниями поддеятельности. А все или большинство переходов являются переходами, которые срабатывают по завершении действий или под-деятельностей в состояниях источника.

### **Дорожки**

Диаграммы деятельности могут быть использованы не только для спецификации алгоритмов вычислений или потоков управления в программных системах. Не менее важная область их применения связана с моделированием бизнес-процессов. Действительно, деятельность любой компании (фирмы) также представляет собой не что иное, как совокупность отдельных действий, направленных на достижение требуемого результата. Однако применительно к бизнес-процессам желательно выполнение каждого действия ассоциировать с конкретным подразделением компании. В этом случае подразделение несет ответственность за реализацию отдельных действий, а сам бизнес-процесс представляется в виде переходов действий из одного подразделения к другому.

Для моделирования этих особенностей в языке UML используется специальная конструкция, получившее название *дорожки (swimlanes)*. Имеется в виду визуальная аналогия с плавательными

дорожками в бассейне, если смотреть на соответствующую диаграмму. При этом все состояния действия на диаграмме деятельности делятся на отдельные группы, которые отделяются друг от друга вертикальными линиями. Две соседние линии и образуют дорожку, а группа состояний между этими линиями выполняется отдельным подразделением (отделом, группой, отделением, филиалом) компании.



Названия подразделений явно указываются в верхней части дорожки. Пересекать линию дорожки могут только переходы, которые в этом случае обозначают выход или вход потока управления в соответствующее подразделение компании. Порядок следования дорожек не несет какой-либо семантической информации и определяется соображениями удобства.

В качестве примера рассмотрим фрагмент диаграммы деятельности торговой компании, обслуживающей клиентов по телефону. Подразделениями компании являются отдел приема и оформления заказов, отдел продаж и склад.

Этим подразделениям будут соответствовать три дорожки на диаграмме деятельности, каждая из которых специфицирует зону ответственности подразделения. В данном случае диаграмма деятельности включает в себе не только информацию о последовательности выполнения рабочих действий, но и о том, какое из подразделений торговой компании должно выполнять то или иное действие.

Из указанной диаграммы деятельности сразу видно, что после принятия заказа от клиента отделом приема и оформления заказов осуществляется распараллеливание деятельности на два потока (переход-разделение). Первый из них остается в этом же отделе и связан с получением оплаты от клиента за заказанный товар. Второй инициирует выполнение действия по подбору товара в отделе продаж (модель товара, размеры, цвет, год выпуска и пр.). По окончании этой работы инициируется действие по отпуску товара со склада. Однако подготовка товара к отправке в торговом отделе начинается только после того, как будет получена оплата за товар от клиента и товар будет отпущен со склада (переход-соединение). Только после этого товар отправляется клиенту, переходя в его собственность.

### **Объекты**

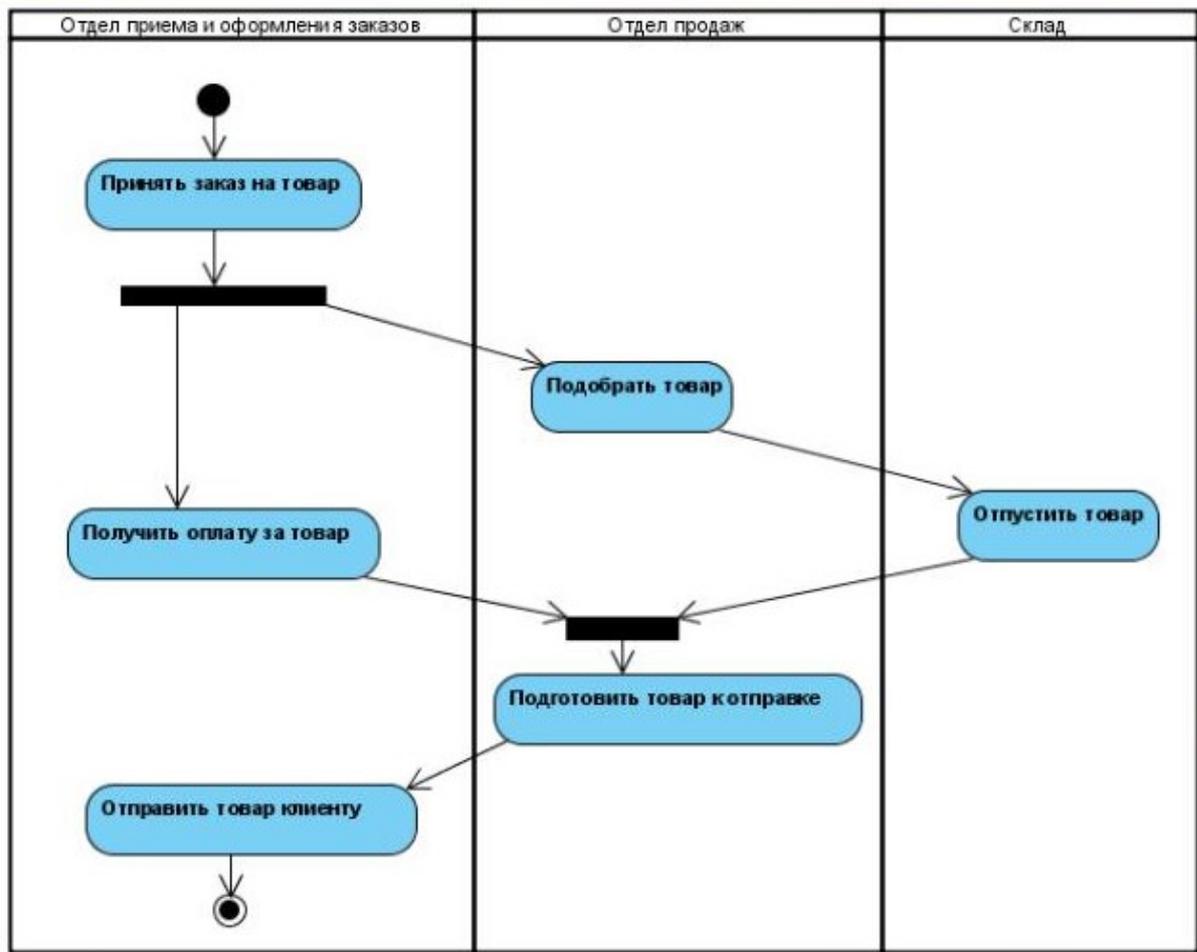
В общем случае действия на диаграмме деятельности выполняются над теми или иными объектами. Эти объекты либо инициируют выполнение действий, либо определяют некоторый результат этих действий. При этом действия специфицируют вызовы, которые передаются от одного объекта графа деятельности к другому. Поскольку в таком ракурсе объекты играют определенную роль в понимании процесса деятельности, иногда возникает необходимость явно указать их на диаграмме деятельности.

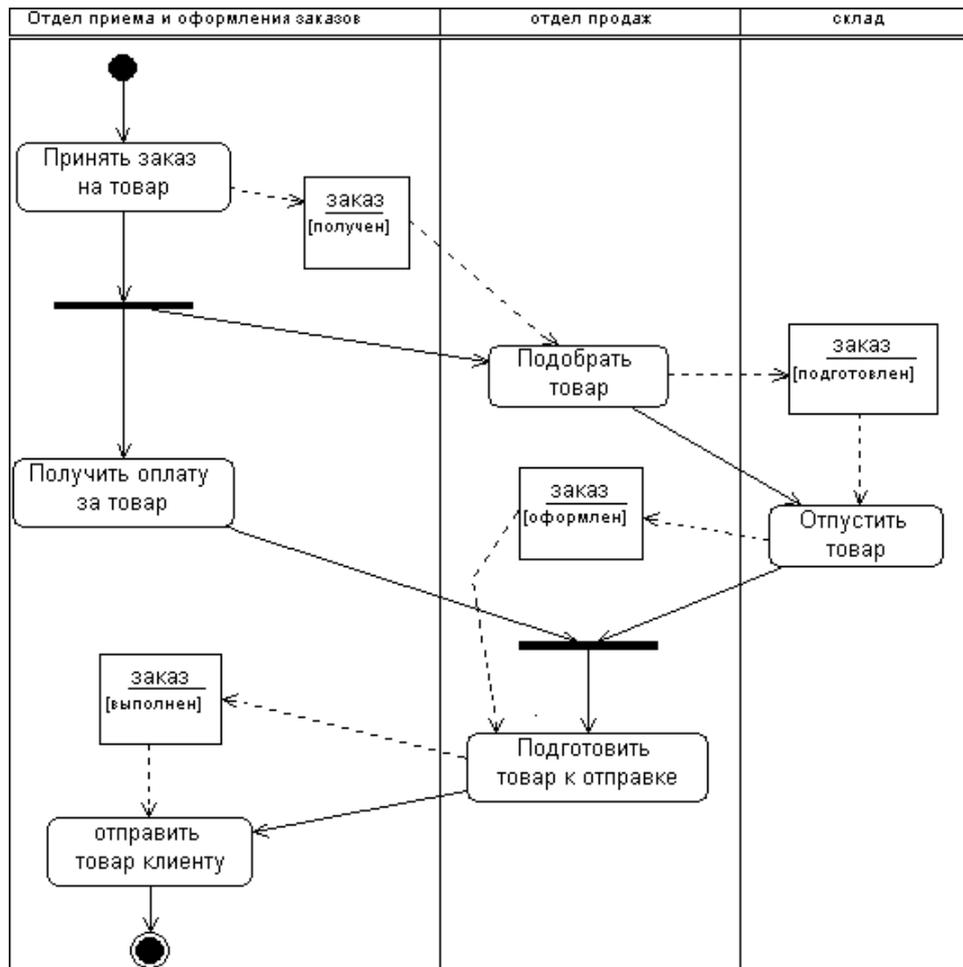
Для **графического представления объектов** используются прямоугольник класса, с тем отличием, что имя объекта подчеркивается. Далее после имени может указываться характеристика состояния объекта в прямых скобках. Такие прямоугольники объектов присоединяются к состояниям действия отношением зависимости пунктирной линией со стрелкой. Соответствующая зависимость определяет состояние конкретного объекта после выполнения предшествующего действия.

На диаграмме деятельности с дорожками расположение объекта может иметь некоторый дополнительный смысл. А именно, если объект расположен на границе двух дорожек, то это может означать, что переход к следующему состоянию действия в соседней дорожке ассоциирован с готовностью некоторого документа (объект в некотором состоянии). Если же объект целиком

расположен внутри дорожки, то и состояние этого объекта целиком определяется действиями данной дорожки.

Возвращаясь к предыдущему примеру с торговой компанией, можно заметить, что центральным объектом процесса продажи является заказ или вернее состояние его выполнения. Вначале до звонка от клиента заказ как объект отсутствует и возникает лишь после такого звонка. Однако этот заказ еще не заполнен до конца, поскольку требуется еще подобрать конкретный товар в отделе продаж. После его подготовки он передается на склад, где вместе с отпуском товара заказ окончательно дооформляется. Наконец, после получения подтверждения об оплате товара эта информация заносится в заказ, и он считается выполненным и закрытым. Данная информация может быть представлена графически в виде модифицированного варианта диаграммы деятельности этой же торговой компании.

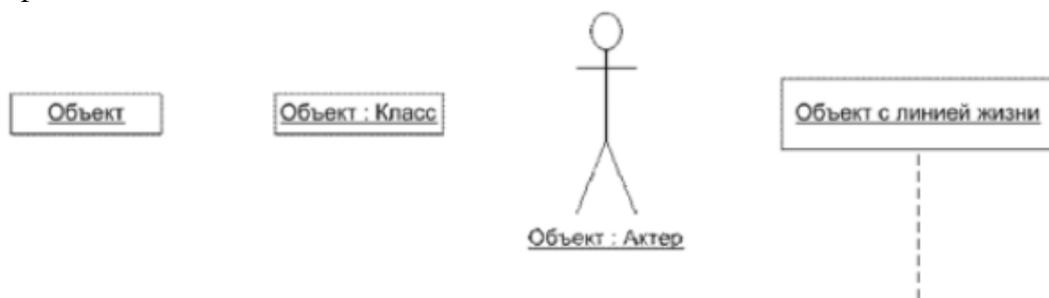




### Построение диаграммы Последовательности

Диаграммы последовательностей описывают взаимодействия множества объектов, включая сообщения, которыми они обмениваются.

В отличие от диаграммы классов, на которой изображаются абстрактные элементы в виде классов, на диаграмме последовательностей используются конкретные экземпляры классов – **объекты**. Объекты отображаются прямоугольником без полей. Для того чтобы подчеркнуть, что это экземпляр абстрактной сущности, название объекта подчеркивается. При необходимости через двоеточие после названия можно указать сущность (класс) экземпляром которой является этот объект. Отметим, что объект может быть экземпляром не только класса, но и других абстракций, например, актера. Обратите внимание, что при указании в качестве классификатора актера изменится графическое обозначение объекта.

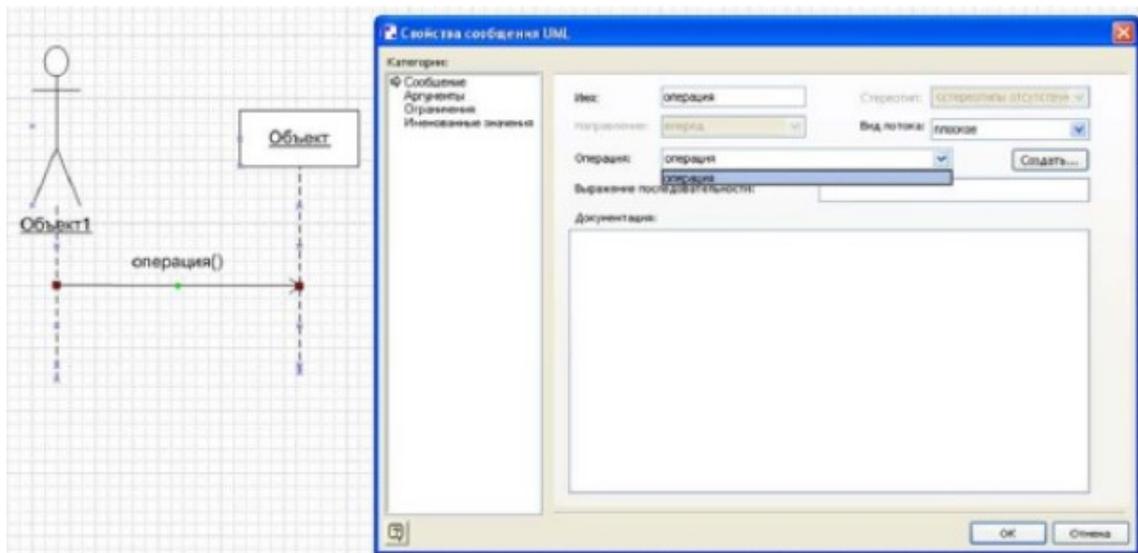


На диаграмме последовательностей у объекта может присутствовать **линия жизни**, на которой отмечаются происходящие с объектом события. Линия жизни отображается пунктирной линией.

Между собой объекты могут быть связаны связями. **Связь** – это экземпляр отношения ассоциация, и имеет такое же графическое обозначение, что и ассоциация.

На диаграмме последовательностей объекты обмениваются сообщениями. **Сообщение** – это спецификация передачи данных от одного объекта другому, который предполагает какое-то ответное действие. Графически сообщение обозначается сплошной линией со стрелкой.

Часто операция вызывает какую-либо операцию в объекте. Очевидно, что класс, экземпляром которого является объект, должен иметь такую операцию. Привязка сообщения к операции класса объекта выполняется в свойствах сообщения.



При построении диаграммы классов обычно определяются только основные свойства сущностей, а такие детали, как операции, удобно создавать при построении диаграммы последовательности, для чего в свойствах сообщения UML есть кнопка создания операции.

Диаграммы последовательностей, как и другие диаграммы для отображения динамических свойств системы, могут быть выполнены в контексте многих сущностей UML. Они могут описывать поведение системы в целом, подсистемы, класса или операции класса и др. К сожалению, Visio недостаточно гибка в плане поддержки раскрытия содержания отдельных элементов с помощью других диаграмм. Например, кликнув правой кнопкой мыши по классу можно обнаружить, что для его описания можно создать лишь диаграммы классов, состояний и деятельности. Поэтому возможность привязать диаграмму последовательностей к элементу, который она реализует, средствами Visio невозможно, эту связь нужно подразумевать.

Диаграммы последовательностей будем делать в контексте прецедентов с диаграммы прецедентов, реализуя те функции, которые должна выполнять наша система.

При построении динамических диаграмм используется уже разработанная структура информационной системы. Для диаграммы последовательностей не нужно придумывать объекты, а достаточно определить, экземпляры каких классов участвуют в этом действии.

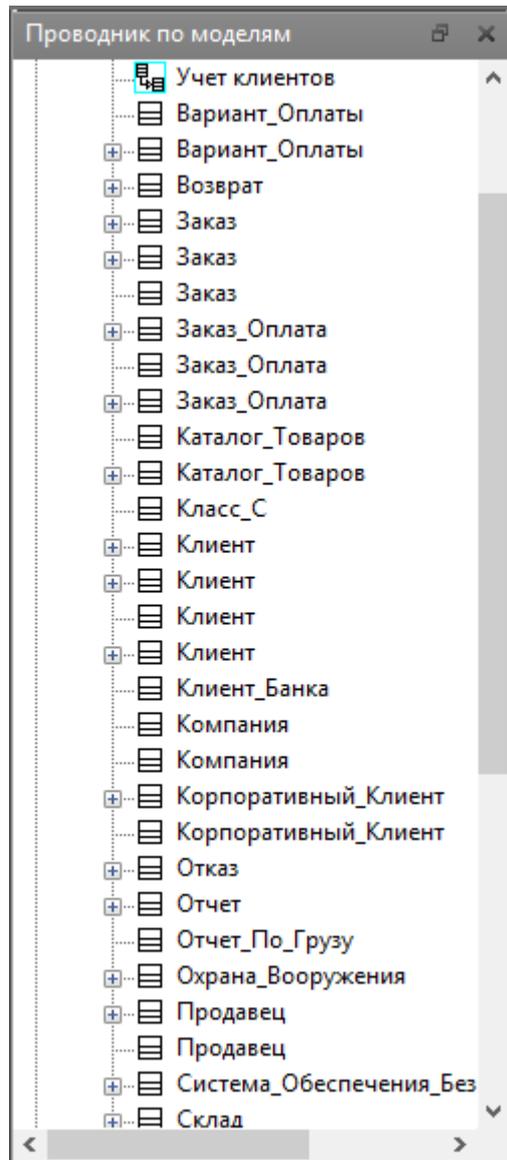
Определив необходимые объекты (как экземпляры классов, так и экземпляры актеров), вторым этапом построения диаграмм последовательностей определяются сообщения, пересылаемые между актерами. Фактически определяется последовательность шагов, для выполнения нужного действия.

### **Методика выполнения**

#### **Построение диаграммы Деятельности**

В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

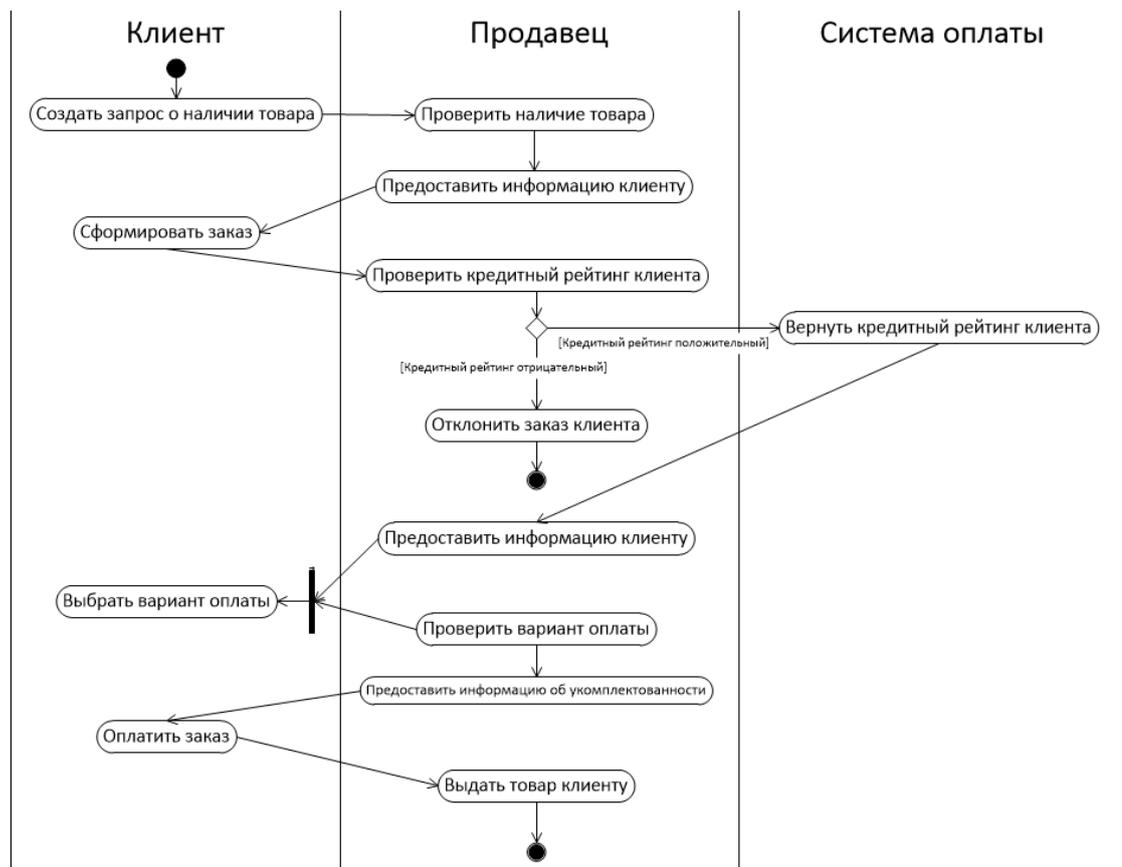
1. Запустите MS Visio.
2. **Откройте файл, созданный в предыдущей работе и содержащий диаграмму классов.**
3. В проводнике по моделям должны отображаться все классы и диаграммы, созданные ранее.



Опишем с помощью диаграммы деятельности процесс формирования заказа и выдачу товара. В бизнес-процессе участвуют 3 действующих лица: клиент, продавец и система оплаты. Следовательно, необходимо добавить 3 дорожки для распределения ответственности между этими лицами.

Для этого, в файле с диаграммой классов, созданной в практическом занятии 8, необходимо проделать следующие действия:

6. Щелкнуть правой кнопкой мыши по классу *Заказ*.
7. В контекстном меню выбрать пункт *Схемы*.
8. Нажать кнопку *Создать* и выбрать *Деятельность*.
9. Переименовать созданный лист в *Деятельность-Заказ*.
10. Построить диаграмму деятельности для класса *Заказ*. Для это выполните действия, описанные ниже.
  - а. Добавить 3 элемента *Дорожка* и изменить их названия на *Клиент*, *Продавец* и *Система оплаты* соответственно.
  - б. Добавить элементы *Начальное состояние*, *Конечное состояние*, *Состояние действия*, *Решение*, *Переход (объединение)*, изменить их названия и задать расположение.



### Построение диаграммы Последовательности

1. В проводнике по моделям должны отображаться все классы и диаграммы, созданные ранее. Построим диаграмму последовательности для варианта использования «Обеспечить покупателя информацией». Для этого добавим на диаграмму последовательности линии жизни и соотнесем объекты с актерами, инициирующими вариант использования «Обеспечить покупателя информацией», и с необходимыми классами.

Для добавления диаграммы последовательности в проект MS Visio выполните следующие действия:

1. В проводнике по моделям найдите ветку «Основной пакет».
2. Нажмите по ней правой кнопкой мыши > Создать ...
3. В контекстном меню выберите пункт «Схема последовательностей».

Добавим сообщения, которыми обмениваются объекты для исполнения варианта использования.

Если объект имеет операцию (посмотреть в практическом занятии №8 «Диаграммы классов» наличие операции у класса, которому принадлежит объект).

1. Из группы фигур «Последовательности UML» добавить три фигуры типа «Линия жизни объекта». Для изменения названия необходимо дважды щелкнуть левой кнопкой мыши по фигуре. Откроется окно свойств объекта и, если в данном файле нет ранее созданных классов, окно создания нового класса. В данном примере необходимо создать три класса «Товар», «Каталог товаров» и «Заказ» и соответственно три объекта с такими же названиями.

2. С помощью поиска фигур найти фигуру «Актер» и добавить ее в рабочую область. Двойным щелчком левой кнопки мыши задать имя «Клиент».

3. Добавить фигуру «Линия жизни» и соедините ее начало с фигурой «Клиент».
4. Протянуть все линии жизни вниз листа.
5. Добавить фигуры «Сообщение» и соединить, руководствуясь следующими принципами:

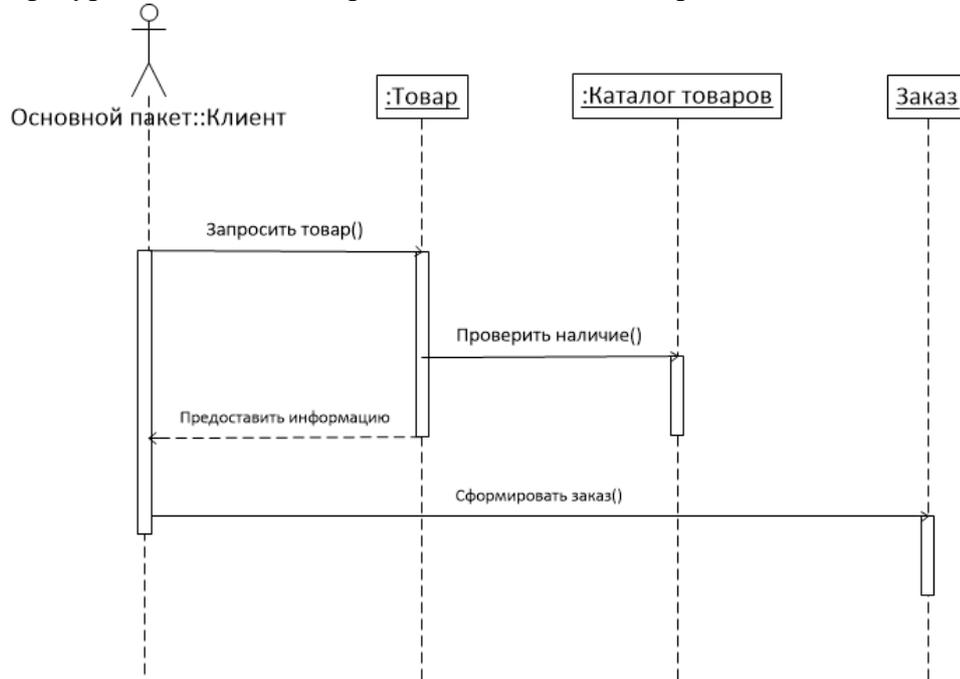
5.1. Соединить фигурой «Сообщение» линию жизни клиента с линией жизни объекта товар. Двойным щелчком по сообщению открыть окно свойств и выбрать операцию запросить товар.

5.2. Соединить фигурой «Сообщение» линию жизни клиента с линией жизни объекта заказ. Двойным щелчком по сообщению открыть окно свойств и выбрать операцию сформировать заказ.

5.3. Соединить фигурой «Сообщение» линию жизни объекта товар с линией жизни объекта каталог товаров. Двойным щелчком по сообщению открыть окно свойств и выбрать операцию проверить наличие.

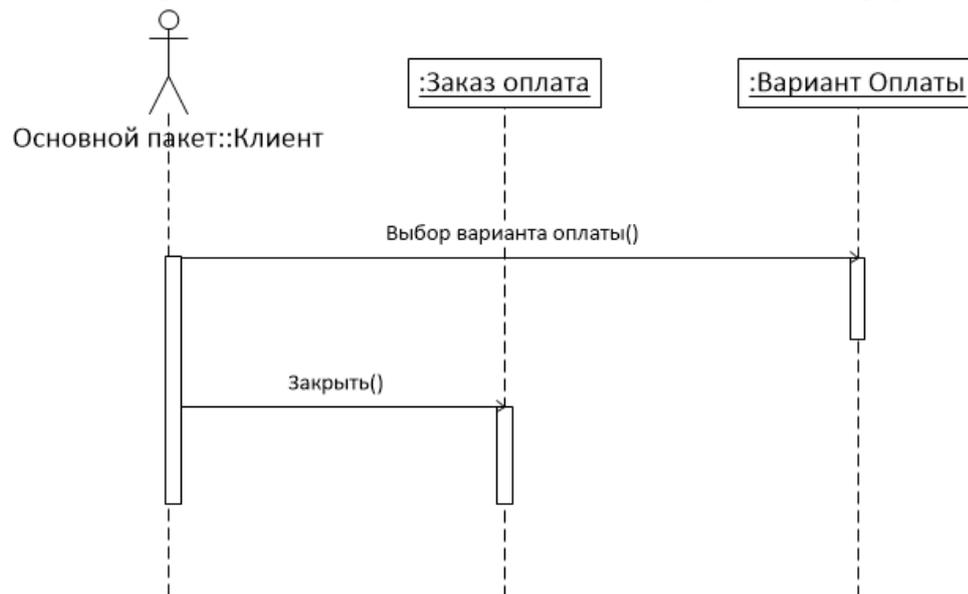
5.4. Соединить фигурой «Сообщение (возврат)» линию жизни объекта товар и линией жизни клиента. Двойным щелчком по сообщению открыть окно свойств и задать текст сообщения «Предоставить информацию».

6. Добавить фигуры «Активация» и расположить их на диаграмме.

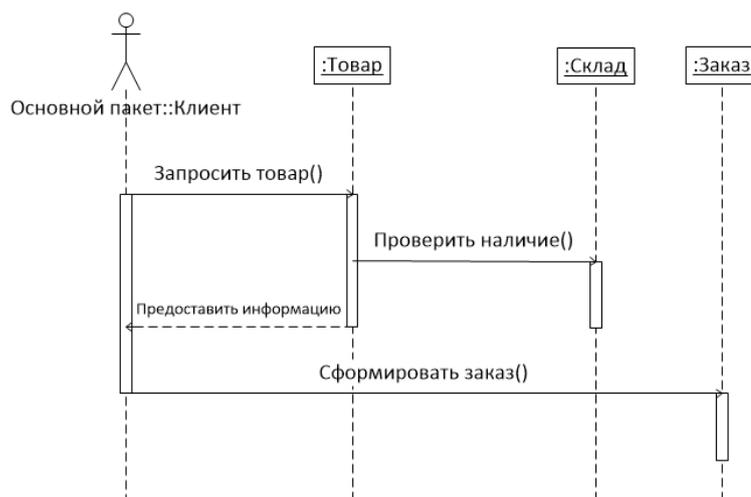


При построении диаграмм последовательностей можно вносить коррективы в диаграмму классов. Если объект класса получает новую операцию, то она добавляется в соответствующий класс на диаграмме классов как метод.

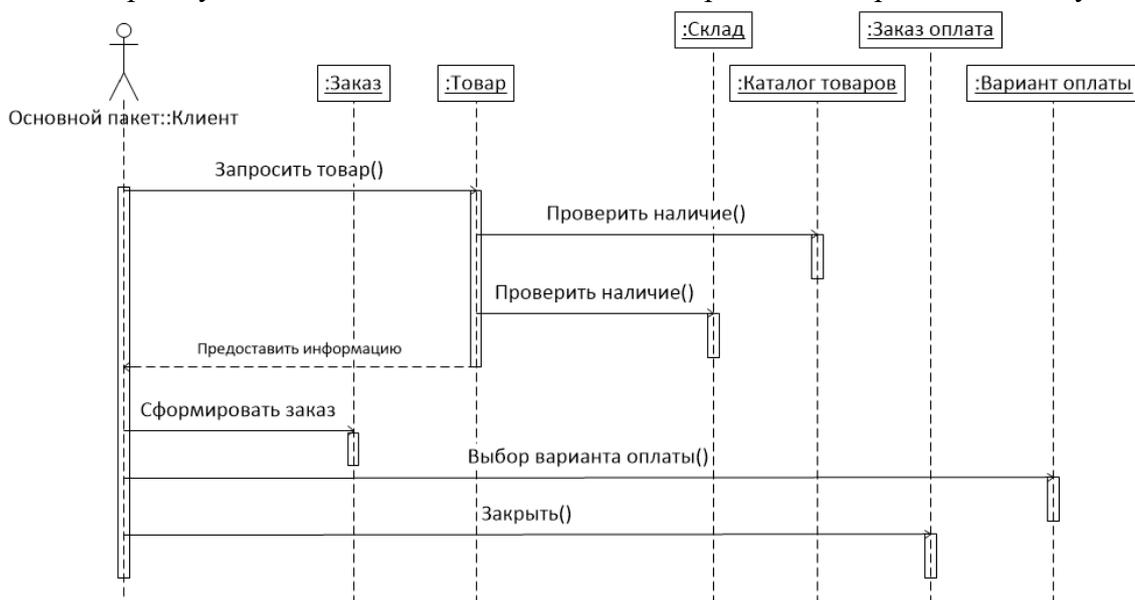
Построим диаграмму последовательности для варианта использования «Согласовать условия оплаты». Действия по построению диаграммы аналогичны построению диаграммы последовательности для варианта использования «Обеспечить покупателя информацией».



Построим диаграмму последовательности для варианта использования «Заказать товар со склада». Действия по построению диаграммы аналогичны построению предыдущих диаграмм последовательностей.



Построим диаграмму последовательности для системы продажи товаров по каталогу.



### Задание практической работы

По образцу построить диаграмму состояний.

### Задание самостоятельной работы

В соответствии с индивидуальным вариантом, построить диаграмму состояний.

Перечень индивидуальных вариантов приведен в приложении А.

Отчет по практическому занятию выполняется в формате MS Word, который содержит экранные формы моделей согласно заданию.

### Контрольные вопросы

1. Дайте определение понятию «диаграмма деятельности».
2. Опишите назначение диаграммы деятельности.
3. Дайте определение понятиям «состояние деятельности» и «состояние действия».

Графическое изображение состояния.

4. Приведите пример ветвления и параллельных потоков управления процессами на диаграмме деятельности.

5. Какие переходы используются на диаграмме деятельности?

6. Что представляет собой дорожка на диаграмме деятельности?

7. Как графически изображаются объекты на диаграмме деятельности?

**Форма отчета:** тетрадь, защита работы.

## Практическое занятие № 17-18

**Тема:** Разработка тестового сценария. Оценка необходимого количества тестов

**Цель:** усвоить знание о видах тестирования; освоить способы обнаружения и фиксирования ошибок. Научиться оценивать необходимое количество тестов.

**Оборудование:** тетрадь, ручка, ПК

**Методические указания:** ознакомьтесь с теоретической частью, выполните задания, ответьте на контрольные вопросы

**Ход выполнения:**

### Теоретические сведения

#### Разработка тестового сценария

Software Configuration Management или Конфигурационное управление подразумевает под собой комплекс методов, направленных на то, чтобы систематизировать изменения, вносимые разработчиками в программный продукт в процессе его разработки и сопровождения, сохранить целостность системы после изменений, предотвратить нежелательные и непредсказуемые эффекты, а также сделать процесс внесения изменений более формальным.

К процедурам можно отнести создание резервных копий, контроль исходного кода, требований проекта, документации и т. д. Степень формальности выполнения данных процедур зависит от размеров проекта, и при правильной ее оценке данная концепция может быть очень полезна. Конфигурационное управление требует выполнения множества трудоемких рутинных операций. На практике, в большинстве случаев, для конфигурационного управления применяются специальные системы контроля версий исходного кода программ. В качестве примера рассмотрим информационную систему Subversion. Это бесплатная система управления версиями с открытым исходным кодом.

Subversion позволяет управлять файлами и каталогами, а так же сделанными в них изменениями во времени. Это позволяет восстановить более ранние версии данных, предоставляет возможность изучить историю всех изменений.

Заповеди по отладки программного средства, предложенные Г. Майерсом.

Заповедь 1. Считайте тестирование ключевой задачей разработки ПС, поручайте его самым квалифицированным и одаренным программистам, нежелательно тестировать свою собственную программу.

Заповедь 2. Хорош тот тест, для которого высока вероятность обнаружить ошибку, а не тот, который демонстрирует правильную работу программы.

Заповедь 3. Готовьте тесты как для правильных, так и для неправильных данных.

Заповедь 4. Документируйте пропуск тестов через компьютер, детально изучайте результаты каждого теста, избегайте тестов, пропуск которых нельзя повторить.

Заповедь 5. Каждый модуль подключайте к программе только один раз, никогда не изменяйте программу, чтобы облегчить ее тестирование.

Заповедь 6. Пропускайте заново все тесты, связанные с проверкой работы какой-либо программы ПС или ее взаимодействия с другими программами, если в нее были внесены изменения (например, в результате устранения ошибки).

#### Оценка необходимого количества тестов

Сколько тестов понадобится, чтобы обнаружить ошибки?

Для ответа на этот вопрос предлагается следующая модель. Утверждается, что вероятность успешности очередного теста зависит от процента оставшихся ошибок. Последние ошибки обнаружить сложнее. (Заметим, что данное утверждение вполне соответствует опыту.)

Пусть  $T_n$  — среднее количество тестов, необходимых для обнаружения  $n$  ошибок,  $N$  — число ошибок в программе. Для оценки  $T_n$  предлагается следующая формула:

$$T_n = \alpha \sum_{k=0}^{n-1} \frac{1}{N-k}.$$

Здесь  $\alpha$  — некий коэффициент, значение которого надо определять экспериментально. Но мы заниматься этим не будем, поскольку формула для вычисления  $T_n$  будет интересовать нас не сама по себе, а лишь как база для последующих рассуждений.

Мы хотим оценить количество тестов, которое потребуется для нахождения всех  $N$  ошибок. Посчитать напрямую  $T_n$  мы не можем, поскольку не знаем коэффициента  $\alpha$ . Чтобы исключить его из вычислений, перейдем от абсолютных величин к относительным. Попробуем оценить, какую часть от всех необходимых тестов придется выполнить для того, чтобы найти первые  $n$  ошибок

$$P = \frac{T_n}{T_N} = \frac{\alpha \sum_{k=0}^{n-1} \frac{1}{N-k}}{\alpha \sum_{k=0}^{N-1} \frac{1}{N-k}} = \frac{\sum_{k=0}^{n-1} \frac{1}{N-k}}{\sum_{k=0}^{N-1} \frac{1}{N-k}}.$$

Теперь осталось подставить в формулу для Р конкретные значения N и n. Результаты — очень интересные — представлены в таблицах. В первой таблице N изменяется от 10 до 100, во второй таблице — от 1 до 10. Последняя таблица построена специально для новичков, программы которых настолько малы по размерам, что в них просто не найдется места для сотни ошибок.

Таблица. Средний процент тестов, необходимых для обнаружения заданного процента ошибок (в зависимости от общего числа ошибок в программе)

Процент найденных ошибок (n/N)	Общее число ошибок в программе (N)									
	10	20	30	40	50	60	70	80	90	100
10	3	3	3	2	2	2	2	2	2	2
20	7	6	5	5	5	5	5	4	4	4
30	11	10	9	8	8	8	7	7	7	7
40	16	14	13	13	11	11	10	10	10	10
50	22	19	17	16	15	15	14	14	14	13
60	29	24	22	21	20	19	19	18	18	18
70	37	32	29	27	26	25	25	24	23	23
80	49	42	39	36	35	34	33	32	31	31
90	66	58	54	51	49	48	46	45	44	44
100	100	100	100	100	100	100	100	100	100	100

Оказывается, что при N = 10 первые 22% тестов обнаружат половину всех ошибок (5 штук). Для того чтобы обнаружить две следующие ошибки, количество тестов придется увеличить в 1,7 раза — до 37%. Поиск следующих двух ошибок потребует увеличения количества тестов еще в 1,8 раза — до 66%. И наконец, поиск последней ошибки потребует оставшихся 34% тестов.

Чем больше ошибок в программе, тем дороже обойдется поиск последних ошибок. При N = 50 для обнаружения 50% ошибок достаточно 15% тестов, 70% ошибок будут найдены с помощью 26%, 90% ошибок — с помощью 49% тестов. Поиск последних 10% ошибок будет стоить дороже, чем поиск первых 90%!

При увеличении количества ошибок с 10 до 100 стоимость поиска последних 10% ошибок возрастает с 34% тестов до 66%.

Заметим, что речь идет не об абсолютном числе тестов, а о их доле. Поскольку программа, содержащая 100 естественных ошибок, скорее всего, сложнее программы, в которой ошибок только 10, есть основания полагать, что общее число тестов также будет возрастать. То есть придется брать больший процент от большего числа тестов

В следующей таблице N изменяется от 1 до 10. В этом случае говорить о процентах найденных ошибок смысла нет. Поэтому в боковике записаны не относительные, а абсолютные значения. Поскольку указанное количество ошибок вполне реально для учебных программ, интересно было сравнить модельные данные, приведенные в таблице, с реальными данными из практики. Надо только помнить, во-первых, что речь идет о средних значениях. А во-вторых, что данные в таблице получены из статистической модели. А статистика любит большие числа.

Таблица. Средний процент тестов, необходимых для обнаружения заданного количества ошибок (в зависимости от общего числа ошибок в программе)

Количество найденных ошибок ( $n$ )	Общее число ошибок в программе ( $N$ )									
	1	2	3	4	5	6	7	8	9	10
1	100	33	18	12	9	7	6	5	4	3
2	—	100	45	28	20	15	12	10	8	7
3	—	—	100	52	34	35	20	16	13	11
4	—	—	—	100	56	39	29	23	19	16
5	—	—	—	—	100	59	42	33	26	22
6	—	—	—	—	—	100	61	45	35	29
7	—	—	—	—	—	—	100	63	47	37
8	—	—	—	—	—	—	—	100	65	49
9	—	—	—	—	—	—	—	—	100	66
10	—	—	—	—	—	—	—	—	—	100

Имея такие оценки относительного количества тестов, в конкретном проекте можно перейти к абсолютным величинам.

Поскольку нам известно, сколько тестов нам понадобилось для нахождения  $n$  ошибок, легко оценить, сколько понадобится для нахождения оставшихся. Если количество ошибок в программе оценено в 10 и для обнаружения первых пяти потребовалось, например, 7 тестов, то для поиска двух следующих ошибок количество тестов придется довести до 12 (5 дополнительных тестов на 2 ошибки), для поиска следующих двух — до 21 (9 дополнительных тестов на 2 ошибки). Все 10 ошибок есть надежда найти за 32 теста.

Подобные оценки можно использовать для планирования времени и ресурсов, выделяемых для процесса тестирования.

### Задание практической работы

#### Методика выполнения

#### Разработка тестового сценария

Задача: Найти минимальный набор тестов для программы нахождения вещественных корней квадратного уравнения  $ax^2 + bx + c = 0$ .

Решение:

Номер теста	a	b	c	Ожидаемый результат	Что проверяется
1	2	-5	2	$x_1 = 2, x_2 = 0.5$	Случай вещественных корней
2	3	2	5	Сообщение	Случай комплексных корней
3	3	-12	0	$x_1 = 4, x_2 = 0$	Нулевой корень
4	0	0	10	Сообщение	Неразрешимое уравнение
5	0	0	0	Сообщение	Неразрешимое уравнение
6	0	5	17	Сообщение	Неразрешимое уравнение
7	9	0	0	$x_1 = x_2 = 0$	Нулевые корни

Таким образом для этой задачи предлагается минимальный набор функциональных тестов, исходя из 7 классов выходных данных.

#### Оценка необходимого количества тестов

Для задачи нахождения вещественных корней квадратного уравнения для нахождения 4 ошибок нам понадобилось 7 тестов.

Следовательно, легко оценить, сколько понадобится для нахождения оставшихся. Если количество ошибок в программе оценено в 4 и для обнаружения первых двух потребовалось, например, 4 теста, то для поиска двух следующих ошибок количество тестов придется довести до 10 (6 дополнительных тестов на 2 ошибки). Все 4 ошибки есть надежда найти за 10 тестов.

#### Задание самостоятельной работы

В соответствии с индивидуальным вариантом, разработать тестовый сценарий и оценить количество необходимых тестов.

Перечень индивидуальных вариантов приведен в приложении В.

### **Контрольные вопросы**

1. Что такое тестирование? Что является объектами тестирования?
2. Опишите виды тестирования.
3. Поясните понятия «тест», «тестовые данные», «тестовый эксперимент».

**Форма отчета:** тетрадь, защита работы.

## **Практическое занятие № 19-20**

**Тема:** Разработка тестовых пакетов

**Цель:** получить навыки разработки тестовых пакетов

**Оборудование:** тетрадь, ручка, ПК

**Методические указания:** ознакомьтесь с теоретической частью, выполните задания, ответьте на контрольные вопросы

**Ход выполнения:**

### **Теоретические сведения**

В ходе выполнения лабораторной работы провести тестирование по принципу «белого ящика».

Критерии покрытия кода:

- покрытие операторов — каждая ли строка исходного кода была выполнена и протестирована;
- покрытие условий — каждая ли точка решения (вычисления истинно ли или ложно выражение) была выполнена и протестирована;
- покрытие путей — все ли возможные пути через заданную часть кода были выполнены и протестированы;
- покрытие функций — каждая ли функция программы была выполнена;
- покрытие вход/выход — все ли вызовы функций и возвраты из них были выполнены;
- покрытие значений параметров — все ли типовые и граничные значения параметров были проверены.

### **Метод покрытия операторов**

Целью этого метода тестирования является выполнение каждого оператора программы хотя бы один раз.

### **Метод покрытия решений (покрытия переходов)**

Согласно методу покрытия решений каждое направление перехода должно быть реализовано, по крайней мере, один раз. Этот метод включает в себя критерий покрытия операторов, так как при выполнении всех направлений переходов выполняются все операторы, находящиеся на этих направлениях.

### **Метод покрытия условий**

Этот метод может дать лучшие результаты по сравнению с предыдущими. В соответствии с методом покрытия условий записывается число тестов, достаточное для того, чтобы все возможные результаты каждого условия в решении выполнялись, по крайней мере, один раз.

### **Метод покрытия решений/условий**

Критерий покрытия решений/условий требует такого достаточного набора тестов, чтобы все возможные результаты каждого условия выполнялись по крайней мере один раз, все результаты каждого решения выполнялись по крайней мере один раз и, кроме того, каждой точке входа передавалось управление по крайней мере один раз.

Недостатки метода:

- не всегда можно проверить все условия;
- невозможно проверить условия, которые скрыты другими условиями;
- недостаточная чувствительность к ошибкам в логических выражениях.

### **Метод комбинаторного покрытия условий**

Критерий комбинаторного покрытия условий удовлетворяет также и критериям покрытия решений, покрытия условий и покрытия решений/условий.

Этот метод требует создания такого числа тестов, чтобы все возможные комбинации результатов условия в каждом решении выполнялись по крайней мере один раз.

### **Задание практической работы**

#### **Методика выполнения**

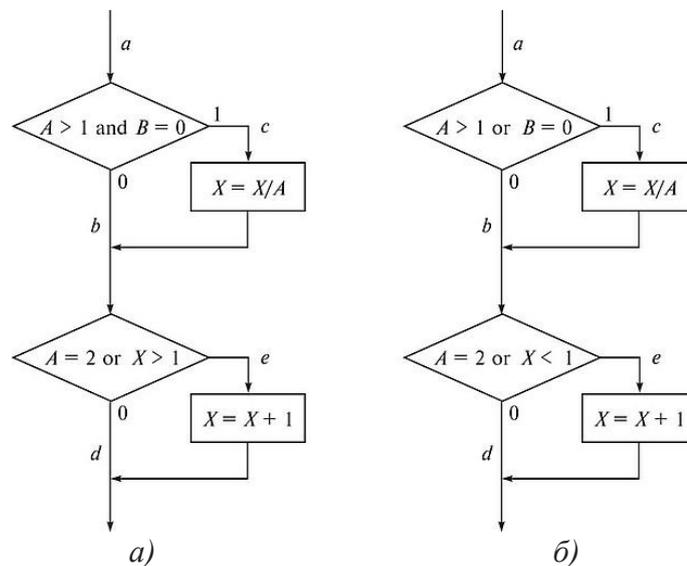
#### **Пример метода покрытия операторов**

Если для тестирования задать значения переменных  $A = 2, B = 0, X = 3$ , будет реализован путь ACE, т.е. каждый оператор программы выполнится один раз. Но если внести в алгоритм ошибки — заменить в первом условии **and** на **or**, а во втором  $X > 1$  на  $X < 1$ , ни одна ошибка не будет обнаружена. Кроме того, путь ABD вообще не будет охвачен тестом, и если в нем есть ошибка, она также не будет обнаружена. В таблице ожидаемый результат определяется по блок-схеме а, а фактический — по б.

Как видно из этой таблицы, ни одна из внесенных в алгоритм ошибок не будет обнаружена.

Таблица. Результат тестирования методом покрытия операторов

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
$A = 2, B = 0, X = 3$	$X = 2,5$	$X = 2,5$	Неуспешно



### Пример метода покрытия решений

Для программы, приведенной на рисунке выше, покрытие решений может быть выполнено двумя тестами, покрывающими пути  $\{ACE, ABD\}$ , либо  $\{ACD, ABE\}$ . Для этого выберем следующие исходные данные;  $\{A = 3, B = 0, X = 3\}$  — в первом случае и  $\{A = 2, B = 1, X = 1\}$  — во втором. Однако путь, где  $X$  не меняется, будет проверен с вероятностью 50 %: если во втором условии вместо условия  $X > 1$  записано  $X < 1$ , то ошибка не будет обнаружена двумя тестами.

Результаты тестирования приведены в таблице.

Таблица. Результат тестирования методом покрытия решений

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
$A = 3, B = 0, X = 3$	$X = 1$	$X = 1$	Неуспешно
$A = 2, B = 1, X = 1$	$X = 2$	* II	Успешно

### Пример метода покрытия условий

В рассматриваемом примере имеем условия:  $\{A > 1, B = 0\}$ ,  $\{A = 2, X > 1\}$ . Следовательно, требуется достаточное число тестов, такое, чтобы реализовать ситуации, где  $A > 1, A < 1, B = 0$  и  $B < 0$  в точке а и  $A = 2, A < 2, X > 1$  и  $X < 1$  в точке б. Тесты, удовлетворяющие критерию покрытия условий, и соответствующие им пути:

а)  $A = 2, B = 0, X = 4$  ACE;

б)  $A = 1, B = 1, X = 0$  ABD.

Таблица. Результаты тестирования методом покрытия условий

Тест	Ожидаемый	Фактический	Результат
	результат	результат	тестирования
$A = 2, B = 0, X = 4$	$X = 3$	$X = 3$	Неуспешно
$A = 1, B = 1, X = 0$	$X = 0$	$X = 1$	Успешно

### Метод покрытия решений/условий

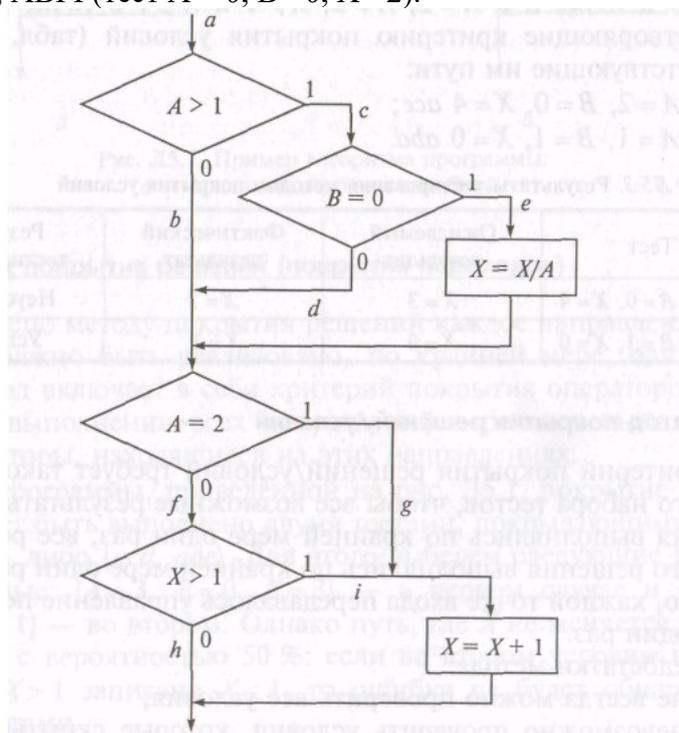
Так, в рассматриваемом примере два теста метода покрытия условий

а)  $A = 2, B = 0, X = 4$  ACEG;

б)  $A = 1, B = 1, X = 0$  ABD

отвечают и критерию покрытия решений/условий. Это является следствием того, что одни условия приведенных решений скрывают другие условия в этих решениях. Так, если условие  $A > 1$  будет ложным, транслятор может не проверять условия  $B = 0$ , поскольку при любом результате условия  $B = 0$  результат решения  $((A > 1) \& (B = 0))$  примет значение ложь. То есть в варианте на рисунке не все результаты всех условий выполняются в процессе тестирования.

Рассмотрим реализацию того же примера. Наиболее полное покрытие тестами в этом случае осуществляется так, чтобы выполнялись все возможные результаты каждого простого решения. Для этого нужно покрыть пути ACEG (тест  $A = 2, B = 0, X = 4$ ), ACDFH (тест  $A = 3, B = 1, X = 0$ ), ABFH (тест  $A = 0, B = 0, X = 0$ ), ABFI (тест  $A = 0, B = 0, X = 2$ ).



Протестировав алгоритм на рисунке выше, нетрудно убедиться в том, что критерии покрытия условий и критерии покрытия решений/условий недостаточно чувствительны к ошибкам в логических выражениях.

### Метод комбинаторного покрытия условий

В рассматриваемом примере должны быть покрыты тестами следующие восемь комбинаций:

1.  $A > 1, B = 0$ .
2.  $A > 1, B < 0$ .
3.  $A < 1, B = 0$ .
4.  $A < 1, B < 0$ .
5.  $A = 2, X > 1$ .
6.  $A = 2, X < 1$ .

7.  $A < 2, X > 1$ .

8.  $A < 2, X < 1$ .

Для того чтобы протестировать эти комбинации, необязательно использовать все 8 тестов. Фактически они могут быть покрыты четырьмя тестами

- $A = 2, B = 0, X = 4$  {покрывает 1, 5};
- $A = 2, B = 1, X = 1$  {покрывает 2, 6};
- $A = 0,5, B = 0, X = 2$  {покрывает 3, 7};
- $A = 1, B = 0, X = 1$  {покрывает 4, 8}.

Таблица. Результаты тестирования методом комбинаторного покрытия условий

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
$A=2, B=0, X=4$	$*=3$	$*=3$	Неуспешно
$A = 2, B = 1, X = 1$	$X = 2$	1 Г) СЧ II	Успешно
$A = 0,5, B = 0, X = 2$	$X = 3$	$X = 4$	Успешно
$A = 1, B = 0, X = 1$	$X = 1$	$X = 1$	Неуспешно

### Задание самостоятельной работы

В соответствии с индивидуальным вариантом, выбрать алгоритм для тестирования, обозначить буквами или цифрами ветви этих алгоритмов. Выписать пути алгоритма, которые должны быть проверены тестами для выбранного метода тестирования. Записать тесты, которые позволят пройти по путям алгоритма. Протестировать разработанную вами программу. Результаты оформить в виде таблицы:

Тест	Ожидаемый результат	Фактический результат	Результат тестирования
...	...	...	...

Перечень индивидуальных вариантов приведен в приложении В.

### Контрольные вопросы

1. Что такое тестирование по принципу «белого ящика»?
2. Назовите критерии покрытия кода.
3. Что такое метод покрытия операторов?
4. Что такое метод покрытия решений (покрытия переходов)?
5. Что такое метод покрытия условий?
6. Что такое метод покрытия решений/условий?

**Форма отчета:** тетрадь, защита работы.

## Практическое занятие № 21-

**Тема:** Оценка программных средств с помощью метрик

**Цель:** знакомство с ГОСТ 28195-89 «Оценка качества программных средств. Общие положения»; определить способы получения информации о ПС; формирование информационно-правовых компетенции обучающихся.

**Оборудование:** тетрадь, ручка, ПК

**Методические указания:** ознакомьтесь с теоретической частью, выполните задания, ответьте на контрольные вопросы

## Ход выполнения:

### Теоретические сведения

Необходимая документация: ГОСТ 28.195-89

Одной из важнейших проблем обеспечения качества программных средств является формализация характеристик качества и методология их оценки. Для определения адекватности качества функционирования, наличия технических возможностей программных средств к взаимодействию, совершенствованию и развитию необходимо использовать стандарты в области оценки характеристик их качества.

Показатели качества программного обеспечения устанавливают ГОСТ 28.195-89 «Оценка качества программных средств. Общие положения» и ГОСТ Р ИСО/МЭК 9126 «Информационная технология. Оценка программной продукции. Характеристика качества и руководства по их применению». Одновременное существование двух действующих стандартов, нормирующих одни и те же показатели, ставит вопрос об их гармонизации. Ниже рассмотрим каждый из перечисленных стандартов.

ГОСТ 28.195-89 «Оценка качества программных средств. Общие положения» устанавливает общие положения по оценке качества программных средств, номенклатуру и применяемость показателей качества.

Оценка качества ПС представляет собой совокупность операций, включающих выбор номенклатуры показателей качества оцениваемого ПС, определение значений этих показателей и сравнение их с базовыми значениями.

Методы определения показателей качества ПС различаются: по способам получения информации о ПС – измерительный, регистрационный, органолептический, расчетный; по источникам получения информации – экспертный, социологический.

Измерительный метод основан на получении информации о свойствах и характеристиках ПС с использованием инструментальных средств. Например, с использованием этого метода определяется объем ПС - число строк исходного текста программ и число строк - комментариев, число операторов и операндов, число исполненных операторов, число ветвей в программе, число точек входа (выхода), время выполнения ветви программы, время реакции и другие показатели.

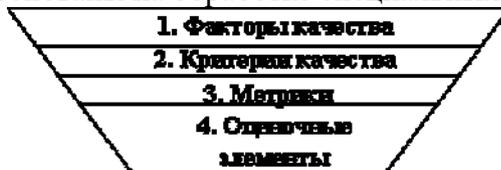
Регистрационный метод основан на получении информации во время испытаний или функционирования ПС, когда регистрируются и подсчитываются определенные события, например, время и число сбоев и отказов, время передачи управления другим модулям, время начала и окончания работы.

Органолептический метод основан на использовании информации, получаемой в результате анализа восприятия органов чувств (зрения, слуха), и применяется для определения таких показателей как удобство применения, эффективность и т. п.

Расчетный метод основан на использовании теоретических и эмпирических зависимостей (на ранних этапах разработки), статистических данных, накапливаемых при испытаниях, эксплуатации и сопровождении ПС. При помощи расчетного метода определяются длительность и точность вычислений, время реакции, необходимые ресурсы.

Определение значений показателей качества ПС экспертным методом осуществляется группой экспертов-специалистов, компетентных в решении данной задачи, на базе их опыта и интуиции. Экспертный метод применяется в случаях, когда задача не может быть решена никаким другим из существующих способов или другие способы являются значительно более трудоемкими. Экспертный метод рекомендуется применять при определении показателей наглядности, полноты и доступности программной документации, легкости освоения, структурности.

Социологические методы основаны на обработке специальных анкет-вопросников.



Показатели качества объединены в систему из четырех уровней. Каждый вышестоящий уровень содержит в качестве составляющих показатели нижестоящих уровней (рисунок 1).

Стандарт ИСО 9126 (ГОСТ Р ИСО/МЭК 9126) «Информационная технология. Оценка программной продукции. Характеристика качества и руководства по их применению».

Определенные настоящим стандартом характеристики дополнены рядом требований по выбору метрик и их измерению для различных проектов ПС. Они применимы к любому типу ПС, включая компьютерные программы и данные, содержащиеся в программируемом оборудовании. Эти характеристики обеспечивают согласованную терминологию для анализа качества ПС. Кроме того, они определяют схему для выбора и специфицирования требований к качеству ПС, а также для сопоставления возможностей различных программных продуктов, таких как функциональные возможности, надежность, практичность и эффективность.

Все множество атрибутов качества ПС может быть классифицировано в структуру иерархического дерева характеристик и субхарактеристик. Самый высший уровень этой структуры состоит из характеристик качества, а самый нижний уровень – из их атрибутов. Эта иерархия не строгая, поскольку некоторые атрибуты могут быть связаны с более чем одной субхарактеристикой. Таким же образом, внешние свойства (такие, как пригодность, корректность, устойчивость к ошибкам или временная эффективность) влияют на наблюдаемое качество. Недостаток качества в использовании (например, пользователь не может закончить задачу) может быть прослежен к внешнему качеству (например, функциональная пригодность или простота использования) и связанным с ним внутренним атрибутам, которые необходимо изменить.

Внутренние метрики могут применяться в ходе проектирования и программирования к неисполняемым компонентам ПС (таким, как спецификация или исходный программный текст). При разработке ПС промежуточные продукты следует оценивать с использованием внутренних метрик, которые измеряют свойства программ, и могут быть выведены из моделируемого поведения. Основная цель внутренних метрик – обеспечивать, чтобы было достигнуто требуемое внешнее качество. Внутренние метрики дают возможность пользователям, испытателям и разработчикам оценивать качество ЖЦ программ и заниматься вопросами технологического обеспечения качества задолго до того, как ПС становится готовым исполняемым продуктом.

Внутренние метрики позволяют измерять внутренние атрибуты или формировать признаки внешних атрибутов путем анализа статических свойств промежуточных или поставляемых программных компонентов. Измерения внутренних метрик используют категории, числа или характеристики элементов из состава ПС, которые, например, имеются в процедурах исходного программного текста, в графе потока управления, в потоке данных и в представлениях изменения состояний памяти. Документация также может оцениваться с использованием внутренних метрик.

Внешние метрики используют меры ПС, выведенные из поведения системы, частью которых они являются, путем испытаний, эксплуатации или наблюдения исполняемого ПС или системы. Перед приобретением или использованием ПС его следует оценить с использованием метрик, основанных на деловых и профессиональных целях, связанных с использованием, эксплуатацией и управлением продуктом в определенной организационной и технической среде. Внешние метрики обеспечивают заказчикам, пользователям, испытателям и разработчикам возможность определять качество ПС в ходе испытаний или эксплуатации.

Когда требования к качеству ПС определены, в них должны быть перечислены характеристики и субхарактеристики, которые составляют полный набор показателей качества. Затем определяются подходящие внешние метрики и их приемлемые диапазоны значений, устанавливающие количественные и качественные критерии, которые подтверждают, что ПС удовлетворяет потребностям заказчика и пользователя. Далее определяются и специфицируются внутренние атрибуты качества, чтобы спланировать удовлетворение требуемых внешних характеристик качества в конечном продукте и обеспечивать их в промежуточных продуктах в ходе разработки. Подходящие внутренние метрики и приемлемые диапазоны специфицируются для получения числовых значений или категорий внутренних характеристик качества, чтобы их можно было использовать для проверки того, что промежуточные продукты в процессе разработки удовлетворяют внутренним спецификациям качества. Рекомендуется использовать внутренние метрики, которые имеют наиболее сильные связи с целевыми внешними метриками, чтобы они могли помогать при прогнозировании значений внешних метрик.

Метрики качества в использовании измеряют, в какой степени продукт удовлетворяет потребности конкретных пользователей в достижении заданных целей с результативностью, продуктивностью и удовлетворением в заданном контексте использования. При этом результативность подразумевает точность и полноту достижения определенных целей пользователями при применении ПС; продуктивность соответствует соотношению

израсходованных ресурсов и результативности при эксплуатации ПС, а удовлетворенность – психологическое отношение к качеству использования продукта. Эта метрика не входит в число шести базовых характеристик ПС, регламентируемых стандартом ИСО 9126, однако рекомендуется для интегральной оценки результатов функционирования комплексов программ.

Оценивание качества в использовании должно подтверждать его для определенных сценариев и задач, оно составляет полный объединенный эффект характеристик качества ПС для пользователя. Качество в использовании – это восприятие пользователем качества системы, содержащей ПС, и оно измеряется скорее в терминах результатов использования комплекса программ, чем собственных внутренних свойств ПС. Связь качества в использовании с другими характеристиками качества ПС зависит от типа пользователя, так, например, для конечного пользователя качество в использовании обуславливают, в основном, характеристики функциональных возможностей, надежности, практичности и эффективности, а для персонала сопровождения ПС качество в использовании определяет сопровождаемость. На качество в использовании могут влиять любые характеристики качества, и это понятие шире, чем практичность, которая связана с простотой использования и привлекательностью. Качество в использовании, в той или иной степени, характеризуется сложностью применения комплекса программ, которую можно описать трудоемкостью использования с требуемой результативностью. Многие характеристики и субхарактеристики ПС обобщенно отражаются неявными технико-экономическими показателями, которые поддерживают функциональную пригодность конкретного ПС. Однако их измерение и оценка влияния на показатели качества, представляет сложную проблему.

#### Задание практической работы

**Задание №1.** Провести сравнение понятий «качество» в государственных и международных стандартах. Выписать документы, в которых даны данные определения.

**Задание №2.** Описать методы получения информации о ПС по ГОСТу. Для каждого метода выделите источник информации.

**Задание №3.** Выбрать стандарты для оценки качества ПС. Перечислите критерии надежности ПС по ГОСТу.

**Задание №4.** Методика оценки качественных показателей ПП основана на составлении метрики ПП. В работе необходимо выполнить следующее:

1. Выбрать показатели качества (не менее 5) и сформулировать их сущность. Каждый показатель должен быть существенным, т.е. должны быть ясны потенциальные выгоды его использования. Показатели представить в виде таблицы.

Показатели качества	Сущность показателя	Экспертная оценка (вес) $w_i$	Оценка, установленная экспериментом $r_i$
---------------------	---------------------	-------------------------------	---

2. Установить веса показателей  $w_i$  ( $\sum w_i = 1$ );

3. Для каждого показателя установить конкретную численную оценку  $r_i$  от 0 до 1, исходя из следующего:

0 – свойство в ПП присутствует, но качество его неприемлемо;

0.5 - 1 – свойство в ПП присутствует и обладает приемлемым качеством;

1 – свойство в ПП присутствует и обладает очень высоким качеством.

Возможно, присвоение промежуточных значений в соответствии с мнением оценивающего лица относительно полезности того или иного свойства ПП.

$$K = \frac{\sum w_i \cdot r_i}{\text{общее количество показателей}}$$

Разработать приложение калькулятор на любом известном языке программирования. Провести его сравнение со стандартным калькулятором Microsoft по следующим оценочным элементам:

1 Надежность ПО

Характеризует способность ПО в конкретных областях применения выполнять заданные функции в соответствии с программными документами в условиях возникновения отклонений в среде функционирования, вызванных сбоями технических средств, ошибками во входных данных, ошибками обслуживания и другими дестабилизирующими воздействиями.

Оценочные элементы фактора «Надежность ИС»:

Код элемента	Наименование	Метод оценки	Оценка калькулятора Microsoft	Оценка калькулятора
--------------	--------------	--------------	-------------------------------	---------------------

H0101	Наличие требований к программе по устойчивости функционирования при наличии ошибок во входных данных	экспертный		
H0102	Возможность обработки ошибочных ситуаций	экспертный		
H0103	Полнота обработки ошибочных ситуаций	экспертный		
H0104	Наличие тестов для проверки допустимых значений входных данных	экспертный		
H0105	Наличие системы контроля полноты входных данных	экспертный		
H0106	Наличие средств контроля корректности входных данных	экспертный		
H0201	Наличие требований к программе по восстановлению процесса выполнения в случае сбоя операционной системы, процессора внешних устройств	экспертный		
H0202	Наличие требований к программе по восстановлению результатов при отказах процессора и операционной системы	экспертный		
H0203	Наличие средств восстановления процессора в случае сбоев оборудования	экспертный		
H0205	Наличие возможности повторного старта с точки прерывания	экспертный		
H0110	Наличие обработки неопределенностей	экспертный		
H0301	Наличие централизованного управления процессами, конкурирующими из-за ресурсов	экспертный		
H0302	Наличие возможности автоматически обходить ошибочные ситуации в процессе вычисления	экспертный		
Всего				

## 2 Сопровождаемость

Характеризует технологические аспекты, обеспечивающие простоту устранения ошибок в ПО и программных документах и поддержания ПО в актуальном состоянии.

### Оценочные элементы фактора «сопровождаемость»

Код элемента	Наименование	Метод оценки	Оценка калькулятора Microsoft	Оценка калькулятора
C0803	Наличие комментариев в точках входа и выхода программы	экспертный		
C0303	Осуществляется ли передача результатов работы модуля через вызывающий его модуль	экспертный		
C0604	Оценка программы по числу циклов	экспертный		
C1001	Используется ли язык высокого уровня	экспертный		
C0301	Наличие проверки корректности передаваемых данных	экспертный		
C0601	Использование при построении программ метода структурного программирования	экспертный		

C0602	Соблюдение принципа разработки программы сверху вниз	экспертный		
C0201	Наличие ограничений на размеры модуля	экспертный		
C0101	Наличие модульной схемы программы	экспертный		
Всего				

### 3 Корректность

Характеризует степень соответствия ПО требованиям, установленным в техническом задании, требованиям к обработке данных и общесистемным требованиям.

#### Оценочные элементы фактора «корректность»

Код элемента	Наименование	Метод оценки	Оценка калькулятора Microsoft	Оценка калькулятора
K0101	Наличие всех необходимых документов для понимания и использования ПС	экспертный		
K0102	Наличие описания и схемы иерархии модулей программы	экспертный		
K0103	Наличие описания основных функций	экспертный		
K0104	Наличие описания частных функций	экспертный		
K0105	Наличие описания данных	экспертный		
K0106	Наличие описания алгоритмов	экспертный		
K0107	Наличие описания интерфейсов между модулями	экспертный		
K0111	Наличие описания всех параметров	экспертный		
K0112	Наличие описания методов настройки системы	экспертный		
K0114	Наличие описания способов проверки работоспособности программы	экспертный		
K0201	Реализация всех исходных модулей	экспертный		
K0202	Реализация всех основных функций	экспертный		
K0203	Реализация всех частных алгоритмов	экспертный		
K0204	Реализация всех алгоритмов	экспертный		
K0209	Наличие определения всех данных; переменные, индексы, массивы и пр.	экспертный		
K0210	Наличие интерфейсов с пользователем	экспертный		
K401	Отсутствие противоречий в выполнении основных функций			
K402	Отсутствие противоречий в выполнении частных функций	экспертный		
K0403	Отсутствие противоречий в выполнении алгоритмов	экспертный		
K0404	Правильность взаимосвязей	экспертный		
K0406	Правильность реализации интерфейса с пользователем	экспертный		
K0407	Отсутствие противоречий в настройке системы	экспертный		

K0701	Комплектность документации в соответствии со стандартами	экспертный		
Всего				

#### 4 Универсальность/гибкость.

Характеризует адаптируемость ПО к новым функциональным требованиям, возникающим вследствие изменения области применения или других условий функционирования;

#### Оценочные элементы фактора «гибкость»

Код элемента	Наименование	Метод оценки	Оценка калькулятора Microsoft	Оценка калькулятора
Г1208	Наличие общих комментариев к программам	экспертный		
Г1301	Использование языков высокого уровня	экспертный		
Г1302	Семантика имен используемых переменных	экспертный		
Г1303	Использование отступов, сдвигов и пропусков при формировании текста	экспертный		
Г0803	Зависимость от других программных средств	экспертный		
Г0101	Оценка числа потенциальных пользователей	экспертный		
Г0201	Наличие схемы иерархии модулей программы	экспертный		
Г0202	Оценка независимости модулей	экспертный		
Г0402	Наличие описания структуры программ	экспертный		
Г0802	Оценка зависимости программы от программ операционной системы	экспертный		
Всего				

#### 5 Оценка качества

Проведём расчёт по формуле, которая дана задании 4, для оценивания качества каждого оценочного элемента, рассмотренных ранее для калькулятора Microsoft и разработанного нами калькулятора.

$$K = \frac{\sum w_i \cdot r_i}{\text{общее количество показателей}}$$

Формула:

Оценки качества по следующим оценочным элементам: надежность программного средства (ПС), сопровождаемость, корректность, гибкость:

Калькулятор Microsoft	Разработанный калькулятор	
K <sub>НМ</sub> =	K <sub>НР</sub> =	
K <sub>СМ</sub> =	K <sub>СР</sub> =	
K <sub>КМ</sub> =	K <sub>КР</sub> =	

К <sub>ГМ</sub> =	К <sub>ГР</sub> =	
К <sub>М</sub> =	К <sub>Р</sub> =	

### Вывод

Сделать вывод какой из калькуляторов является более удачным.

Результатом выполнения данной работы является отчет

### Контрольные вопросы

В каком ГОСТе приведены метрики качество

**Форма отчета:** тетрадь, защита работы.

## Практическое занятие № 22

**Тема:** Инспекция программного кода на предмет соответствия стандартам кодирования

**Цель:** научиться выполнять реорганизацию программного кода на основании стандартов кодирования

**Оборудование:** тетрадь, ручка, ПК

**Методические указания:** ознакомьтесь с теоретической частью, выполните задания, ответьте на контрольные вопросы

**Ход выполнения:**

### Теоретические сведения

Инспекция в целом

■ Инспекция - (от лат. inspectio - осмотр) - орган, осуществляющий контроль за соблюдением установленных государством правил.

■ Инспекция - орган управления, призванный следить за выполнением установленных правил и совмещающий контрольные функции с определенными административными правами. В их задачи входит также принятие на месте мер к исправлению недостатков.



■ Программный код программы - это текст, набор команд, выполненный на особом языке программирования, понятном машине.

■ Код программы необходим в первую очередь для написания и редактирования его человеком. Код программы также называют исходным кодом или исходным текстом программы.

■ Стандарт кодирования — набор правил и соглашений, которые описывают базовые принципы оформления программного кода, используемого совместно группой разработчиков.

■ Цель использования стандарта — упрощение восприятия программного кода человеком, сокращение нагрузки на память и зрение при чтении программы.

**Некоторые из стандартов кодирования приведены ниже:**

**1 Ограниченное использование глобалов:**

Эти правила говорят о том, какие типы данных могут быть объявлены глобальными, а какие нет.

## **2 Стандартные заголовки для разных модулей:**

Для лучшего понимания и обслуживания кода заголовков различных модулей должен соответствовать стандартному формату и информации. Формат заголовка должен содержать ниже вещи, которые используются в различных компаниях:

1. Наименование модуля
2. Дата создания модуля
3. Автор модуля
4. История изменений
5. Краткое описание модуля о том, что делает модуль
6. В модуле поддерживаются различные функции, а также их входные и выходные параметры
7. Глобальные переменные, доступные или измененные модулем

## **3 Соглашения об именах для локальных переменных, глобальных переменных, констант и функций:**

Некоторые из соглашений об именах приведены ниже:

1. Содержательное и понятное название переменных помогает любому понять причину его использования.
2. Локальные переменные должны быть названы с использованием букв верблюда, начинающихся с маленькой буквы (например, `localData`), тогда как имена глобальных переменных должны начинаться с заглавной буквы (например, `GlobalData`). Имена констант должны быть сформированы только заглавными буквами (например, `CONSDATA`).
3. Лучше избегать использования цифр в именах переменных.
4. Названия функции должны быть написаны в верблюжьем регистре, начиная с маленьких букв.
5. Название функции должно четко и кратко описывать причину ее использования.

## **4 Отступ:**

Правильный отступ очень важен для улучшения читабельности кода. Чтобы сделать код читабельным, программисты должны правильно использовать пробелы. Некоторые из интервалов даны ниже:

1. После запятой между аргументами функции должен быть пробел.
2. Каждый вложенный блок должен иметь правильные отступы и интервалы.
3. Надлежащий отступ должен быть в начале и в конце каждого кадра в программе.
4. Все фигурные скобки должны начинаться с новой строки, а код, следующий за окончанием фигурных скобок, также начинается с новой строки.

## **5 Возвращаемые значения ошибок и соглашения об обработке исключений:**

Все функции, которые сталкиваются с ошибкой, должны возвращать 0 или 1 для упрощения отладки.

С другой стороны, руководящие принципы кодирования дают некоторые общие рекомендации относительно стиля кодирования, которому необходимо следовать для улучшения понятности и читабельности кода. Некоторые из руководств по кодированию приведены ниже:

## **6 Избегайте использования стиля кодирования, который слишком сложен для понимания:**

Код должен быть легко понятным. Сложный код делает обслуживание и отладку трудной и дорогой.

## **7 Избегайте использования идентификатора для нескольких целей:**

Каждой переменной должно быть дано описательное и осмысленное имя, указывающее причину ее использования. Это невозможно, если идентификатор используется для нескольких целей, что может привести к путанице у читателя. Более того, это приводит к большим трудностям при будущих улучшениях.

## **8 Код должен быть хорошо документирован:**

Код должен быть правильно прокомментирован для понимания. Комментарии относительно утверждений повышают понятность кода.

## **9 Длина функций не должна быть очень большой:**

Длинные функции очень трудно понять. Вот почему функции должны быть достаточно маленькими, чтобы выполнять небольшую работу, а длинные функции должны быть разбиты на маленькие для выполнения небольших задач.

### **10 Старайтесь не использовать оператор GOTO:**

Оператор GOTO делает программу неструктурированной, что снижает ее понятность и затрудняет отладку.

### **Цель инспекции программного кода**

Обнаружение и исправление ошибок, которые были пропущены, остались незамеченными при разработке. Результат инспекции как правило – улучшение качество ПО и навыки разработчика.

### **Сервисы инспекции программного кода**

#### 1 Reviewable

Это новый на рынке инструмент инспекции кода, он помогает повысить качество кода с помощью синтаксического выделения, находит баги/ проблемы, кастомизирует шрифт кода и многое другое.

#### 2 RhodeCode

Ещё один отличный инструмент для инспекции кода и поиска ошибок и проблем в коде.

#### 3 CodeStriker

Бесплатное веб-приложение с открытым исходным кодом, призванное помочь разработчику инспектировать код в вебе.

#### 4 Code Brag

#### 5 Phabricator

Open source ПО и веб-приложение, включающее проверку кода, хостинг GIT/Hg/SVN, поиск ошибок, аудит исходного кода и т. д.

#### 6 Codifferous

#### 7 Getbarkeep

#### 8 Crucible

#### 9 Code Review Tool

#### 10 Malevich

#### 11 SmartBear

#### 12 Veracode

#### 13 Gerrit

Веб-приложение инспекции кода облегчает онлайн-ревью для проектов, использующих распределённую систему управления версиями Git.

#### 14 Review Assistant

#### 15 Review Board

#### 16 Peer Review Plugin

#### 17 Codereview

#### 18 Code Reviewer

#### 19 Code Analysis Tool

#### 20 jArchitect

### **Задание практической работы**

#### **Методика выполнения**

Переписать программный код, используя общепринятые соглашения и рекомендации по именованию и форматированию переменных, операторов, выражений.

#### **Требования:**

- добавить комментарии в программный код;
- проверить правильность именования переменных, констант, методов;
- проверить правильность объявления переменных и констант.

#### **Было**

Класс Main

пакетная игра;

импорт игры. Персонажи. \*;

импорт игры. Персонажи. Персонажи; импорт игры. Энергетика. Энергетика; импорт игры. Энергетика. Освещение; импорт игры. Уровни. Блок;

```

импорт игры. Уровни. Уровень; импортировать game.Levels.Level_data; импорт игры.
Weapon.Bullet;
импорт игры. Оружие. Оружие;
импорт javafx.animation.AnimationTimer; импорт javafx.application.Application; импорт
javafx.scene.Scene;
импорт javafx.scene.image.Image; импорт javafx.scene.input.KeyCode; импорт
javafx.scene.layout.Pane; импорт javafx.stage.Stage;
импорт java.io.DataInputStream; импорт java.io.FileInputStream; импорт java.io.IOException; импорт
java.util.ArrayList; импорт java.util.HashMap;
публичный класс Main расширяет приложение {
    public static ArrayList <Block> blocks = new ArrayList <> (); public static ArrayList <Bullet> bullets
= new ArrayList <> ();
    public static ArrayList <Bullet> врагаBullets = новый ArrayList <> (); public static ArrayList
<EnemyBase> враги = новый ArrayList <> (); static HashMap <KeyCode, Boolean> keys = new
HashMap <> (); публичная статическая сцена этапа;
    публичная статическая сцена;
    public static Pane gameRoot = new Pane (); public static Pane appRoot = new Pane (); публичное
статическое меню;
    публичный статический Персонаж букера; публичная статическая HUD HUD; статическое
оружие общего назначения; публичная статика елизавета елизавета; статический VendingMachine
vendingMachine; статическое учебное пособие;
    частные статические CutScenes cutScene; публичная статика Энергетика энергетика; публичная
статическая молния молнии; public static int levelNumber;
    уровень статического уровня;
    public static AnimationTimer timer = new AnimationTimer () { @Override
    public void handle (давно) { Обновить();
    }
    };
    приватное статическое void update () { для (EnemyBase враг: враги) { enemy.update ();
    if (врага.delete () ) { enemies.remove (враг); переменная;
    }
    }
    Bullet.update (); Controller.update (); booker.update ();
    if (! energetic.getName (). equals ("")) energetic.update ();
    if (levelNumber > 0) elizabeth.update ();
    если (молния! = ноль) { lightning.update ();
    if (lightning.delete () ) молния = ноль;
    }
    menu.update (); hud.update (); weapon.update ();
    if (booker.getTranslateX () > Level_data.BLOCK_SIZE * 295) cutScene = новые CutScenes
(levelNumber);
    }
    @Override
    public void start (Stage primaryStage) выдает исключение { stage = primaryStage;
    сцена = новая сцена (appRoot, 1280, 720);
    . AppRoot.getChildren () добавить (gameRoot); уровень = новый уровень ();
    try (DataInputStream dataInputStream = new DataInputStream (новый FileInputStream ("C:
/DeadShock/saves/data.dat"))) {
    levelNumber = dataInputStream.readInt (); level.createLevels (levelNumber); level.changeImageView
(levelNumber); vendingMachine = new VendingMachine (); букер = новый персонаж ();
    booker.setMoney (dataInputStream.readInt ()); booker.setSalt (dataInputStream.readByte ());
    booker.setCountLives (2);
    оружие = новое оружие ();
    weapon.setWeaponClip (dataInputStream.readInt ()); weapon.setBullets (dataInputStream.readInt ());
    hud = новый HUD ();

```

```
Елизавета = новая Елизавета (); энергичный = новый Энергетический ();  
} catch (IOException e) { levelNumber = 0; level.createLevels (levelNumber);  
vendingMachine = new VendingMachine (); букер = новый персонаж ();  
оружие = новое оружие (); hud = новый HUD ();  
энергичный = новый Энергетический (); tutorial = new Tutorial (levelNumber);  
}
```

```
switch (levelNumber) { случай 0:
```

```
    враги.аdd (новый EnemyComstock (Level_data.BLOCK_SIZE * 117, 200)); враги.аdd (новый  
EnemyComstock (Level_data.BLOCK_SIZE * 127, 200)); враги.аdd (новый EnemyComstock  
(Level_data.BLOCK_SIZE * 148, 200)); враги.аdd (новый EnemyComstock (Level_data.BLOCK_SIZE  
* 161, 200)); враги.аdd (новый EnemyComstock (Level_data.BLOCK_SIZE * 171, 200)); враги.аdd  
(новый EnemyComstock (Level_data.BLOCK_SIZE * 185, 200)); враги.аdd (новый EnemyComstock  
(Level_data.BLOCK_SIZE * 204, 200)); враги.аdd (новый EnemyComstock (Level_data.BLOCK_SIZE  
* 215, 200)); враги.аdd (новый EnemyComstock (Level_data.BLOCK_SIZE * 228, 200)); враги.аdd  
(новый EnemyComstock (Level_data.BLOCK_SIZE * 233, 200)); враги.аdd (новый EnemyComstock  
(Level_data.BLOCK_SIZE * 243, 200)); враги.аdd (новый EnemyComstock (Level_data.BLOCK_SIZE  
* 252, 200)); враги.аdd (новый EnemyComstock (Level_data.BLOCK_SIZE * 262, 200)); враги.аdd  
(новый EnemyComstock (Level_data.BLOCK_SIZE * 277, 200)); враги.аdd (новый EnemyComstock  
(Level_data.BLOCK_SIZE * 280, 200)); враги.аdd (новый EnemyComstock (Level_data.BLOCK_SIZE  
* 286, 200)); переменна;
```

```
    Дело 1:
```

```
    враги.аdd (новый EnemyRedEye (Level_data.BLOCK_SIZE * 57, 200)); враги.аdd (новый  
EnemyRedEye (Level_data.BLOCK_SIZE * 67, 200)); враги.аdd (новый EnemyRedEye  
(Level_data.BLOCK_SIZE * 74, 200)); враги.аdd (новый EnemyComstock (Level_data.BLOCK_SIZE *  
87, 200)); враги.аdd (новый EnemyRedEye (Level_data.BLOCK_SIZE * 104, 150)); враги.аdd (новый  
EnemyComstock (Level_data.BLOCK_SIZE * 117, 200)); враги.аdd (новый EnemyRedEye  
(Level_data.BLOCK_SIZE * 133, 200)); враги.аdd (новый EnemyRedEye (Level_data.BLOCK_SIZE *  
156, 200)); враги.аdd (новый EnemyComstock (Level_data.BLOCK_SIZE * 177, 200)); враги.аdd  
(новый EnemyComstock (Level_data.BLOCK_SIZE * 193, 200)); враги.аdd (новый EnemyRedEye  
(Level_data.BLOCK_SIZE * 201, 200)); враги.аdd (новый EnemyRedEye (Level_data.BLOCK_SIZE *  
216, 200)); враги.аdd (новый EnemyRedEye (Level_data.BLOCK_SIZE * 224, 200)); враги.аdd (новый  
EnemyComstock (Level_data.BLOCK_SIZE * 246, 200)); враги.аdd (новый EnemyRedEye  
(Level_data.BLOCK_SIZE * 260, 200)); враги.аdd (новый EnemyComstock (Level_data.BLOCK_SIZE  
* 277, 100));
```

```
    Level_data.enemyBlocks.add (новый блок («невидимый», Level_data.BLOCK_SIZE * 34,  
Level_data.BLOCK_SIZE * 13));
```

```
    Level_data.enemyBlocks.add (новый блок («невидимый», Level_data.BLOCK_SIZE * 36,  
Level_data.BLOCK_SIZE * 13));
```

```
    Level_data.enemyBlocks.add (новый блок («невидимый», Level_data.BLOCK_SIZE * 60,  
Level_data.BLOCK_SIZE * 9));
```

```
    Level_data.enemyBlocks.add (новый блок («невидимый», Level_data.BLOCK_SIZE * 61,  
Level_data.BLOCK_SIZE * 9));
```

```
    Level_data.enemyBlocks.add (новый блок («невидимый», Level_data.BLOCK_SIZE * 106,  
Level_data.BLOCK_SIZE * 7));
```

```
    Level_data.enemyBlocks.add (новый блок («невидимый», Level_data.BLOCK_SIZE * 107,  
Level_data.BLOCK_SIZE * 7));
```

```
    Level_data.enemyBlocks.add (новый блок («невидимый», Level_data.BLOCK_SIZE * 168,  
Level_data.BLOCK_SIZE * 13));
```

```
    Level_data.enemyBlocks.add (новый блок («невидимый», Level_data.BLOCK_SIZE * 170,  
Level_data.BLOCK_SIZE * 13));
```

```
    Level_data.enemyBlocks.add (новый блок («невидимый», Level_data.BLOCK_SIZE * 196,  
Level_data.BLOCK_SIZE * 13));
```

```
    Level_data.enemyBlocks.add (новый блок («невидимый», Level_data.BLOCK_SIZE * 197,  
Level_data.BLOCK_SIZE * 13));
```

```

    Level_data.enemyBlocks.add (новый блок («невидимый», Level_data.BLOCK_SIZE * 232,
Level_data.BLOCK_SIZE * 8));
    Level_data.enemyBlocks.add (новый блок («невидимый», Level_data.BLOCK_SIZE * 233,
Level_data.BLOCK_SIZE * 8));
    переменна;
    }
    меню = новое меню ();
    appRoot.getChildren () добавить (menu.menuBox).
    booker.translateXProperty (). addListener (((наблюдаемый, oldValue, newValue) -> { int offset =
newValue.intValue ();
    if (offset > 600 && offset < gameRoot.getWidth () - 680) { gameRoot.setLayoutX (- (смещение - 600));
level.setBackground (). setLayoutX ((смещение - 600) / 1,5);
    }
    если (смещение <= 100) level.setBackground () setLayoutX (0).
    }));
    vendingMachine.createButtons ();
    stage.getIcons (). add (новое изображение ("файл: / C: / DeadShock/images/icon.jpg")); stage.setTitle
("DeadShock");
    stage.setResizable (ложь); stage.setWidth (scene.getWidth ()); stage.setHeight (scene.getHeight ());
stage.setScene (сцены);
    stage.show (); timer.start ();
    }
    public static void main (String [] args) { запуск (arg);
    }
    }

```

### Стало

Класс Main

пакетная игра;

```

импорт игры. Персонажи. *;
импорт игры. Персонажи. Персонажи; импорт игры. Энергетика.
Энергетика; импорт игры. Энергетика. Освещение; импорт игры. Уровни.
Блок;
импорт игры. Уровни. Уровень; импортировать game.Levels.Level_data;
импорт игры. Weapon.Bullet;
импорт игры. Оружие. Оружие;

```

```

import javafx.animation.AnimationTimer;

```

```

import javafx.application.Application;

```

```

import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.input.KeyCode;
import javafx.scene.layout.Pane;
import javafx.stage.Stage;
import java.io.DataInputStream;
import java.io.FileInputStream;

```

```

импорт java.io.IOException;
импорт java.util.ArrayList;
импорт java.util.HashMap;

```

```

публичный класс Main расширяет приложение {

```

```
public static ArrayList <Block> blocks = new ArrayList <> ();
public static ArrayList <Bullet> bullets = new ArrayList <> ();
public static ArrayList <Bullet> врагаBullets = новый ArrayList <> ();
public static ArrayList <EnemyBase> враги = новый ArrayList <> ();
static HashMap <KeyCode, Boolean> keys = new HashMap <> ();
```

```
публичная статическая сцена этапа;
публичная статическая сцена;
```

```
public static Pane gameRoot = new Pane ();
public static Pane appRoot = new Pane ();
```

```
публичное статическое меню;
публичный статический Персонаж букера;
публичная статическая HUD HUD;
статическое оружие общего назначения;
публичная статика елизавета елизавета;
статический VendingMachine vendingMachine;
статическое учебное пособие;
частные статические CutScenes cutScene;
публичная статика Энергетика энергетика;
публичная статическая молния молнии;
```

```
public static int levelNumber;
```

```
уровень статического уровня;
```

```
public static AnimationTimer timer = new AnimationTimer () {
    @Override
    public void handle (давно) {
        Обновить ();
    }
};
```

```
private void initContent () {
    . AppRoot.getChildren () добавить (gameRoot); уровень = новый
уровень ();
    try (DataInputStream dataInputStream = new DataInputStream
(новый FileInputStream ("C:/DeadShock/saves/data.dat"))) {
        levelNumber = dataInputStream.readInt ();
        level.createLevels (levelNumber);
        level.changeImageView (levelNumber);
        vendingMachine = new VendingMachine ();

        букер = новый персонаж ();

        booker.setMoney (dataInputStream.readInt ());
        booker.setSalt (dataInputStream.readByte ());
        booker.setCountLives (2);

        оружие = новое оружие ();
        weapon.setWeaponClip (dataInputStream.readInt ());
        weapon.setBullets (dataInputStream.readInt ());
        hud = новый HUD ();

        Елизавета = новая Елизавета ();
```

```

        энергичный = новый Энергетический ();
    } catch (IOException e) {
        levelNumber = 0;
        level.createLevels (levelNumber);
        vendingMachine = new VendingMachine ();

        букер = новый персонаж ();
        оружие = новое оружие ();

        hud = новый HUD ();

        энергичный = новый Энергетический ();

        tutorial = new Tutorial (levelNumber);
    }
    createEnemies ();

    меню = новое меню ();

    appRoot.getChildren () добавить (menu.menuBox).
    booker.translateXProperty (). addListener ((наблюдаемый,
    oldValue, newValue) -> {
        int offset = newValue.intValue ();
        if (offset > 600 && offset < gameRoot.getWidth () - 680) {
            gameRoot.setLayoutX (- (смещение - 600));
            level.getBackground (). setLayoutX ((смещение - 600) /
            1,5);
        }
        если (смещение <= 100) level.getBackground () setLayoutX
        (0).
    }));

    vendingMachine.createButtons ();
}

public static void createEnemies () {
    switch (levelNumber) {
        случай 0:
            враги.аdd (новый EnemyComstock
            (Level_data.BLOCK_SIZE * 117, 200));
            враги.аdd (новый EnemyComstock
            (Level_data.BLOCK_SIZE * 127, 200));
            враги.аdd (новый EnemyComstock
            (Level_data.BLOCK_SIZE * 148, 200));
            враги.аdd (новый EnemyComstock
            (Level_data.BLOCK_SIZE * 161, 200));
            враги.аdd (новый EnemyComstock
            (Level_data.BLOCK_SIZE * 171, 200));
            враги.аdd (новый EnemyComstock
            (Level_data.BLOCK_SIZE * 185, 200));
            враги.аdd (новый EnemyComstock
            (Level_data.BLOCK_SIZE * 204, 200));
            враги.аdd (новый EnemyComstock
            (Level_data.BLOCK_SIZE * 215, 200));
            враги.аdd (новый EnemyComstock
            (Level_data.BLOCK_SIZE * 228, 200));
    }
}

```

```
враги.аdd (новый EnemyComstock  
(Level_data.BLOCK_SIZE * 233, 200));  
враги.аdd (новый EnemyComstock  
(Level_data.BLOCK_SIZE * 243, 200));  
враги.аdd (новый EnemyComstock  
(Level_data.BLOCK_SIZE * 252, 200));  
враги.аdd (новый EnemyComstock  
(Level_data.BLOCK_SIZE * 262, 200));  
враги.аdd (новый EnemyComstock  
(Level_data.BLOCK_SIZE * 277, 200));  
враги.аdd (новый EnemyComstock  
(Level_data.BLOCK_SIZE * 280, 200));  
враги.аdd (новый EnemyComstock  
(Level_data.BLOCK_SIZE * 286, 200));  
перемена;
```

случай 1:

```
враги.аdd (новый EnemyRedEye (Level_data.BLOCK_SIZE  
* 57, 200));  
враги.аdd (новый EnemyRedEye (Level_data.BLOCK_SIZE  
* 67, 200));  
враги.аdd (новый EnemyRedEye (Level_data.BLOCK_SIZE  
* 74, 200));  
враги.аdd (новый EnemyComstock  
(Level_data.BLOCK_SIZE * 87, 200));  
враги.аdd (новый EnemyRedEye (Level_data.BLOCK_SIZE  
* 104, 150));  
враги.аdd (новый EnemyComstock  
(Level_data.BLOCK_SIZE * 117, 200));  
враги.аdd (новый EnemyRedEye (Level_data.BLOCK_SIZE  
* 133, 200));  
враги.аdd (новый EnemyRedEye (Level_data.BLOCK_SIZE  
* 156, 200));  
враги.аdd (новый EnemyComstock  
(Level_data.BLOCK_SIZE * 177, 200));  
враги.аdd (новый EnemyComstock  
(Level_data.BLOCK_SIZE * 193, 200));  
враги.аdd (новый EnemyRedEye (Level_data.BLOCK_SIZE  
* 201, 200));  
враги.аdd (новый EnemyRedEye (Level_data.BLOCK_SIZE  
* 216, 200));  
враги.аdd (новый EnemyRedEye (Level_data.BLOCK_SIZE  
* 224, 200));  
враги.аdd (новый EnemyComstock  
(Level_data.BLOCK_SIZE * 246, 200));  
враги.аdd (новый EnemyRedEye (Level_data.BLOCK_SIZE  
* 260, 200));  
враги.аdd (новый EnemyComstock  
(Level_data.BLOCK_SIZE * 277, 100));  
Level_data.enemyBlocks.add (новый блок  
(«невидимый», Level_data.BLOCK_SIZE * 34,  
Level_data.BLOCK_SIZE * 13));  
Level_data.enemyBlocks.add (новый блок  
(«невидимый», Level_data.BLOCK_SIZE * 36,  
Level_data.BLOCK_SIZE * 13));
```

```

Level_data.enemyBlocks.add (новый блок
(«невидимый», Level_data.BLOCK_SIZE * 60,
Level_data.BLOCK_SIZE * 9));
Level_data.enemyBlocks.add (новый блок
(«невидимый», Level_data.BLOCK_SIZE * 61,
Level_data.BLOCK_SIZE * 9));
Level_data.enemyBlocks.add (новый блок
(«невидимый», Level_data.BLOCK_SIZE * 106,
Level_data.BLOCK_SIZE * 7));
Level_data.enemyBlocks.add (новый блок
(«невидимый», Level_data.BLOCK_SIZE * 107,
Level_data.BLOCK_SIZE * 7));
Level_data.enemyBlocks.add (новый блок
(«невидимый», Level_data.BLOCK_SIZE * 168,
Level_data.BLOCK_SIZE * 13));
Level_data.enemyBlocks.add (новый блок
(«невидимый», Level_data.BLOCK_SIZE * 170,
Level_data.BLOCK_SIZE * 13));
Level_data.enemyBlocks.add (новый блок
(«невидимый», Level_data.BLOCK_SIZE * 196,
Level_data.BLOCK_SIZE * 13));
Level_data.enemyBlocks.add (новый блок
(«невидимый», Level_data.BLOCK_SIZE * 197,
Level_data.BLOCK_SIZE * 13));
Level_data.enemyBlocks.add (новый блок
(«невидимый», Level_data.BLOCK_SIZE * 232,
Level_data.BLOCK_SIZE * 8));
Level_data.enemyBlocks.add (новый блок
(«невидимый», Level_data.BLOCK_SIZE * 233,
Level_data.BLOCK_SIZE * 8));
перемена;
}
}

приватное статическое void update () {
    для (EnemyBase враг: враги) {
        enemy.update ();
        If (enemy.GetDelete ()) {
            enemies.remove (enemy);
            break;
        }
    }
    Bullet.update ();
    Controller.update ();
    booker.update ();
    If (!energetic.GetName().Equals("")) energetic.update ();
    if (levelNumber > 0) elizabeth.update ();
    if (lightning != null) {
        lightning.update ();
        If (lightning.GetDelete ()) lightning = null;
    }
    menu.update ();
    hud.update ();
    weapon.update ();
    if (booker.getTranslateX() > Level_data.BLOCK_SIZE * 295)
    cutScene = new CutScenes(levelNumber);
}

```

```

    }
    @Override

    public void start(Stage primaryStage) throws Exception {
        stage = primaryStage;
        scene = new Scene(appRoot, 1280, 720);
        initContent();
        stage.getIcons().add(new
        Image("file:/C:/DeadShock/images/icon.jpg"));
        stage.setTitle("DeadShock");
        stage.setResizable(false);
        stage.setWidth(scene.getWidth());
        stage.setHeight(scene.getHeight());
        stage.setScene(scene);
        stage.show();
        timer.start();
    }

    public static void main(String[] args) {
        launch(args);
    }
}

```

### **Задание самостоятельной работы**

1 В соответствии с индивидуальным вариантом, переписать программный код, используя общепринятые соглашения и рекомендации по именованию и форматированию переменных, операторов, выражений.

2 Провести инспекцию кода с помощью любого из перечисленных сервисов.

3 Сравнить полученные результаты. Сделать вывод.

Перечень индивидуальных вариантов приведен в приложении Г.

### **Контрольные вопросы**

1. Для чего нужны стандарты кодирования.
2. Какие существуют соглашения по именованию переменных, функций.
3. В каком формате следует записывать константы в тексте программы.
4. Какие стили расстановки скобок существуют? В чем их различия?

**Форма отчета:** тетрадь, защита работы.

## **Раздел 2. Средства разработки программного обеспечения**

### **Практическое занятие № 1**

**Цель:** формирование навыков постановки задачи и разработки технического задания на программный продукт

**Задание 1:** площадь квадрата по заданной стороне.

1. Выбрать вариант задания на проектирование и разработку учебной программы.

2. В соответствии с вариантом выполнить разработку технического задания, которое должно включать:

о введение;

о основание для разработки;

о назначение;

о требования к программе и программному продукту;

о требования к программной документации.

3. Оформить отчет. Содержание отчета:

о тема лабораторной работы

о цель лабораторной работы

о ответы на контрольные вопросы

о задание на лабораторную работу

о разработанное техническое задание

о выводы по проделанной работе.

### Варианты заданий

1. Ввести вещественную матрицу размерности  $n * m$  построчно, а вывести по столбцам.
2. Выяснить сколько положительных элементов содержит матрица размерности  $n * m$ , если  $a_{ij} = \sin(i+j/2)$ .
3. Дана квадратная вещественная матрица размерности  $n$ . Является ли матрица симметричной относительно главной диагонали.
4. Дана квадратная вещественная матрица размерности  $n$ . Транспонировать матрицу.
5. Дана квадратная вещественная матрица размерности  $n$ . Сравнить сумму элементов матрицы на главной и побочной диагоналях.
6. Дана квадратная вещественная матрица размерности  $n$ . Найти количество нулевых элементов, стоящих:  
выше главной диагонали;  
ниже главной диагонали;  
выше и ниже побочной.
7. Дана вещественная матрица размерности  $n * m$ . По матрице получить логический вектор, присвоив его  $k$ -ому элементу значение True, если выполнено указанное условие и значение False иначе:  
все элементы  $k$  столбца нулевые;  
элементы  $k$  строки матрицы упорядочены по убыванию;  
 $k$  строка массива симметрична.
8. Дана вещественная матрица размерности  $n * m$ . Сформировать вектор  $b$ , в котором элементы вычисляются как:  
произведение элементов соответствующих строк;  
среднее арифметическое соответствующих столбцов;  
разность наибольших и наименьших элементов соответствующих строк;  
значения первых отрицательных элементов в столбце.
9. Дана вещественная матрица размерности  $n * m$ . Вывести номера столбцов, содержащих только отрицательные элементы.
10. Дана вещественная матрица размерности  $n * m$ . Вывести номера строк, содержащих больше положительных элементов, чем отрицательных.
11. Дана вещественная матрица размерности  $n * m$ . Найти общую сумму элементов только тех столбцов, которые имеют хотя бы один нулевой элемент.
12. Дана вещественная матрица размерности  $n * m$ . Поменять местами строки с максимальным и минимальным элементами.
13. Дана вещественная матрица размерности  $n * m$ . Удалить  $k$  столбец матрицы.
14. Дана вещественная квадратная матрица размерности  $n$ . Поменять местами элементы главной и побочной диагоналей матрицы:  
  
по строкам;  
по столбцам.
15. Дана вещественная матрица размерности  $m * n$ . Упорядочить элементы каждой четной строки по возрастанию.
16. Дана вещественная матрица размерности  $m * n$ . Расположить все элементы матрицы по убыванию. Обход матрицы осуществлять по строкам.
17. Дана вещественная матрица размерности  $m * n$ . Определить индексы первого нулевого элемента матрицы. Обход матрицы осуществлять по столбцам.
18. Известно положение двух ферзей на шахматной доске. Бьют ли они друг друга?

### Контрольные вопросы

1. Перечислите этапы разработки программных продуктов.
2. Для чего необходимо техническое задание?
3. Кто занимается разработкой технического задания?
4. Какие пункты включает техническое задание?

**Форма отчета:** отчет, защита работы.

### Практическое занятие № 2

**Цель:** Научиться разрабатывать перечни артефактов и протоколы проекта

**Задание 1:**

## Системный анализ и пути решения задачи

При разработке ПС человек имеет дело с системами. Под *системой* будем понимать совокупность взаимодействующих (находящихся в отношениях) друг с другом элементов. ПС можно рассматривать как пример системы. Логически связанный набор программ является другим примером системы. Любая отдельная программа также является системой. Понять систему — значит осмысленно перебрать все пути взаимодействия между ее элементами.

Целью системного анализа в наиболее общем виде является описание и исследование систем, определение путей и методов разработки ПО. Система характеризуется структурой и поведением. Применительно к разработке ПО системный анализ представляет собой анализ существующей структуры отношений в рамках конкретной предметной области, выявление роли и места будущей программной системы, ее основных функций и свойств. В этой связи системный анализ также можно назвать *внешним проектированием*.

Этап системного анализа состоит из следующих трех стадий:

1. обоснование необходимости разработки программы;
2. научно-исследовательские работы (НИР);
3. разработка и утверждение технического задания.

На первой стадии выполняются постановка задачи, сбор исходных материалов, Выбор и обоснование критериев эффективности и качества разрабатываемой программы, обоснование необходимости проведения научно-исследовательских работ.

На стадии научно-исследовательских работ решаются следующие задачи: определяется структура входных и выходных данных, осуществляется предварительный выбор методов решения задач, обосновывается целесообразность применения ранее разработанных программ, определяются требования к техническим средствам, обосновывается принципиальная возможность решения поставленной задачи.

На стадии разработки и утверждения технического задания определяются требования к программе, разрабатываются технико-экономического обоснования разработки программ, определяются стадии, этапы и сроки разработки программы и документации на нее, согласовывается и утверждается *техническое задание*.

Результат системного анализа — **спецификация** (техническое задание) как самостоятельный документ имеет очень важное значение. Этот документ является формальным соглашением между заказчиком продукта и его разработчиками.

В настоящее время можно выделить пять основных подходов к организации процесса создания и использования программного обеспечения.

1. *Водопадный подход*. При таком подходе разработка ПС состоит из цепочки этапов. На каждом этапе создаются документы, используемые на последующем этапе. В исходном документе фиксируются требования к ПС. В конце этой цепочки создаются программы, включаемые в ПС.
2. *Исследовательское программирование*. Этот подход предполагает быструю (насколько это возможно) реализацию рабочих версий программ ПС, выполняющих лишь в первом приближении требуемые функции. После экспериментального применения реализованных программ производится их модификация с целью сделать их более полезными для пользователей. Этот процесс повторяется до тех пор, пока ПС не будет достаточно приемлемо для пользователей. Такой подход применялся на ранних этапах развития программирования, когда технологии программирования не придавали большого значения (использовалась интуитивная технология). В настоящее время этот подход применяется для разработки таких ПС, для которых пользователи не могут точно сформулировать требования (например, для разработки систем искусственного интеллекта).
3. *Прототипирование*. Этот подход моделирует начальную фазу исследовательского программирования вплоть до создания рабочих версий программ, предназначенных для проведения экспериментов с целью установить требования к ПС. В дальнейшем должна последовать разработка ПС по установленным требованиям в рамках какого-либо другого подхода (например, водопадного).
4. *Формальные преобразования*. Этот подход включает разработку формальных спецификаций ПС и превращение их в программы путем корректных преобразований.

На этом подходе базируется компьютерная технология (CASE-технология) разработки ПС.

5. *Сборочное программирование*. Этот подход предполагает, что ПС конструируется, главным образом, из компонентов, которые уже существуют. Должно быть некоторое хранилище (библиотека) таких компонентов, каждая из которых может многократно использоваться в разных ПС. Такие компоненты называются *повторно используемыми (reusable)*. Процесс разработки ПС при данном подходе состоит скорее из сборки программ из компонентов, чем из их программирования.

### **Задание**

1. В соответствии с подготовленным техническим заданием выполнить разработку спецификаций на программный продукт, которые должны включать:
  - о спецификации процессов;
  - о словарь терминов;
  - о диаграммы переходов состояний;
  - о диаграммы потоков с детализацией.
2. Оформить отчет. Содержание отчета:
  - о тема лабораторной работы;
  - о цель лабораторной работы;
  - о ответы на контрольные вопросы;
  - о задание на лабораторную работу;
  - о разработанные спецификации процессов;
  - о словарь терминов;
  - о диаграммы переходов состояний;
  - о диаграммы потоков с детализацией;
  - о выводы по проделанной работе.

### **Контрольные вопросы**

1. Для чего разрабатываются спецификации на программный продукт?
2. Что должны включать спецификации на программный продукт?
3. Что должна содержать спецификация процессов
4. Что такое словарь терминов и для чего он используется?
5. Что такое диаграмма переходов состояний и для чего ее используют?
6. Что такое диаграмма потоков и для чего ее используют?

**Форма отчета:** отчет, защита работы.

### **Практическое занятие № 3**

**Цель:** отработать навыки настройки работы системы контроля версий

**Задание 1:**

Система управления/контроля версиями (от англ. Version Control System или Revision Control System) — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости, возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение и многое другое.

Такие системы наиболее широко применяются при разработке программного обеспечения, для хранения исходных кодов разрабатываемой программы. Однако, они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов, в частности, они всё чаще применяются в САПР, обычно, в составе систем управления данными об изделии (PDM). Управление версиями используется в инструментах конфигурационного управления (Software Configuration Management Tools).

Распространённые системы управления версиями

- Subversion
- Darcs
- Microsoft Visual SourceSafe
- Bazaar
- Rational ClearCase
- Perforce
- BitKeeper
- Mercurial
- Git
- GNU Arch
- CVS — устаревшая. Потомок: Subversion
- RCS — устаревшая. Потомок: CVS

Основные понятия

Репозиторий (repository) – центральное хранилище, которое содержит версии файлов. Очень часто репозиторий организуется средствами какой-нибудь СУБД.

Версия файла (revision) – состояние файла в определенный момент времени. Репозиторий предоставляет возможность хранить неограниченное число версий одного и того же файла.

Актуальная версия файла – обычно это самая последняя версия файла, размещенного в репозитории.

Рабочая версия файла (working copy) – версия файла, с которой в текущий момент ведется работа, и которая не загружена в репозиторий.

Загрузка (Upload) – размещение файла в репозитории. В процессе загрузки в репозиторий помещается рабочая версия файла.

Выгрузка (Checkout) – получение файла из репозитория. В процессе выгрузки осуществляется получение из репозитория необходимой версии файла.

Синхронизация (update, sync) – приведение в соответствие рабочих версий файлов с актуальными версиями в репозитории. В процессе синхронизации в репозиторий загружаются те файлы, рабочие копии которых являются более "свежими" (т.е. имеют более поздние версии), по сравнению с файлами в репозитории, и выгружаются те файлы, рабочие копии которых устарели по сравнению с копиями в репозитории.

Borland StarTeam

Borland StarTeam – очень мощный и функциональный кросс-платформенный продукт, разрабатываемый в прошлом фирмой StarBase, которую Borland приобрела в конце 2002 г. Заметное преимущество данного решения состоит в том, что версия 2005 выступает центральным элементом стратегии управления жизненным циклом приложений (Application Lifecycle Management, ALM) компании Borland и обладает расширенными возможностями интеграции со всеми ее ключевыми пакетами, используемыми при разработке программного обеспечения.

MS SourceSafe

Microsoft Visual SourceSafe (Visual SourceSafe, VSS) — программный продукт компании Майкрософт, файл-серверная система управления версиями, предназначенная для небольших команд разработчиков. VSS позволяет хранить в общем хранилище файлы, разделяемые несколькими пользователями, для каждого файла хранится история версий. VSS входит в состав

- Subversion
  - [https://github.com/irgups/project\\_2015\\_01](https://github.com/irgups/project_2015_01)
  - [https://github.com/irgups/project\\_2015\\_02](https://github.com/irgups/project_2015_02)

**Форма отчета:** отчет, защита работы.

## Практическое занятие № 4

**Цель:** Приобретение навыков проектирования структур данных.

### Задание 1:

1. Начало работы в Microsoft SQL Server Management Studio

Для создания баз данных используем среду Microsoft SQL Server Management Studio.

На запрос соединения с сервером выбираем (рис. 1):

Тип сервера: **Компонент Database Engine**

Имя сервера: **SQL-MS.**

Под таким именем в домене fizmat.vspu.ru. доступна машина, на которой установлены серверные компоненты MS SQL Server 2005. Можно попробовать выбрать сервер из выпадающего списка серверов. Можно также обратиться к этой машине по IP-адресу 192.168.10.152 из локальной сети.

Проверка подлинности: **Проверка подлинности SQL Server.**

Такая настройка позволяет создавать пользователей данного экземпляра SQL Server независимо от компьютера, с которого производится вход.

Имя входа: **studentMBS21.**

Пароль: **student.**

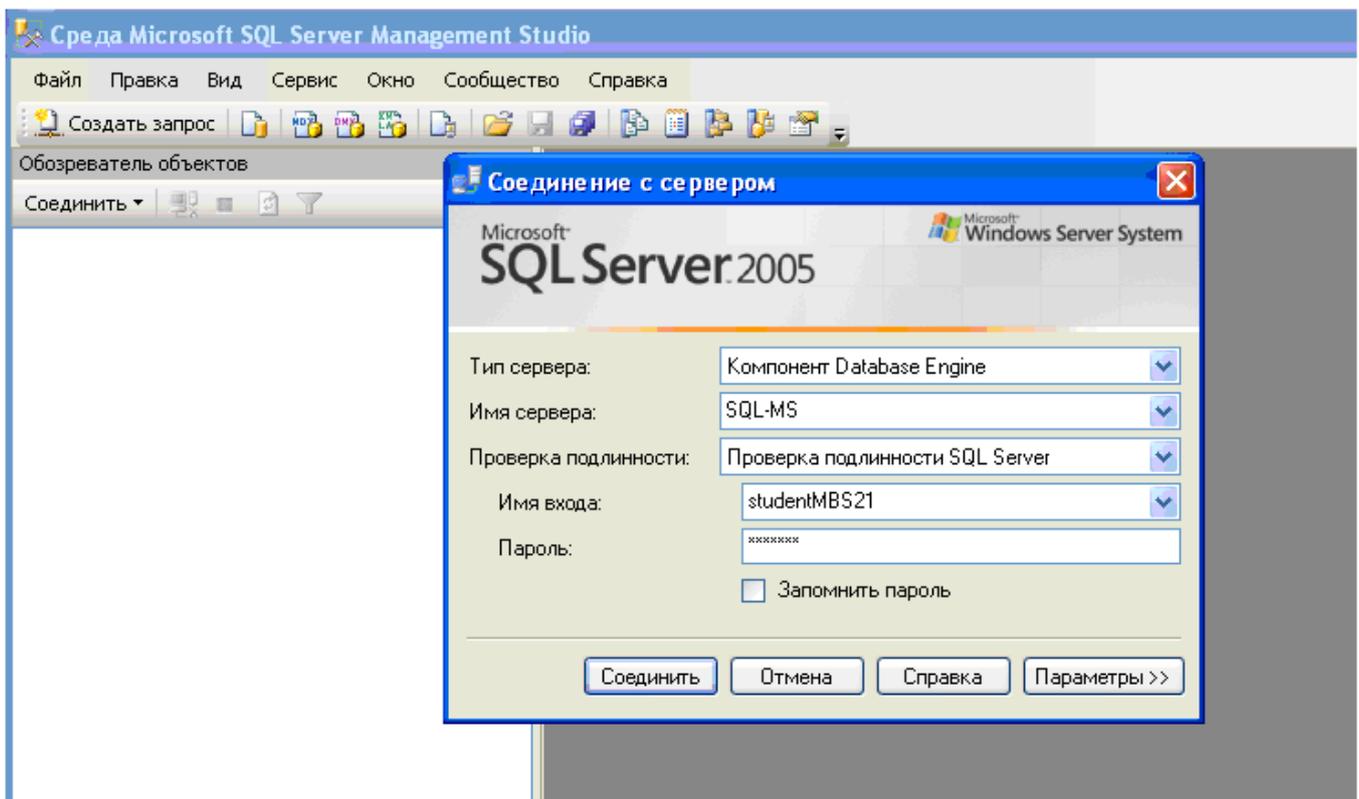


Рисунок 1. Окно входа в Microsoft SQL Server Management Studio 2005

Примечание. Пользователь studentMBS21 обладает большими полномочиями на этом сервере, поэтому пользоваться им надо очень аккуратно. Под этим пользователем мы создадим базу данных, а заполнять её и производить поиск по ней мы будем под другими пользователями. Предпочтительнее всего использовать свою учетную запись в домене fizmat.vspu.ru. В этом случае надо выбрать проверку подлинности Windows.

Теперь нажимаем кнопку «Параметры» и выбираем (рис. 2):

Соединение с базой данных → Обзор сервера... → Пользовательские базы данных → trial\_base.

Сетевой протокол → TCP/IP  
Нажимаем кнопку «Соединить».

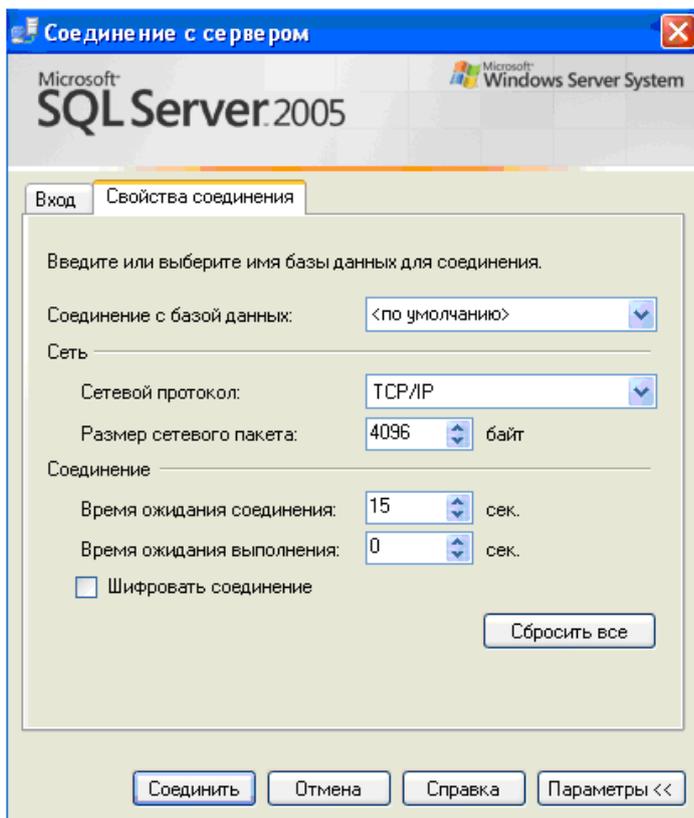


Рисунок 2. Окно входа в Microsoft SQL Server Management Studio 2005 (вкладка Параметры)

*Примечание.* База данных trial\_base является базой данной по умолчанию для пользователя studentMBS21, она была создана при регистрации этого пользователя. В случае, когда права доступа пользователя не ограничены (как в рассматриваемом случае), вкладку Параметры можно не открывать. Если же пользователь имеет доступ только к определенным базам данных, при подключении к серверу нужно одну из этих баз указывать.

После успешного соединения с базой данных на экране видим следующую картинку (рис. 3):

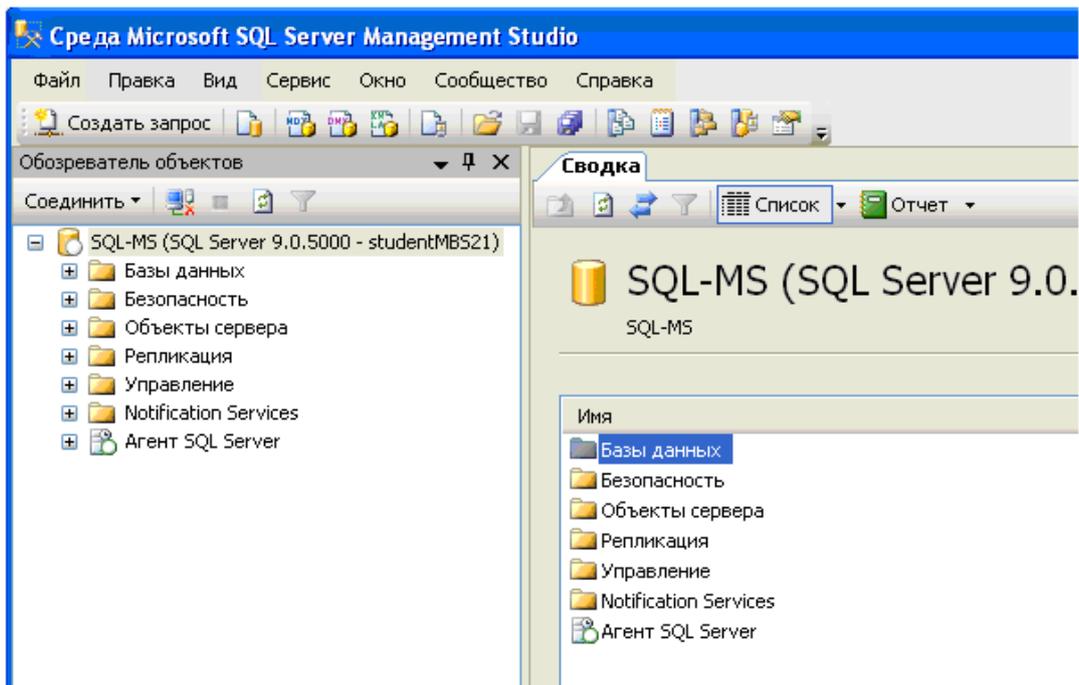


Рисунок 3. Подключение к SQL - серверу установлено

Среда MS SQL Management Studio предоставляет удобный инструмент для создания, редактирования, заполнения баз данных. Но настоящие профессионалы в своей работе редко пользуются этой средой, а для выполнения своих задач используют SQL-запросы. Мы будем пользоваться, когда это удобно и наглядно, графическим режимом, но основной упор будем делать на освоении базы языка SQL.

## 2. Создание базы данных в среде Microsoft SQL Server Management Studio

В разделе «Базы данных» правой кнопкой выбираем «Создать базу данных...» (рис. 4). Назовем базу данных по индексу группы – mbs21. Владельцем базы данных назначим пользователя, под именем которого был произведен вход – studentMBS21. В разделе «Параметры» выбираем тип сортировки Cyrillic\_General\_BIN (для примера), нажимаем ОК.

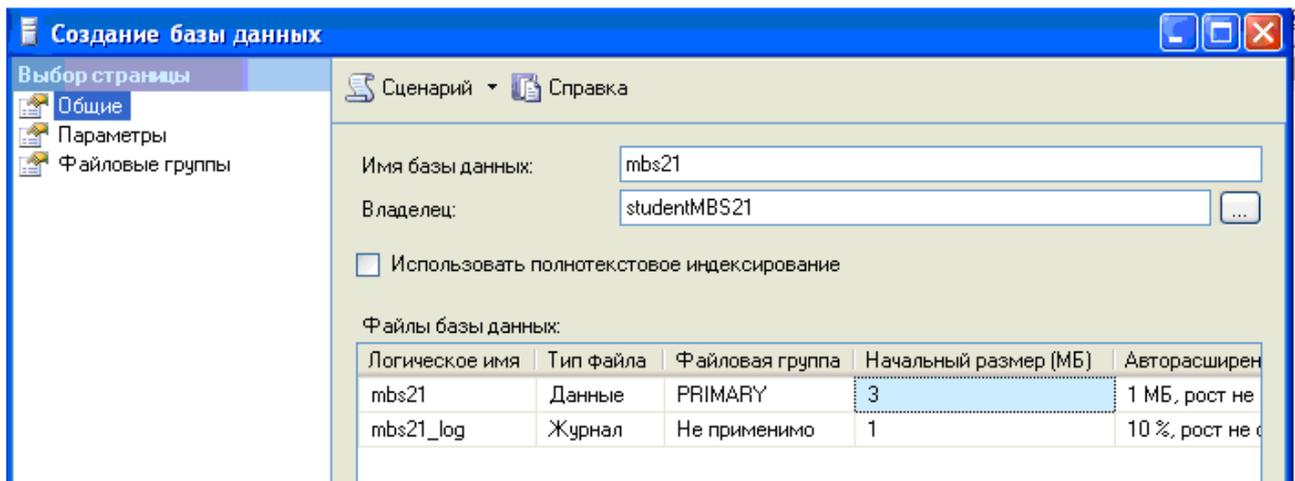


Рисунок 4. Создание базы данных

В разделе «Базы данных» Обзорера объектов появилась вновь созданная mbs21 (проверьте!):

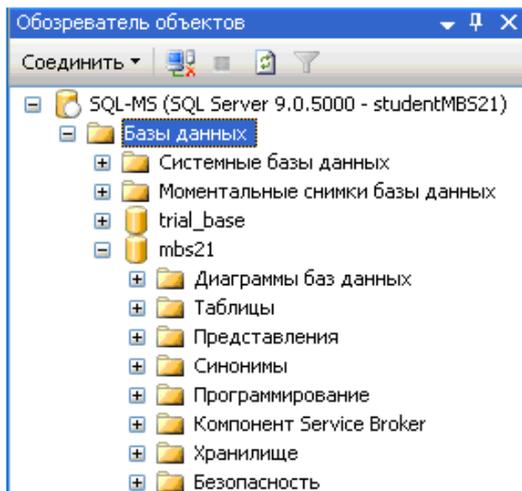


Рисунок 5. Обзорера объектов

### 3. Создание таблиц базы данных в среде Microsoft SQL Server Management Studio Начнем с создания таблицы Speciality. Структура таблицы приведена ниже:

Имя поля (столбца)	Содержание	Тип данных	Возможность содержать NULL
Num	Первичный ключ	int	нет
Name	Название специальности	varchar(60)	нет

В реляционных базах данных первичный ключ используется как уникальный идентификатор записи. Это поле является обязательным, оно используется для связи таблиц по внешним ключам (примеры такого связывания будут рассмотрены далее). Первичный ключ должен иметь целочисленный тип (в данном случае - *int*). Во втором поле будет храниться название специальности - некоторая строка, поэтому мы выбираем для этого поля тип *varchar(60)*. Число в скобках означает максимальное число символов в строке. Детальную информацию об этих типах можно посмотреть в справке.

Простейшим образом можно создавать таблицы средствами MS SQL Server Management Studio (правая кнопка мыши на заголовке «Таблицы» > Создать таблицу.). Получаем следующее:

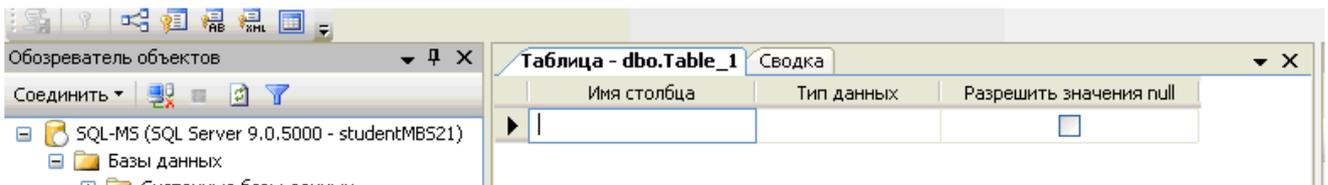


Рисунок 6. Создание таблицы

Вводим имя первого столбца Num (первичный ключ – в том столбце хранится номер записи), выбираем из выпадающего списка тип данных int. Первичный ключ не может быть пустым, поэтому и оставляем неотмеченным поле «Разрешить значения null». Затем аналогичным образом вводим имя второго столбца, задаем тип, запрещаем полю иметь значение null. Таблица принимает следующий вид:

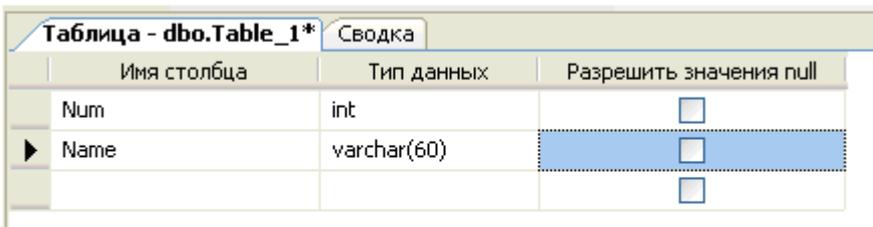
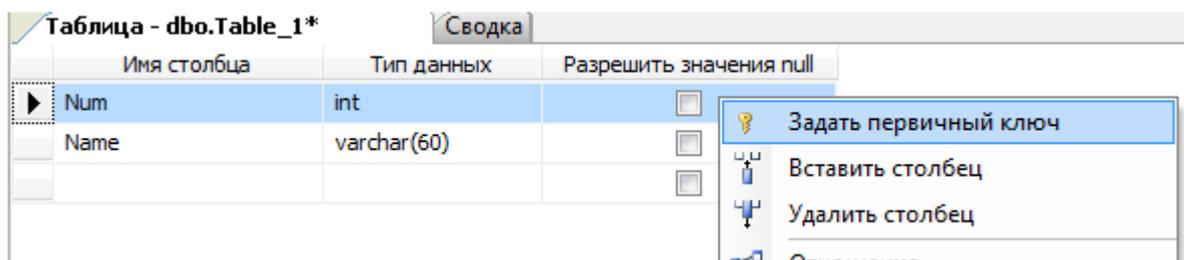


Рисунок 7.

Теперь необходимо указать, что поле Num будет являться первичным ключом. Правой кнопкой мыши щелкаем по этому полю и выбираем «Задать первичный ключ»:



Сохраняем таблицу под именем Speciality (после этого таблица должна появиться в обозревателе объектов). Теперь можно перейти к заполнению этой таблицы (для этого нужно в обозревателе объектов выбрать эту таблицу и в контекстном меню нажать «Открыть таблицу»):

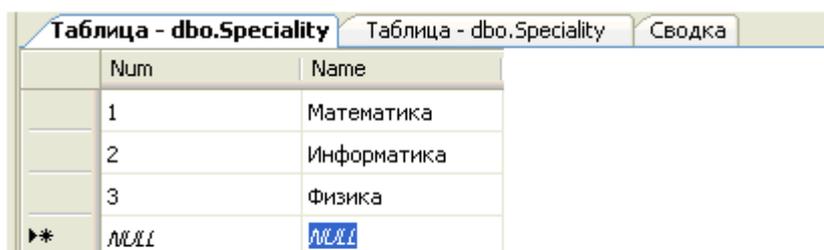


Рисунок 9.

При заполнении вы обнаружите, что каждый раз приходится вводить не только полезную информацию (название специальности), но и номер записи. Чтобы вводить номер записи автоматически, нужно задать спецификацию идентифицирующего столбца. Для этого необходимо в свойствах столбца указать, что данный столбец является идентифицирующим (рис. 10):

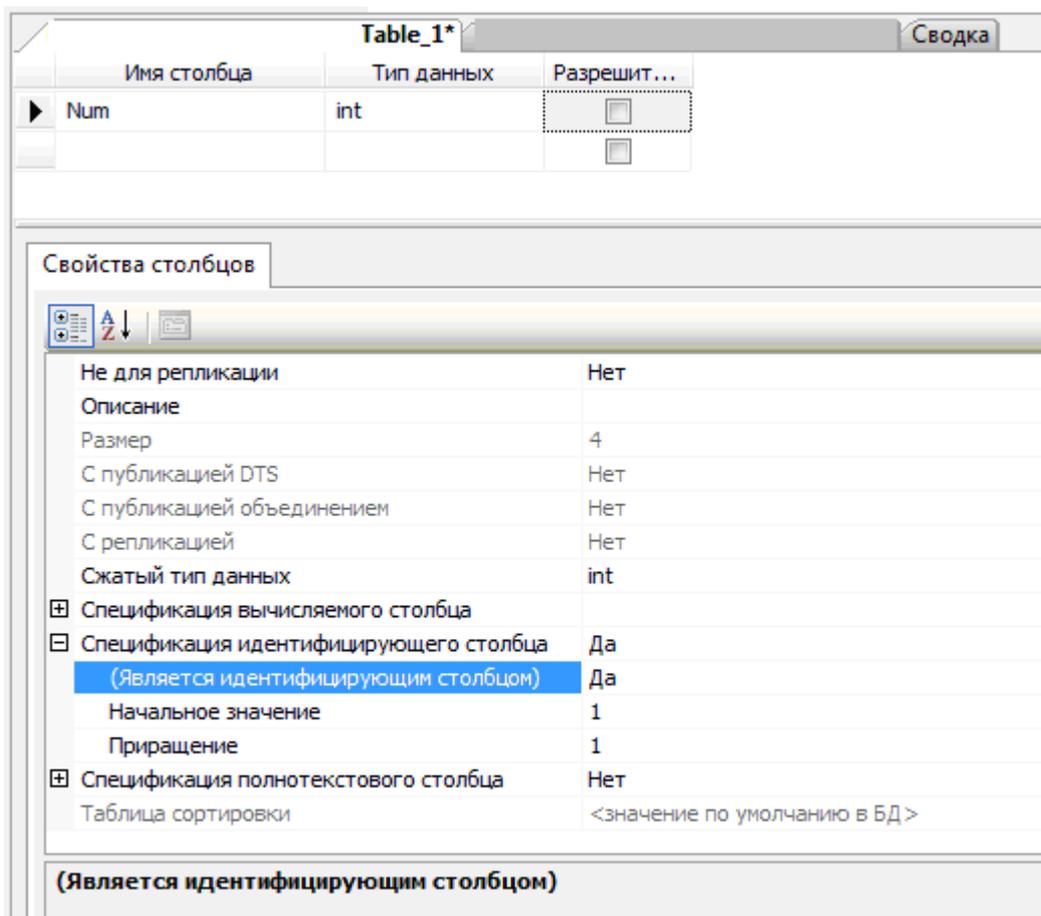


Рисунок 10. Определение свойств идентифицирующего столбца

#### 4. Создание таблиц базы данных с помощью SQL-запроса

Создание таблиц в графическом режиме, безусловно, удобно, однако не универсально. При использовании других средств разработки баз данных (например, IBM DB2) придется привыкать к новым приемам работы. Использование конструкций языка SQL позволяет работать с базами данных, исходя из единого подхода, в любой среде управления базами данных.

Выберите на панели инструментов «Создать запрос»:

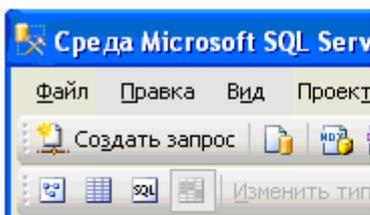


Рисунок 11.

Создадим новую базу данных запросом. Напишем

```
CREATE DATABASE mbs21_query
```

и нажмем F5. В обозревателе объектов должна появиться новая база (если сразу не появилась, то надо выделить мышью раздел «Базы данных» и в контекстном меню выбрать «Обновить»).

Теперь создадим таблицу Speciality. Упрощенный синтаксис создания таблиц следующий:

```
CREATE TABLE <имя таблицы> (
  <имя столбца 1> <тип данных> [NOT NULL] [DEFAULT <значение по умолчанию>],
  <имя столбца 2> <тип данных> [NOT NULL] [DEFAULT <значение по умолчанию>],
  ...
)
```

Введем новый запрос:

```
/* создание таблицы Специальность*/
USE mbs21_query -- определяем базу данных, в которую входит таблица
CREATE TABLE Speciality(
  Num INT IDENTITY(1,1) PRIMARY KEY NOT NULL, -- первичный ключ
  NameSpec VARCHAR(60) -- название специальности
)
```

В обозревателе объектов видим, что таблица действительно создана. Файл с SQL-запросом сохраняем в своей папке (в конце работы необходимо показать запросы, которые были выполнены, преподавателю). Слово IDENTITY(1,1) добавлено, чтобы поле первичного ключа Num автоматически нумеровалось начиная с единицы (фактически, эта конструкция определяет спецификацию идентифицирующего столбца).

Таким же образом необходимо создать остальные таблицы. Рассмотрим таблицу Course.

### Таблица Course (курс)

Имя поля (столбца)	Содержание	Тип данных	Возможность содержать NULL
Num	Первичный ключ	int	нет
Name	Название специальности	varchar(60)	нет
YearEntry	Год поступления	int	нет
YearFinal	Год выпуска	int	да
Speciality	Специальность (внешний ключ ссылается на первичный ключ таблицы Speciality)	int	нет

Эта таблица содержит поле *Speciality*, которое ссылается на первичный ключ таблицы *Speciality*. Чтобы создать такую таблицу, необходимо выполнить запрос:

```
/* создание таблицы Курс */
USE mbs21_query -- определяем базу данных, в которую входит таблица
CREATE TABLE Course(
  Num INT IDENTITY(1,1) PRIMARY KEY NOT NULL, -- первичный ключ
  YearEntry INT NOT NULL, -- год поступления
  YearFinal INT, -- год окончания
  Speciality INT FOREIGN KEY REFERENCES Speciality(Num) -- специальность,
  -- ссылка по внешнему ключу на поле Num таблицы Speciality
)
```

**Примечание.** Ссылку можно создать только на существующую таблицу. Задать ссылку по внешнему ключу можно и после создания таблицы

### Задание 2:

Создайте все остальные таблицы, указанные в Приложении, используя SQL – запросы.

Таблица **Speciality** (специальность)

Имя поля (столбца)	Содержание	Тип данных	Возможность содержать NULL
Num	Первичный ключ	int	Нет
Name	Название	varchar(60)	Нет

Таблица **Course** (курс)

Имя поля (столбца)	Содержание	Тип данных	Возможность содержать NULL
Num	Первичный ключ	int	нет
Name	Название специальности	varchar(60)	нет
YearEntry	Год поступления	int	нет
YearFinal	Год выпуска	int	да
Speciality	Специальность (внешний ключ ссылается на первичный ключ таблицы Speciality)	int	нет

Таблица **Group** (группа)

Имя поля (столбца)	Содержание	Тип данных	Возможность содержать NULL
Num	Первичный ключ	int	нет
Name	Название специальности	varchar(60)	нет
Course	Курс (внешний ключ ссылается на первичный ключ таблицы Course )	int	нет

Таблица **Discipline** (дисциплина)

Имя поля (столбца)	Содержание	Тип данных	Возможность содержать NULL
Num	Первичный ключ	int	Нет
Name	Название (возможные значения: программирование, алгебра...)	varchar(60)	Нет

Таблица **Account** (тип отчетности)

Имя поля (столбца)	Содержание	Тип данных	Возможность содержать NULL
Num	Первичный ключ	int	Нет
Name	Название (возможные значения: экзамен, зачет, дифференцированный зачет...)	varchar(30)	Нет

Таблица **Mark** (отметка)

Имя поля (столбца)	Содержание	Тип данных	Возможность содержать NULL
Num	Первичный ключ	int	Нет
Name	Название (возможные значения: зачтено, не зачтено, отлично, хорошо...)	varchar(30)	Нет
Value	Значение (возможные значения: 0, 1, ..., 5)	int	Нет

Таблица **Status** (академический статус студента)

Имя поля (столбца)	Содержание	Тип данных	Возможность содержать NULL
Num	Первичный ключ	int	Нет
Name	Название (возможные значения: обучается, отчислен, в академическом отпуске, в отпуске по уходу за ребенком)	varchar(60)	Нет

Таблица **Position** (должность)

Имя поля (столбца)	Содержание	Тип данных	Возможность содержать NULL
Num	Первичный ключ	int	Нет
Name	Название (возможные значения: ассистент, старший преподаватель, доцент...)	varchar(60)	Нет

Таблица **People** (люди)

Имя поля (столбца)	Содержание	Тип данных	Возможность содержать NULL
Num	Первичный ключ	int	Нет
LastName	Фамилия	varchar(30)	Нет
FirstName	Имя	varchar(30)	Нет
MiddleName	Отчество	varchar(30)	Да
Male	Пол	char(1)	Нет
BrthDate	День рождения	datetime	Да
Addr	Адрес	varchar(100)	Да

Таблица **Student** (студент)

Имя поля (столбца)	Содержание	Тип данных	Возможность содержать NULL
Num	Первичный ключ	int	Нет
People	Человек (внешний ключ ссылается на первичный ключ таблицы People)	int	Нет
Group	Группа (внешний ключ ссылается на первичный ключ таблицы Group)	int	Нет
StudNum	Номер студенческого билета	varchar(30)	Нет
Status	Академический статус студента (внешний ключ ссылается на первичный ключ таблицы Status)	int	Нет

Таблица **Teacher** (преподаватель)

Имя поля (столбца)	Содержание	Тип данных	Возможность содержать NULL
Num	Первичный ключ (табельный номер сотрудника)	int	Нет
People	Человек (внешний ключ ссылается на первичный ключ таблицы People)	int	Нет
Position	Должность (внешний ключ ссылается на первичный ключ таблицы Position)	int	Нет

Таблица **SemesterResults** (результаты сессии)

Имя поля (столбца)	Содержание	Тип данных	Возможность содержать NULL
Num	Первичный ключ	int	Нет
Student	Студент (внешний ключ ссылается на первичный ключ таблицы Student)	int	Нет
Semester	Порядковый номер семестра	int	Нет
Account	Тип отчетности (внешний ключ ссылается на первичный ключ таблицы Account)	int	Нет
Discipline	Дисциплина (внешний ключ ссылается на первичный ключ таблицы Discipline)	int	Нет
Teacher	Преподаватель (внешний ключ ссылается на первичный ключ таблицы Teacher)	int	Нет
Mark	Отметка (внешний ключ ссылается на первичный ключ таблицы Mark)	int	Нет
Date	Дата сдачи отчетности	DateTime	Нет

### Задание 3:

Ответьте на вопросы:

Каким образом можно получить доступ к MS SQL Server 2005?

С помощью каких средств можно создать таблицу для MS SQL Server 2005?

Что такое первичный ключ?

Каким образом можно создать автоматическую нумерацию строк таблицы?

Что означают Not Null?

**Форма отчета:** отчет, защита работы.

### Практическое занятие № 5

**Цель:** освоение технологии создания однотабличной БД.

#### Задание 1:

1. Создать структуру таблицы базы данных «Ученик», содержащую следующие поля: **фамилия, имя, школа, класс, дата рождения, вес**. Типы и форматы полей определить самостоятельно.
2. Определить **первичный ключ** таблицы.
3. В режиме **таблицы ввести** в базу данных **пять** любых записей.
4. **Добавить** в структуру таблицы после поля «дата рождения» поле «рост».
5. **Заполнить** поле «рост».
6. С помощью **мастера форм** создать **форму** для редактирования таблицы.
7. В режиме **формы ввести** в таблицу **пять** любых записей.
8. **Удалить** из структуры таблицы поле «вес».
9. **Удалить** из таблицы **вторую** и **пятую** записи.

**Форма отчета:** отчет, защита работы.

### Практическое занятие №6

**Цель:** обобщить и систематизировать знания и умения по теме "Базы данных"

#### Задание 1:

1. Создайте базу данных «**Воскресная школа программирования**», содержащую три таблицы: **Группы, Списки, Личные данные**.

1. Создайте структуру таблицы **Группы**. Поле «Учебная группа» сделайте ключевым. Сохраните таблицу, не заполняя её.
2. Создайте структуру таблицы **Списки**. Ключевое поле Код студента (тип числовой). Сохраните таблицу, не заполняя её.
3. Создайте структуру таблицы **Личные данные** по образцу. Сохраните таблицу, не заполняя её.
4. Установите связи между таблицами
5. Заполните таблицу **Группы**
6. Для заполнения двух других таблиц создайте соответствующие формы.

**Таблица Группы**

Учебная группа	Преподаватель
101	Верзаков С.А.
102	Белоусов А.И.
103	Масалова В.А.
104	Новикова Е.В.
105	Зачёсова Т.П.

**Таблица Списки**

Код студента	Фамилия	Имя	Отчество	Год рождения	Школа	Класс	Учебная группа
1	Иванова	Анна	Ивановна	2001	1	9	101
2	Баранова	Ирина	Алексеевна	2000	3	10	102
3	Корнилова	Ольга	Владимировна	2001	5	9	103
4	Воробьёв	Алексей	Петрович	2000	1	10	101
5	Воробьёв	Алексей	Иванович	2001	3	9	104
6	Воробьёв	Олег	Григорьевич	2002	5	8	105
7	Скоркин	Александр	Евгеньевич	1999	1	11	101
8	Володина	Анна	Алексеевна	2001	3	9	102
9	Новосёлов	Алексей	Антонович	2000	5	10	103
10	Александрова	Елена	Алексеевна	2001	1	9	101

**Таблица Личные данные**

Код студента	Адрес	Номер телефона
1	Центральная 11-15	5-17-22
2	Солнечная 8-117	5-18-22
3	Сиреневый 7-16	5-19-22
4	Центральная 14-81	5-20-22
5	Сиреневый 7-16	5-21-22
6	Солнечная 2-121	5-22-22
7	Школьная 5-34	5-23-22
8	Центральная 30-7	5-24-22
9	Сиреневый 7-16	5-25-22
10	Солнечная 6-34	5-26-22

2. С помощью запросов выполнить следующие задания:

1. Вывести учащихся, год рождения которых 2001.
2. Вывести список учащихся школы №5, 10 класса.
3. Вывести учащихся, родившихся позднее 2000 года.
4. Вывести список учащихся с номерами телефонов (показать поля «Фамилия», «Имя», «Отчество», «Номер телефона»).
5. Вывести информацию о девочках, имя которых Анна (показать поля «Фамилия», «Имя», «Отчество», «Адрес», «Школа», «Класс»).

**Форма отчета:** отчет, защита работы.

### Практическое занятие №7

**Цель:** обобщить и систематизировать знания и умения по теме "Базы данных"

#### Задание 1:

Имеются следующие данные по регистрации заказов на товары:

Код товара	Тип товара	Производитель	Модель	Дата заказа	Цена	Количество	№ заказа
851365125874	Весы	Marta	M-150	12.12.12	980	20	948
680587369521	Кофемолка	Bork	B-56	19.12.12	650	12	1001
460123654987	Комбайн	LG	L1200	12.12.12	4320	10	1002
680587369521	Кофемолка	Bork	B-56	16.12.12	650	5	1011
461357951852	Нагреватель	Омск	H-1500	13.12.12	680	5	1003
481258741951	Кипятильник	Казань	ЭК-11	12.11.12	62	25	900
851365125874	Весы	Marta	M-150	16.12.12	980	10	1007
851365125874	Весы	Marta	M-150	2.12.12	980	8	1008
680587369521	Кофемолка	Bork	B-56	14.12.12	650	10	1005
460123654987	Комбайн	LG	L1200	16.12.12	4320	10	1006
569258147456	Утюг	Braun	DT-16	12.12.12	3240	5	948
481258741951	Кипятильник	Казань	ЭК-11	17.12.12	62	23	1012
461357951852	Нагреватель	Омск	H-1500	18.12.12	680	19	1013
569258147458	Утюг	Bork	BT-16	12.11.12	1320	15	904
460123654988	Комбайн	LG	G500	19.12.12	3240	5	1001
654321987597	Утюг	LG	L569	14.12.12	3240	3	1005
481258741951	Кипятильник	Казань	ЭК-11	12.12.12	62	25	1002
680587369527	Кофемолка	LG	G196	2.12.12	1650	7	1008

1. В Access создать базу данных «Регистрация заказов», состоящую из двух таблиц («Товары» и «Заказы»). Состав полей каждой таблицы, типы данных, ключевые поля назначить самостоятельно. Учесть, что одинаковые товары имеют одинаковые коды, не должно быть дублирующих записей, один и тот же товар за день может быть заказан более одного раза (по разным заказам). По одному заказу – только разные товары.
2. Заполнить созданную базу имеющимися данными.  
Отсортировать таблицу «Товары» по стоимости - в тетрадь зафиксировать самый дешёвый и самый дорогой товар.
3. Создать и сохранить следующие запросы:
  - a. Все данные из обеих таблиц (как в исходной таблице в начале задания).
  - b. Данные о заказах товаров с количеством больше 8 единиц в заказе, с информацией о цене и производителе товара.
  - c. Данные о заказах товаров на 12 декабря.
  - d. Данные о стоимости заказов по каждому товару (вычисляемое поле - произведение цены на количество) с указанием типа, модели и производителя товара, даты заказа. Отсортировать таблицу «Заказы» по номеру в тетрадь зафиксировать самый дешёвый и самый дорогой заказ.
  - e. Данные о том, когда и сколько заказывали товар фирмы LG, а также их стоимость и информацию о модели.
4. Создать форму для ввода, просмотра и редактирования данных таблиц.

При создании формы учесть: в таблице «Товары» - поле для фото товара (изображение необходимо отыскать в интернете), каждой форме присвоить заголовок, в формах применить размер шрифтов более 10 пт.

**Форма отчета:** отчет, защита работы.

### Практическое занятие № 8

**Цель:** Изучение объектов Visual Basic for Application на примере линейной программы.

**Задание 1:** ознакомьтесь с материалом

**Форма (UserForm)** представляет собой диалоговое окно, в котором можно размещать различные элементы управления. В приложении может быть как одна, так и несколько форм. Новая форма добавляется в проект выбором команды **Insert(Вставка) → UserForm**.

В **VBA** имеется обширный набор встроенных элементов управления.

Элементы управления являются объектами. Как любые объекты, они обладают свойствами, методами и событиями.

1. Элементы управления создаются при помощи **Панели элементов**, которая отображается на экране либо выбором команды **Вид (View) → Панель элементов (Toolbox)**, либо нажатием кнопки  панели инструментов **Standard**.

На этой панели представлены кнопки, позволяющие конструировать элементы управления. Для создания элементов управления служат все кнопки панели инструментов, за исключением кнопки **Выбор объекта** .

**Таблица 1.** Основные элементы управления и соответствующих кнопок панели элементов.

Элемент управления	Название	Кнопка	Элемент управления	Название	Кнопка
Поле	TextBox		Переключатель	OptionButton	
Надпись	Label		Флажок	CheckBox	
Кнопка	CommandButton		Рисунок	Image	
Список	ListBox		Поле со списком	ComboBox	

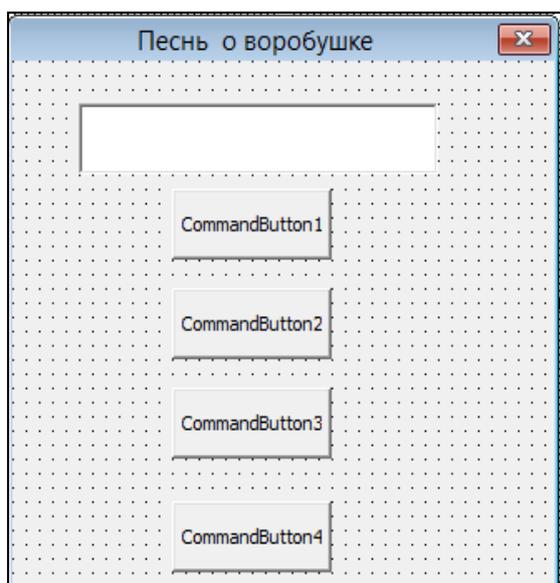
**Таблица 2.** Основные общие свойства элементов управления.

Свойство	Описание
Caption	Надпись, отображаемая при элементе управления.
Visible	Допустимые значения: True (элемент управления отображается во время выполнения программы) и False (в противном случае).
Enabled	Допустимые значения: True (пользователь вручную может управлять элементом управления) и False (в противном случае).
Height, Width	Устанавливают геометрические размеры объекта (высоту и ширину).
Left и Top	Устанавливают координаты верхнего левого угла элемента управления, определяющие его местоположение в форме.
BackColor, ForeColor	Устанавливают цвет заднего и переднего плана элемента управления, также его границы.

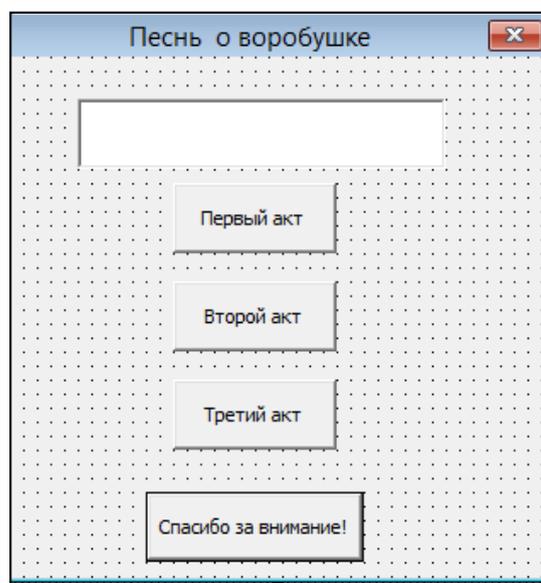
Свойство	Описание
BackStyle	Устанавливает тип заднего фона.
Picture (создание картинки)	Внедряет картинку на элемент управления. Например, на поверхности кнопки картинка отображается с помощью следующей инструкции: <code>CommandButton1.Picture = LoadPicture("c:\my doc\Круг.bmp")</code> .
Picture (удаление картинки)	После того как картинка создана на элементе управления, иногда возникает необходимость ее удалить. Это легко достигается присвоением свойству Picture значения <code>LoadPicture("")</code> <code>CommandButton1.Picture = LoadPicture("")</code>

После размещения элементов управления на форме необходимо связать объект на форме с кодом.

**Задание 2:** Создать пользовательскую форму на примере простейшего диалогового окна.



**Рисунок 1**



**Рисунок 2**

1. Откройте новую рабочую книгу.
2. Нажмите комбинацию клавиш **Alt +F11** для открытия редактора Visual Basic.
3. Выполнить команду **Insert / UserForm**. В редакторе появятся *окно с пользовательской формой* и *панель с элементами управления*.
4. Используя панель инструментов, заполните пользовательскую форму элементами управления, создав требуемое диалоговое окно приложения.

Для этого на форму поместите элементы управления: четыре кнопки(CommandButton); одно текстовое поле (TextBox).

5. Используя диалоговое окно **Свойства (Properties)**, отображаемое нажатием кнопки



, свойство *Caption* пользовательской формы определить равным *Песнь о воробушке и бабочке*, а кнопок - равными *Первый акт*, *Второй акт* и *Третий акт*, *Спасибо за внимание*.

6. Написать процедуру обработки события - нажатие кнопки.

Для написания процедуры обработки события - нажатия кнопки *Первый акт*, дважды ее щелкнуть. Откроется редактор кода на листе модуля **UserForm1**.

Введите текст процедуры:

```
Private Sub CommandButton1_Click()
    TextBox1.Text = "Воробышек за бабочкой скок-скок-скок"
End Sub
```

Для написания процедуры обработки событий - нажатий кнопок *Второй акт* и *Третий акт* ввести аналогичные процедуры:

```

Private Sub CommandButton2_Click()
    TextBox1.Text = "Воробышек за бабочкой прыг-прыг-прыг"
End Sub
Private Sub CommandButton3_Click()
    TextBox1.Text = "Воробышек бабочку ням-ням-ням"
End Sub
Private Sub CommandButton4_Click()
End
End Sub

```

Процесс создания диалогового окна и процедур, связанных с ним, завершен. Для выполнения созданной программы нажать кнопку  или выполнить команды меню **Run / Run Sub UserForm** или нажать функциональную клавишу **F5**.

**Форма отчета:** отчет, защита работы.

### Практическое занятие № 9

**Цель:** Овладение основными приемами создания Web-страниц на языке HTML

#### Задание 1:

Создать веб-сайт, рассказывающий о башнях Московского Кремля.

1. Создать специальную папку, дать ей имя Kremlin.
- 2.. В ней сохранить веб-страницу и графические файлы (рисунки) Кремля.

Создание Web-сайта «Московский кремль»

1. Запустить текстовый редактор Блокнот (туда скопировать теги из файла «основные теги»).
2. Дать странице название «Башни Московского Кремля»
3. Между тегами <BODI> </BODI> вставить скопированный текст из файла «История Кремля».
4. Сохранить под именем index.htm в папке Kremlin.
5. Просмотреть полученную страницу в браузере (открыть с помощью Internet Explorer)

**Форма отчета:** отчет, защита работы.

### Практическое занятие № 10

**Цель:** отработать навыки работы с базами данных при разработке веб приложений

#### Задание 1:

Задание 1. Установка и настройка IIS:

- 1) добейтесь отображения на мониторе компьютера окна «Компоненты Windows»; 2) позиционируйте дерево настроек в окне «Компоненты Windows» таким образом, чтобы были видны первоначальные настройки пункта «службы IIS»;
  - 3) получите скриншот окна «Компоненты Windows» с первоначальными настройками служб IIS и включите его в отчет по контрольной работе;
  - 4) отметьте все необходимые компоненты IIS и выполните их установку;
  - 5) перезагрузите операционную систему.
- Задание 2. Проверка работоспособности IIS:
- 1) с помощью командной строки запустите приложение IIS Manager и убедитесь, что на мониторе отобразилось окно, аналогичное представленному на рис. 1.4;
  - 2) получите скриншот окна «Диспетчер служб IIS» и включите его в отчет по контрольной работе;
  - 3) запустите браузер, в адресной строке введите http://localhost; убедитесь, что в браузере отобразилось окно, аналогичное представленному на рис. 1.5;
  - 4) получите скриншот окна браузера со стартовой страницей IIS 7 и включите его в отчет по контрольной работе.

Задание 3. Контрольные вопросы:

- 1) поясните термины: «сервер», «клиент», «браузер», «HTTP», «HTML», «web-приложение»;
- 2) поясните термины: «IIS», «IIS Manager» «ASP.NET», «Web Forms», «MVC»; 3
- 3) поясните термины: «SOAP», «web-сервис», «WCF».

**Форма отчета:** отчет, защита работы.

### **Практическое занятие № 11**

**Цель:** Овладение основными приемами разработки веб приложения

#### **Задание 1:**

Задание 1. Создание простейшего приложения Web Forms ASP.NET:

- 1) запустите Visual Studio и создайте шаблонное приложение Web Forms ASP.NET; 2) сделайте скриншот окна «Обозреватель решений» проекта и поместите его в отчет по контрольной работе;
- 3) откройте в проекте файл Default.aspx и замените текст «Добро пожаловать в ASP.NET!» на текст, содержащий ваше имя, фамилию и отчество;
- 4) запустите созданное web-приложение на выполнение в отладочном режиме;
- 5) сделайте скриншот окна браузера, отображающего стартовую страницу приложения, и поместите его в отчет по контрольной работе;
- 6) откройте окно отладочного сервера ASP.NET Development Server, сделайте скриншот и поместите его в отчет по контрольной работе.

Задание 2. Публикация приложения на IIS:

- 1) запустите приложение IIS Manager и с его помощью создайте сайт;
- 2) опубликуйте в этот сайт созданное в задании 1 приложение методом простого копирования файлов;
- 3) убедитесь в работоспособности сайта;
- 4) с помощью приложения IIS Manager создайте еще один сайт;
- 5) опубликуйте в последний созданный сайт созданное в задании 1 приложение методом отличным от простого копирования файлов;
- 6) сделайте скриншот окна приложения IIS Manager, который бы демонстрировал наличие двух созданных в этом задании сайтов; поместите скриншот в отчет по контрольной работе.

Задание 3. Контрольные вопросы:

- 1) поясните термины: «шаблон приложения», «ASP.NET Development Server»;
- 2) поясните термины: «сайт», «публикация приложения»;
- 3) что такое пакет развертывания приложения, как он может быть сформирован и как применен?
- 4) перечислите параметры, которые необходимо указать при создании сайта с помощью приложения IIS Manager;
- 5) перечислите известные вам способы публикации web-приложений.

**Форма отчета:** отчет, защита работы.

### **Практическое занятие № 12**

**Цель:** Овладение основными приемами разработки веб приложения

#### **Задание 1:**

Задание 1. Исследование структуры приложения и модели со- бытий Web Forms ASP.NET:

- 1) создайте приложение Web Forms ASP.NET с помощью Visual Studio;
- 2) измените страницу Default.aspx так, чтобы она была такой же, как на рис. 3.4; Рис. 3.5. Файл Default.aspx

```

<%@ Page Title="Домашняя страница" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="WebApplication5._Default" %>

<asp:Content ID="HeaderContent" runat="server" ContentPlaceHolderID="HeadContent"></asp:Content>
<asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainContent">
  <h2> Практическая работа № 3 </h2>
  <p>
    <br /><br />
    <asp:Button ID="Button1" runat="server" Text="Button1"
      onclick="Button1_Click" />
    <asp:Button ID="Button2" runat="server" Text="Button2" style="margin-left: 34px"
      onclick="Button2_Click" />
    <br /><br />
    <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
  </p>
</asp:Content>

```

Рис. 3.5. Файл Default.aspx

3) откорректируйте программный код в файле Default.aspx.cs, чтобы он стал таким же, как на рис. 3.6; отличаться может имя пространства имен (указывается в операторе namespace), которое в шаблонном приложении совпадает с именем приложения, задаваемым при создании; 24 Рис. 3.6. Файл Default.aspx.cs

```

namespace WebApplication5
{
    public partial class _Default : System.Web.UI.Page
    {
        private int k = 0;
        protected void Page_Init(object sender, EventArgs e)
        { this.Label1.Text += "Init[" + (++k).ToString() + "]-"; }

        protected void Page_Load(object sender, EventArgs e)
        { this.Label1.Text += "Load[" + (++k).ToString() + "]-"; }

        protected void Button1_Click(object sender, EventArgs e)
        { this.Label1.Text += "Click1[" + (++k).ToString() + "]-"; }

        protected void Button2_Click(object sender, EventArgs e)
        { this.Label1.Text += "Click2[" + (++k).ToString() + "]-"; }

        protected void Page_PreRender(object sender, EventArgs e)
        { this.Label1.Text += "PreRender[" + (++k).ToString() + "]-"; }

        protected void Page_Unload(object sender, EventArgs e)
        { this.Label1.Text += "Unload[" + (++k).ToString() + "]-"; }
    }
}

```

Рис. 3.6. Файл Default.aspx.cs

4) запустите приложение на выполнение в отладочном режиме; убедитесь, что в окне браузера отображается форма такая же, как на рис. 3.7; Рис. 3.7. Отображение формы Default.aspx в окне браузера

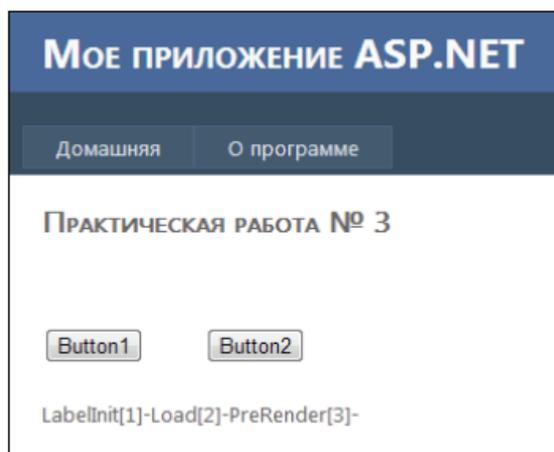


Рис. 3.7. Отображение формы Default.aspx в окне браузера

- 1) сделайте скриншот окна браузера, отображающего страницу Default.aspx приложения и поместите его в отчет по контрольной работе;
- 2) с помощью мыши нажмите клавишу Button1; обратите внимание на изменение текста строки, расположенной под кнопками;
- 3) сделайте скриншот окна браузера, отображающего страницу Default.aspx приложения и поместите его в отчет по контрольной работе;
- 4) с помощью мыши нажмите клавишу Button2; обратите внимание на изменение текста строки, расположенной под кнопками;
- 5) сделайте скриншот окна браузера, отображающего страницу Default.aspx приложения и поместите его в отчет по контрольной работе;
- 6) опубликуйте приложение на IIS и убедитесь в его работоспособности.

Задание 2. Контрольные вопросы:

- 1) перечислите все типы файлов, которые могут входить в приложение Web Forms ASP.NET, поясните их назначение;
- 2) перечислите все наименования стандартных папок приложения Web Forms ASP.NET, поясните их назначение;
- 3) перечислите в порядке их появления все события, которые могут быть обработаны методами класса Global (файл Global.asax.cs);
- 4) перечислите все события страницы в порядке их появления; 5) объясните принцип формирования строки на форме приложения, разработанного в задании 1

**Форма отчета:** отчет, защита работы.

### Практическое занятие № 13

**Цель:** отработать навыки по созданию и выполнению команд к источникам данных

**Задание 1:**

Задание 1. Разработка HTTP-обработчика ASP.NET:

- 1) разработайте HTTP-обработчик, реагирующий на GET-запрос страниц с расширением .bstu; HTTP-обработчик должен принимать значения двух параметров: faculty и course; в результате выполнения в окне браузера должна отобразиться следующая строка: GET: .bstu, faculty = val\_parm1, course = val\_parm2, где val\_parm1 и val\_parm2 – значения первого и второго параметров.
- 2) разработайте HTTP-обработчик, реагирующий на POST-запрос страниц с расширением .bstu; HTTP-обработчик должен принимать значения двух параметров: faculty и course; в результате выполнения в окне браузера должна отобразиться следующая строка: POST: .bstu, faculty = val\_parm1, course= val\_parm2, где val\_parm1 и val\_parm1 – значения первого и второго параметров.

- 3) опубликуйте HTTP-обработчики на IIS;
- 4) с помощью приложения IIS Manager отобразите окно «сопоставление обработчиков», найдите разработанные в предыдущих пунктах задания обработчики, сформируйте скриншот и поместите его в отчет по контрольной работе;
- 5) с помощью браузера выполните GET-запрос к HTTP-обработчику и убедитесь в его работоспособности;
- 6) разработайте web-форму, формирующую POST-запрос к HTTP-обработчику; опубликуйте эту web-форму IIS; выполните с помощью браузера запрос к форме, сформируйте POST-запрос и убедитесь в работоспособности обработчика; текст разработанной web-формы поместите в отчет по контрольной работе;
- 7) поместите исходные коды HTTP-обработчиков в отчет по контрольной работе.

**Задание 2.** Разработка клиента, взаимодействующего с HTTP-обработчиком ASP.NET:

- 1) разработайте консольное приложение на языке C#, формирующее GET и POST-запросы к HTTP-обработчикам, разработанным в задании 1; приложение должно отображать ответ, опривленный обработчиками в окне консоли;
- 2) выполните приложение, разработанное в предыдущем пункте задания; убедитесь в работоспособности приложения и HTTP-обработчиков; сформируйте скриншот, отражающий результат выполнения приложения; поместите скриншот и исходный код приложения в отчет по контрольной работе.

**Задание 3.** Контрольные вопросы:

- 1) поясните понятие «HTTP-обработчик ASP.NET»;
- 2) каким образом можно отобразить перечень HTTP-обработчиков, действующих в рамках сайта?
- 3) поясните структуру HTTP-обработчика (интерфейс, назначение методов);
- 4) где располагается информация связывающая класс HTTP-обработчика, метод HTTP-запроса и расширение страниц;
- 5) в каких случаях целесообразно применять HTTP-обработчики?

**Форма отчета:** отчет, защита работы.

### **Практическое занятие № 14.**

**Цель:** отработать навыки по созданию таблиц

**Задание 1:**

В таблице могут использоваться следующие способы выравнивания:

По горизонтали: - слева; По вертикали: - вверх.	По горизонтали: - слева; По вертикали: - по центру. (этот способ используется по умолчанию)	По горизонтали: - слева; По вертикали: - вниз.
По горизонтали: - по центру; По вертикали: - вверх.	По горизонтали: - по центру; По вертикали: - по центру.	По горизонтали: - по центру; По вертикали: - вниз.
По горизонтали: - справа; По вертикали: - вверх.	По горизонтали: - справа; По вертикали: - по центру.	По горизонтали: - справа; По вертикали: - вниз.

Сохраните страницу в личной папке в файл table1.html

### Форматирование заднего фона таблиц, рядов и ячеек:

1	2	3
4	5	6
7	8	9

**Форма отчета:** отчет, защита работы.

### Практическое занятие № 15

**Цель:** Научиться управлять процессом передачи данных от источника данных к DataSet.

#### Задание 1:

Создайте базу данных EmployeeInfo средствами Visual Studio.Net в SQL Server. Модель базы данных представлена на рис. 3.2.

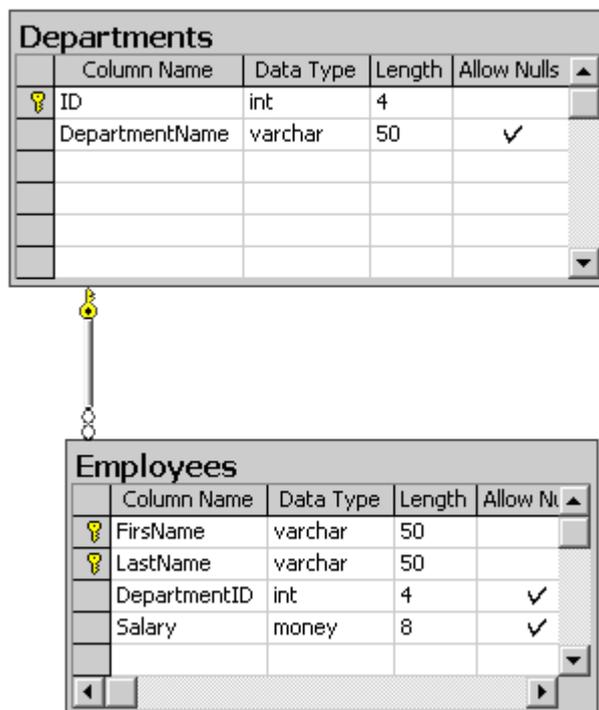


Рис. 3.2. Модель базы данных EmployeeInfo

2. Установите связи и заполните таблицы необходимой информацией.
3. Откройте форму "Data Set" проекта к практикуму по теме "Отсоединенная модель программирования".
4. Создайте новую кнопку DataAdapterFill. Вставьте в процедуру обработки щелчка по командной кнопке код, приведенный ниже. Не забудьте импортировать необходимые пространства имен:

```
Imports System.Data, System.Data.SqlClient
Private Sub Button7_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button7.Click
    ReadData()
End Sub
```

Код процедуры общего назначения для чтения данных из источника данных ReadData():

```
Private Sub ReadData()
    Dim rows As Integer
    Dim daDep As SqlDataAdapter = New SqlDataAdapter("select * from Departments", "server=Persist Security Info=False; Integrated Security=SSPI; database=EmployeeInfo; server=abrzh")
    dsE = New DataSet
    rows = daDep.Fill(dsE, "Departments")
    DisplayDataset(dsE)
End Sub
```

После создания объекта daDep с двумя аргументами, которые содержат команду Select и строку подключения, вызывается метод Fill для наполнения данными таблицы Departments набора данных dsE. Метод Fill также возвращает количество записей, которые включены (или обновлены) в набор данных dsE. При выполнении метода Fill провайдером данных неявно выполняются следующие действия:

- Для объекта `SelectCommand` открывается подключение к источнику данных, если оно еще не открыто.
- Выполняется команда, указанная в свойстве `CommandText` объекта `SelectCommand` (вместе с параметрами, если таковые имеются).
- Создается объект `DataReader` для возвращения имен полей и типов, использованных для создания нового объекта `DataTable` в указанном наборе данных `DataSet`, если этого объекта еще не существует.
- Объект `DataReader` используется для извлечения данных и вставки их в таблицу.
- Объект `DataReader` закрывается.
- Подключение к источнику данных закрывается, если оно было открыто объектом `DataReader`, в противном случае оно остается открытым.

При выполнении одной команды по отношению к источнику данных (наш с вами случай) обычно проще и эффективнее создавать объекты `Command` и `Connection` неявно при создании объекта `DataAdapter` (именно так мы с вами и поступили). Однако при выполнении нескольких команд по отношению к одному источнику данных эффективнее создавать объект `Connection` явно и затем присвоить его объекту `DataAdapter`. Это позволяет поддерживать подключение постоянно открытым, без часто повторяющихся операций его открытия и закрытия, что снижает производительность. Эквивалентный код представлен ниже.

```
Private Sub ReadData1()
Dim rows As Integer
Dim daDep As New SqlDataAdapter
Dim cnn As New SqlConnection("server=Persist Security
Info=False; Integrated Security=SSPI;
database=EmployeeInfo; server=abrzh")
Dim cmdSelect As New SqlCommand("select * from
Departments")
dsE = New DataSet
cmdSelect.Connection = cnn
daDep.SelectCommand = cmdSelect
cnn.Open()
rows = daDep.Fill(dsE, "Departments")
DisplayDataset(dsE)
cnn.Close()
End Sub
```

Конечно, для эффективного использования явно созданных объектов `cnn` и `cmdSelect` желательно, чтобы количество операций с базой данных было достаточно большим.

Еще одно дополнение к написанному коду. Методу `Fill` мы передаем ссылку на набор данных `dsE` и имя таблицы `Departments`, в которую вставляются данные:

```
rows = daDep.Fill(dsE, "Departments")
```

Возможны другие варианты вызова метода Fill. Вместо имени таблицы можно было бы передать ссылку на объект DataTable или передать только ссылку на объект DataSet, а метод Fill в таком случае по умолчанию загрузит данные в объект DataTable по имени Table.

```
rows = daDep.Fill(dsE)
```

Для указания информативных имен таблиц Table, Table1 и так далее, используется коллекция объектов DataTableMapping и метод Add:

```
daDep.TableMappings.Add("Table", "Отделы")  
daDep.TableMappings.Add("Table1", "Сотрудники")
```

**Форма отчета:** отчет, защита работы.

### Практическое занятие № 16

**Цель:** Научиться управлять процессом передачи данных от источника данных к DataSet.

#### Задание 1:

Добавьте в процедуру ReadData() описание еще одного адаптера с другой командой Select для загрузки данных из таблицы Employees. Скомпонуйте проект, щелкните по кнопке DataAdapterFill, в поле со списком, как и прежде, будет отображена информация о содержании набора данных dsE, но теперь она извлекается из базы данных под управлением SQL Server, а не генерируется локально кодом приложения (рис. 3.3).

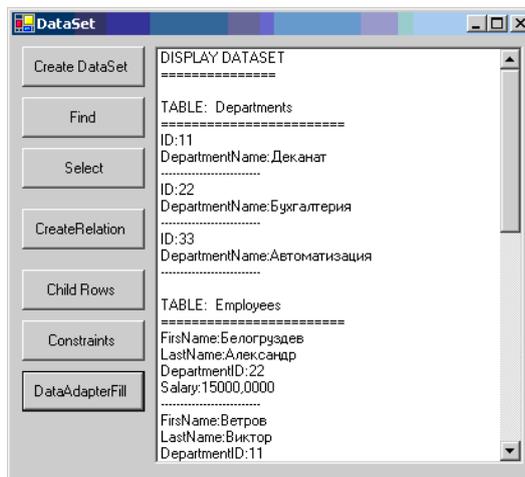


Рис. 3.3. Данные из таблиц базы данных, управляемой SQL Server (не из локальной версии, как ранее)

Для установления родительско-дочерних связей между записями в этих таблицах можно создать объект DataRelation, который служит отношением между ними.

```
dsE.Relations.Add("D_E",  
dsE.Tables("Departments").Columns("ID"),  
dsE.Tables("Employees").Columns("DepartmentID"))
```

**Форма отчета:** отчет, защита работы.

### Практическое занятие № 17

**Цель:** отработать навыки фильтрации и поиска данных

#### Задание 1:

Исходным файлом является DVD.xls, содержащий информацию о DVD-коллекции видеофильмов (названии фильма на русском и английском языках, годе создания фильма, жанре, актерам и режиссере).

*Технические требования:*

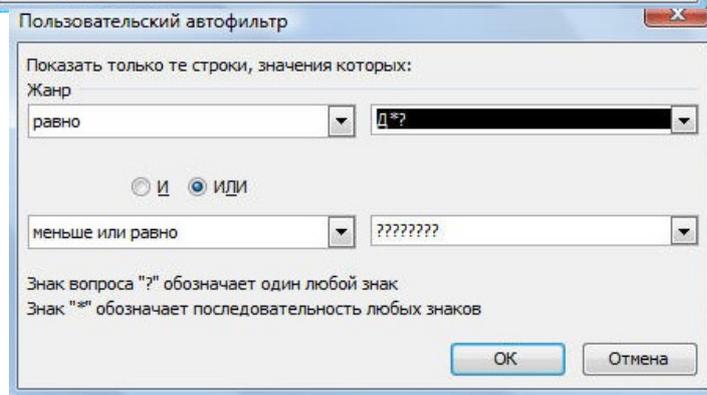
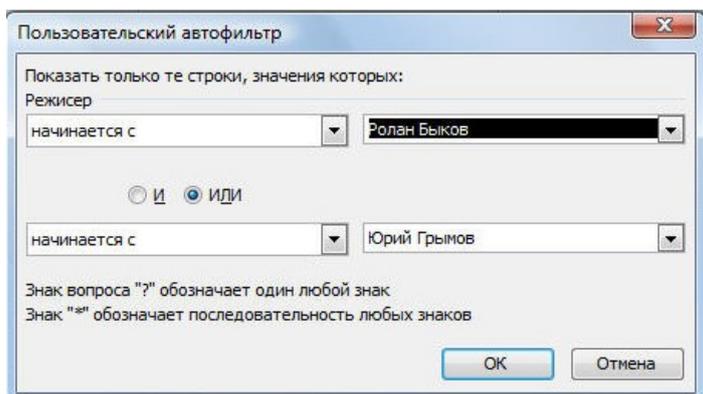
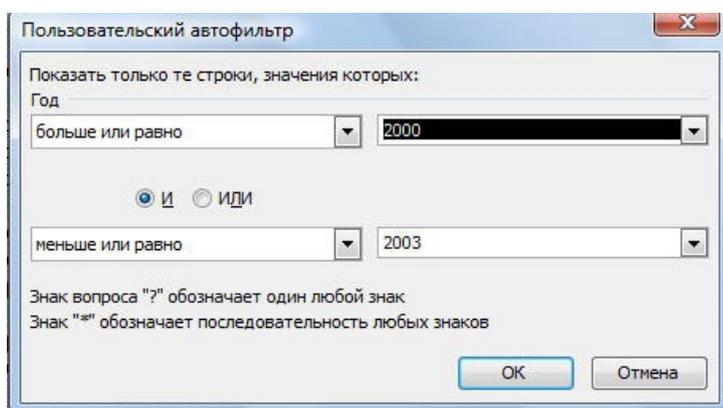
1. Каждое задание выполнять на новом листе рабочей книги с именем, указанным в поле «Имя нового листа».
2. Фильм будем считать отечественным, если название на английском языке является пустым.
3. Каждая запись списка должна соответствовать основным типам полей (числовое, текстовое, денежный...).
4. ДОБАВИТЬ запись «цена» и ввести значения.
5. Сортировать по возрастанию список   
произвести сортировку по указанному полю: Номер диска, Актеры, Режиссер.
6. Сформировать и реализовать следующие запросы:

- простой (автофильтр) *Данные – фильтр – автофильтр – самостоятельно определить критерии.*

-расширенный  
(пользовательский)

При выборе “по году” допустимо 4 варианта: все, только указанного года, в период между двумя указанными годами, ранее или позднее указанного года.

Выборка “Режиссеры”.



7. Фильтр с использованием шаблона, например Детектив-драма.
8. Найти максимальную и минимальную цену фильма.
9. Сформировать Промежуточные итоги (Данные-итоги)- самостоятельно определить критерии.

**Форма отчета:** отчет, защита работы.

### Практическое занятие № 18

**Цель:** отработать навыки работы DataSet со строгим контролем.

**Задание 1:**

Создайте объект DataSet в режиме создания компонентов (панель Data) для модели Dogovor. Проверьте идентичность результатов создания таблиц, установления связей, заполнения таблиц для случая программного кодирования

объекта DataSet и для случая визуального программирования. Воспользуйтесь разработанными процедурами отображения дочерних записей связанных таблиц.

**Форма отчета:** отчет, защита работы.

### **Практическое занятие № 19**

**Цель:** отработать навыки обновления данных

#### **Задание 1:**

Примените процедуру просмотра имеющихся ограничений для таблиц объекта DataSet Dogovor. Убедитесь, что все уникальные поля таблиц и установленные вами связи между таблицами реализованы в объектах UniqueConstraint и ForeignKeyConstraint.

**Форма отчета:** отчет, защита работы.

### **Практическое занятие № 20**

**Цель:** отработать навыки автоматизации приложения

#### **Задание 1:**

Ознакомьтесь с материалом

### **Разработка эскизного проекта**

Эскизный проект возникает как результат анализа требований, предъявленных к программному продукту. В нем в общем виде формулируются указания по созданию программного продукта. Здесь ставится задача для каждого разработчика, описываются алгоритм решения задачи, способы взаимодействия создаваемого продукта с другими программами и устройствами ввода-вывода, выбираются структуры данных, определяются способы хранения данных на диске или в базе данных.

Эскизный проект не может быть слишком большим. Он должен быть обозримым, схематичным, четко показывающим основные этапы создания программного продукта. Обычно эскизный проект содержит не больше 5—6 страниц текста. К нему прилагаются диаграммы, рисунки и чертежи, а также календарный план выполнения проекта.

После того как эскизный проект создан, он раздается всем участникам разработки для изучения и обсуждения. Каждый разработчик обдумывает свой участок проекта, вносит свои предложения и дополнения, конкретизирует план выполнения проекта.

### **Разработка технического проекта**

После изучения эскизного проекта всеми заинтересованными лицами наступает время создания технического проекта. В его обсуждении принимает участие вся команда разработчиков под руководством менеджера проекта. Каждый разработчик вносит свои предложения по реализации и улучшению проекта, уточняет и детализирует относящиеся к нему положения проекта, согласует интерфейсы с другими разработчиками.

Технический проект будет рабочим документом на все время реализации проекта, поэтому он должен быть понятен и приемлем для всех программистов. В нем не должно быть недомолвок, двусмысленностей, не должно оставаться пробелов и недоговоренностей.

При разработке технического проекта окончательно определяется конфигурация технических средств, и вся дальнейшая работа ведется с учетом этой конфигурации. Уточняется операционная среда, в которой будет функционировать программный продукт, и системное программное обеспечение. Например, Web-приложение работает в браузере. Браузеры по-разному интерпретируют языки HTML и JavaScript, поэтому надо сразу решить, будет ли программный продукт рассчитан на определенный браузер или он должен работать в любом. В первом случае разработчики могут включить в продукт дополнительные возможности языков HTML и JavaScript, интерпретируемые данным браузером, во втором — должны использовать только стандартные конструкции, что может значительно затруднить разработку.

в техническом проекте уточняются типы и структуры исходных и промежуточных данных, полностью детализируется алгоритм решения задачи. Задача разбивается на модули, которые распределяются среди программистов.

При объектно-ориентированном проектировании в техническом проекте определяются все объекты, необходимые для осуществления проекта и выявляются связи между ними. Полностью выписывается строение каждого объекта, его поля и методы. Объекты записываются в виде интерфейсов или абстрактных классов, дальнейшая разработка которых поручается конкретным программистам.

После проработки технического проекта каждым участником разработки собираются и обобщаются их уточнения и замечания. Окончательная версия проекта обсуждается командой разработчиков. Менеджер проекта выносит технический проект на утверждение руководством фирмы-разработчика и заказчиком программного продукта. После этого технический проект становится рабочим проектом для группы разработчиков.

### **Рабочий проект**

После утверждения технического проекта он становится основным рабочим документом для команды разработчиков программного продукта. Рабочий проект — это большой, подробный документ, наиболее полно описывающий будущий программный продукт и план его создания. В нем содержатся детальные указания каждому разработчику и команде в целом, определена структура базы данных и других хранилищ данных, которой будут руководствоваться все разработчики. Короче говоря, в рабочем проекте должны содержаться все сведения, нужные каждому разработчику и команде в целом. В частности, в нем должны быть записаны этапы и сроки разработки, чтобы каждый программист твердо знал их.

При объектно-ориентированном проектировании в рабочем проекте должны быть полностью описаны все классы и связи между ними. Это описание можно сделать в виде абстрактных классов или интерфейсов, на языке разработки или на языке описания. Важно, чтобы все участники проекта правильно понимали эту запись и одинаково интерпретировали ее.

Каждому участнику проекта выдается экземпляр рабочего проекта. При всяком изменении рабочего проекта участники получают его новую версию. В настоящее время с развитием Web-технологии, как правило, создается собственный сайт для каждого проекта. Все рабочие документы публикуются на этом сайте, а при каждом их изменении участники проекта получают уведомление по электронной почте.

### **Задание 2:**

1. Разработайте проект автоматизации библиотечного каталога.
2. Проведите анализ работы деканата и разработайте проект его автоматизации.
3. Проанализируйте информационные потоки вашего факультета и спроектируйте компьютерную систему их обработки.

**Форма отчета:** отчет, защита работы.

### **Практическое занятие № 21**

**Цель:** Получение навыков разработки юнит-тестов и проведения тестирования программного обеспечения

### **Задание 1:**

1. Откройте выбранную IDE и создайте проект на основе существующих программных кодов, реализующих алгоритм пирамидальной сортировки.
2. Подключите к проекту библиотеку JUnit.
3. Создайте каркас для юнит-тестов (Например, в IDE Eclipse можно выбрать нужный класс, открыть контекстное меню, и выбрать New->JUnit test case, в появившемся диалоговом окне выбрать методы, для которых понадобятся юнит-тесты).
4. Создайте юнит-тест согласно описанным требованиям.
5. Отладьте и запустите юнит-тест.
6. Оцените результаты выполнения юнит-тестирования и сделайте соответствующие выводы.

**Форма отчета:** отчет, защита работы

### **Практическое занятие № 22**

**Цель:** отработать навыки администрирования приложения

### **Задание 1:**

1. Описать этапы проектирования модулей программы.
2. Составить в виде блок-схемы алгоритм решения задачи.
3. Составить отчет по практической работе.

**Отчет по практической работе должен включать:**

1. Алгоритм решения задачи.
2. Набор тестов для отладки программы.

**Задача.** Составить алгоритм решения задачи, приведенной ниже, с использованием структурных единиц: процедур и/или функций.

### **Варианты индивидуальных заданий.**

1. Даны два двумерных массива вещественных элементов. Размер исходных массивов не превосходит  $10 \times 10$  элементов. Для каждого из массивов указать номера столбцов, содержащих только положительные элементы. Если таковых столбцов в массиве нет, то вывести соответствующее сообщение. Проверку столбца на положительность элементов оформить в виде процедуры с передачей в нее всех элементов текущего столбца.
2. Даны два двумерных массива натуральных элементов. Размер исходных массивов не превосходит  $10 \times 10$  элементов. Для каждого из массивов указать номера столбцов, содержащих только кратные 5 или 7 элементы. Если таких столбцов в массиве нет, то вывести соответствующее сообщение. Проверку столбца на наличие указанных элементов оформить в виде процедуры с передачей в нее всех элементов текущего столбца.
3. Даны пять одномерных массива вещественных элементов. Размер каждого массива не превосходит 100 элементов. Для каждого из массивов определить, составляют ли его элементы знакопередающуюся последовательность. Если да, то указать порядковый номер такого массива, в противном случае вывести отрицательный ответ. Проверку массива на выполнение условия оформить в виде процедуры с передачей в нее всех элементов рассматриваемого массива.
4. Даны два двумерных массива символьных (буквы русского алфавита) элементов. Размер исходных массивов не превосходит  $10 \times 10$  элементов. Для каждого из массивов указать номера строк, содержащих элементы только строчных букв, если таких строк нет ни для какого массива, то вывести соответствующее сообщение. Проверку строки на наличие указанных элементов оформить в виде процедуры с передачей в нее всех элементов текущей строки.
5. Даны два двумерных массива вещественных элементов. Размер исходных массивов не превосходит  $10 \times 10$  элементов. Для каждого из массивов указать количество столбцов, содержащих только не положительные элементы. Если таких столбцов нет ни для одного из массивов, то вывести соответствующее сообщение. Проверку столбца на наличие указанных элементов оформить в виде процедуры с передачей в нее всех элементов текущего столбца.
6. Даны пять одномерных массива вещественных элементов. Размер каждого массива не превосходит 100 элементов. Для каждого из массивов определить, составляют ли его элементы одного знака. Если да, то указать порядковый номер такого массива, в противном случае вывести отрицательный ответ. Проверку массива на выполнение условия оформить в виде процедуры с передачей в нее всех элементов рассматриваемого массива.
7. Даны два двумерных массива целочисленных элементов. Размер исходных массивов не превосходит  $10 \times 10$  элементов. Для каждого из массивов указать количество строк, содержащих элементы, четность которых чередуется, а вторым в четных строках является нечетный элемент. Если таких строк нет ни для одного из массивов, то вывести соответствующее сообщение. Проверку строки на наличие указанных элементов оформить в виде процедуры с передачей в нее всех элементов текущего столбца.
8. Даны пять одномерных массива символьных (только латинские буквы) элементов. Размер каждого массива не превосходит 100 элементов. Для каждого из массивов определить, чередуются ли в нем буквы строчные и прописные. Если да, то указать

- порядковый номер такого массива, в противном случае вывести отрицательный ответ. Проверку массива на выполнение условия оформить в виде процедуры с передачей в нее всех элементов рассматриваемого массива.
9. Даны два двумерных массива целочисленных элементов. Размер исходных массивов не превосходит  $10 \times 10$  элементов. Для каждого из массивов указать количество строк, для которых сумма элементов, стоящих на нечетных местах в строке, является положительным числом. Если таких строк нет ни для одного из массивов, то вывести соответствующее сообщение. Проверку строки на выполнение условия и расчет оформить в виде процедуры с передачей в нее всех элементов текущей строки.
  10. Даны два двумерных массива вещественных элементов. Размер исходных массивов не превосходит  $10 \times 10$  элементов. Для каждого из массивов указать номера столбцов, произведение отрицательных элементов которых является положительным числом. Если таких столбцов нет ни для одного из массивов, то вывести соответствующее сообщение. Проверку столбца на выполнение условия и расчет оформить в виде процедуры с передачей в нее всех элементов текущего столбца.
  11. Даны пять одномерных массива символьных (только латинские буквы) элементов. Размер каждого массива не превосходит 100 элементов. Для каждого из массивов определить, расположены ли в нем строчные буквы в алфавитном порядке. Если да, то указать порядковый номер такого массива, в противном случае вывести отрицательный ответ. Проверку массива на выполнение условия оформить в виде процедуры с передачей в нее всех элементов рассматриваемого массива.
  12. Даны два двумерных массива целочисленных элементов. Размер исходных массивов не превосходит  $10 \times 10$  элементов. Для каждого из массивов проверить выполнение условия: все четные строки массива таковы, что суммы их элементов образуют возрастающую последовательность. Вывести соответствующее сообщение. Вычисление суммы элементов массива и проверку последовательности чисел на выполнение условия оформить в виде процедуры с передачей в нее всех необходимых элементов.
  13. Даны два двумерных массива вещественных элементов. Размер исходных массивов не превосходит  $10 \times 10$  элементов. Преобразовать все нечетные строки каждого массива так, чтобы элементы составляли возрастающую по абсолютной величине последовательность. Вывести преобразованные массивы. Упорядочивание элементов оформить в виде процедуры с передачей в нее всех необходимых элементов.
  14. Даны два двумерных массива целочисленных элементов. Размер исходных массивов не превосходит  $10 \times 10$  элементов. Для каждого столбца массивов вычислить суммы и количества элементов, значения которых находятся в заданном диапазоне. Если чисел, удовлетворяющих этому условию нет, то вывести соответствующее сообщение. Вычисление для элементов столбца массива оформить в виде процедуры с передачей в нее всех необходимых элементов.
  15. Даны пять одномерных массива символьных (только латинские буквы) элементов. Размер каждого массива не превосходит 100 элементов. Преобразовать все массивы так, чтобы все строчные буквы были расположены по алфавиту. При этом переставлять только строчные буквы, оставив прописные буквы на своих местах. Преобразование каждого массива оформить в виде процедуры с передачей в нее всех необходимых элементов. Если перестановка элементов не потребовалась, то есть исходные массивы удовлетворяют требуемому условию, то вывести соответствующее сообщение.  
удовлетворяют требуемому условию, то вывести соответствующее сообщение.

#### **Задание.**

1. Разработать модули программы, спроектированные во время практического занятия
2. Отладить программу с использованием тестов, составленных во время практического занятия

**Форма отчета:** отчет, защита работы.

### **Практическое занятие № 23**

**Цель:** изучить основные подходы к проектированию тестов

**Задание 1:** ознакомиться с материалом

Рассмотрим два основных подхода к проектированию тестов.

Первый подход ориентируется только на стратегию тестирования, называемую стратегией "черного ящика", тестированием с управлением по данным или тестированием с управлением по входу-выходу. При использовании этой стратегии программа рассматривается как черный ящик. Тестовые данные используются только в соответствии со спецификацией программы (т. е. без учета знаний о ее внутренней структуре). Недостижимый идеал сторонника первого подхода — проверить все возможные комбинации и значения на входе. Обычно их слишком много даже для простейших алгоритмов. Так, для программы расчета среднего арифметического четырех чисел надо готовить  $10^7$  тестовых данных.

При первом подходе обнаружение всех ошибок в программе является критерием исчерпывающего входного тестирования. Последнее может быть достигнуто, если в качестве тестовых наборов использовать все возможные наборы входных данных. Следовательно, приходим к выводу, что для исчерпывающего тестирования программы требуется бесконечное число тестов, а значит, построение исчерпывающего входного теста невозможно. Это подтверждается двумя аргументами: во-первых, нельзя создать тест, гарантирующий отсутствие ошибок; во-вторых, разработка таких тестов противоречит экономическим требованиям. Поскольку исчерпывающее тестирование исключается, нашей целью должна стать максимизация результативности капиталовложений в тестирование (максимизация числа ошибок, обнаруживаемых одним тестом). Для этого необходимо рассматривать внутреннюю структуру программы и делать некоторые разумные, но, конечно, не обладающие полной гарантией достоверности предположения.

Второй подход использует стратегию "белого ящика", или стратегию тестирования, управляемую логикой программы, которая позволяет исследовать внутреннюю структуру программы. В этом случае тестирующий получает тестовые данные путем анализа только логики программы; стремится, чтобы каждая команда была выполнена хотя бы один раз. При достаточной квалификации добивается, чтобы каждая команда условного перехода выполнялась бы в каждом направлении хотя бы один раз. Цикл должен выполняться один раз, ни разу, максимальное число раз. Цель тестирования всех путей извне также недостижима. В программе из двух последовательных циклов внутри каждого из них включено ветвление на десять путей, имеется  $10^{18}$  путей расчета. Причем выполнение всех путей расчета не гарантирует выполнения всех спецификаций.

Сравним способ построения тестов при данной стратегии с исчерпывающим входным тестированием стратегии "черного ящика". Неверно предположение, что достаточно построить такой набор тестов, в котором каждый оператор исполняется хотя бы один раз. Исчерпывающему входному тестированию может быть поставлено в соответствие исчерпывающее тестирование маршрутов. Подразумевается, что программа проверена полностью, если с помощью тестов удастся осуществить выполнение этой программы по всем возможным маршрутам ее потока (графа) передач управления.

Последнее утверждение имеет два слабых пункта: во-первых, число не повторяющих друг друга маршрутов — астрономическое; во-вторых, даже если каждый маршрут может быть проверен, сама программа может содержать ошибки (например, некоторые маршруты пропущены).

Свойство пути выполняться правильно для одних данных и неправильно для других — называемое чувствительностью к данным, наиболее часто проявляется за счет численных погрешностей и погрешностей усечения методов. Тестирование каждого из всех маршрутов одним тестом не гарантирует выявления чувствительности к данным.

В результате всех изложенных выше замечаний отметим, что ни исчерпывающее входное тестирование, ни исчерпывающее тестирование маршрутов не могут стать полезными стратегиями, потому что оба они нереализуемы. Поэтому реальным путем, который позволит создать хорошую, но, конечно, не абсолютную стратегию, является сочетание тестирования программы несколькими методами.

Рассмотрим пример тестирования оператора:

*if A and B then...*

при использовании разных критериев полноты тестирования.

При критерии покрытия условий требовались бы два теста:  $A = true, B = false$  и  $A = false, B = true$ . Но в этом случае не выполняется then-предложение оператора if.

Существует еще один критерий, названный покрытием решений/условий. Он требует такого достаточного набора тестов, чтобы все возможные результаты каждого условия в решении выполнялись, по крайней мере, один раз; все результаты каждого решения выполнялись тоже один раз и каждой точке входа передавалось управление, по крайней мере, один раз.

Недостатком критерия покрытия решений/условий является невозможность его применения для выполнения всех результатов всех условий. Часто подобное выполнение имеет место вследствие того, что определенные условия скрыты другими условиями. Например, если условие AND есть ложь, то никакое из последующих условий в выражении не будет выполнено. Аналогично, если условие OR есть истина, то никакое из последующих условий не будет выполнено. Следовательно, критерии покрытия условий и покрытия решений/условий недостаточно чувствительны к ошибкам в логических выражениях.

Критерием, который решает эти и некоторые другие проблемы, является комбинаторное покрытие условий. Он требует создания такого числа тестов, чтобы все возможные комбинации результатов условия в каждом решении и все точки входа выполнялись, по крайней мере, один раз.

В случае циклов число тестов для удовлетворения критерию комбинаторного покрытия условий обычно больше, чем число путей.

Легко видеть, что набор тестов, удовлетворяющий критерию комбинаторного покрытия условий, удовлетворяет также и критериям покрытия решений, покрытия условий и покрытия решений/условий.

Таким образом, для программ, содержащих только одно условие на каждое решение, минимальным является критерий, набор тестов которого вызывает выполнение всех результатов каждого решения, по крайней мере, один раз; передает управление каждой точке входа (например, оператор CASE).

Для программ, содержащих решения, каждое из которых имеет более одного условия, минимальный критерий состоит из набора тестов, вызывающих все возможные комбинации результатов условий в каждом решении и передающих управление каждой точке входа программы, по крайней мере, один раз.

Деление алгоритма на типовые стандартные структуры позволяет минимизировать усилия программиста, затрачиваемые им на тестирование. Запрет на вложенные структуры как раз и объясняется излишними затратами на тестирование. Использование цепочки простых альтернатив с одним действием или структуры ВЫБОР вместо вложенных простых АЛЬТЕРНАТИВ значительно сокращает число тестов!

## Задание 2:

1. Оформить внешнюю спецификацию.
2. Составить в виде блок-схемы алгоритм решения задачи.
3. Создать программу решения задачи на любом алгоритмическом языке программирования.
4. Составить набор тестов и провести тестирование созданной программы с помощью методов «белого ящика» (покрытия операторов, покрытия решений, покрытия условий, комбинаторного покрытия условий).

**Задача. «Нахождение характерных точек функции».** Составить алгоритм и написать программу последовательного вычисления значений заданной функции  $Y(X)$  до тех пор, пока не будет пройдена некоторая характерная точка графика функции. Значения аргумента  $X$  составляют возрастающую последовательность с шагом  $h$ . Начальное значение  $X_0$  и шаг изменения аргумента  $h$  задаются пользователем.

### Варианты индивидуальных заданий.

	Вид	функции	$Y(X)$ ,		Вид	функции	$Y(X)$ ,
--	-----	---------	----------	--	-----	---------	----------

№ п/п	характерные точки	№ п/п	характерные точки
	Локальный минимум функции $x^2 + 0.5 - \sin(3 \cdot x)$	6	Точка (точки), в которой функция $\left  \frac{2x^3 - 3}{x^2 + 1} \right $ равна 10.
	Точка (точки), для которой $\sqrt{x^2 + 5} - 1 = 5$ .	7	Локальный минимум функции $3(x+4)^2 - \sin(x+15)$
	Пересечение графиков функций $\sqrt{x^2 + 1}(3+x)$ и $\frac{5}{x^2 - 7}$ .	8	Точка (точки), в которой функция $\frac{2x^2}{\sin(x-1)} - 1$ равна -500.
	Все нули функции $x^2 - \sin(3x)$	9	Локальный максимум функции $\frac{2x+15}{3+x^2}$
	Локальный минимум функции $\frac{5}{3x - x^2 - 3}$	0	Пересечение графиков функций $(x^2 + 1)\sin(3+x)$ и $\frac{x+5}{x^2 + 1}$ .
	Точка (точки), в которой функция $x^2 - e^x$ равна -10.	1	Локальный минимум функции $x^2 - 3x + 15$
	Пересечение графиков функций $100\sin(1+x)$ и $x^3 + 5$ .	2	Все нули функции $\sqrt{x^2 + 0.5} - \sin(3x)$ .
	Локальный минимум функции $\sqrt{e^x} - x$	3	Локальный максимум функции $200x - e^x$
	Все нули функции $\frac{x^2 - 15}{\ln x^2 + 3x - 7}$ .	4	Все нули функции $x - e^x \cos(x)$ .
0	Локальный максимум функции $\frac{\sqrt{2x^2 + 1}}{3 + x^2}$	5	Точка (точки), в которой функция $\frac{x^2 + 5}{x^2 + 7}$ равна 15.
1	Пересечение графиков функций $10\cos(x)$ и $x^3/3 + 5$ .	6	Все нули функции $\frac{x^2 - 15}{2x^2 + 3x - 7}$ .
2	Локальный максимум функции $-(x-2)^2 + \cos(x)$	7	Пересечение графиков функций $3x^2 - 15$ и $10\cos(x) + 7$ .
3	Точка (точки), в которой функция $3x^2 - \frac{12x - 5}{x^2 + 1}$ равна 5.	8	Локальный минимум функции $-100x + e^x$
4	Все нули функции $\frac{x+5}{x^2 - 7}$ .	9	Локальный максимум функции $-\frac{x}{x^2 + 3}$
5	Локальный максимум функции $-e^{-x} - 100x$	0	Пересечение графиков функций $4(x-5)^2 - 15$ и $\frac{100}{x^2 + 2} - 7$ .

**Форма отчета:** отчет, защита работы.

### **Практическое занятие № 24**

**Цель:** получение практических навыков обработки исключений

#### **Задание 1:**

1. Помогите тестировщику определить возможные исключительные ситуации вашей ИС, определить экспериментально, ошибки каких классов могут быть сгенерированы в вашем приложении

2. Создать обработчики исключительных ситуаций с использованием выявленных классов и всех секций конструкции обработчика с соответствующими сообщениями, позволяющими корректно выполнить программу. Обработка исключительных ситуаций рассмотрена на примере Java. Вы пишете обработчики на выбранном языке программирования для Вашей системы.

#### **Задание 2:**

Ответьте на вопросы:

Что представляет собой исключение?

2. На какие части исключения позволяют разделить вычислительный процесс? Достоинства такого подхода?

3. Какой оператор используется для генерации исключительной ситуации?

4. Что представляет собой контролируемый блок? Для чего он нужен?

5. Что представляет собой секция-ловушка? Для чего она нужна?

6. Какие формы может иметь спецификация исключения в секции ловушке? В каких ситуациях используются эти формы?

7. Какой стандартный класс можно использовать для создания собственной иерархии исключений?

8. Каким образом можно создать собственную иерархию исключений?

9. В какой части программы может генерироваться исключение?

**Форма отчета:** отчет, защита работы.

### **Практическое занятие № 25**

**Цель:** получение практических навыков применения отладочных классов в проекте

#### **Задание 1:**

1. Создать основной класс. Создать 3 класса наследника. Создать в основном классе 5 методов и 3 свойства, связанных с его назначением, отличных от методов и свойств других подгрупп.

2. Придумать и создать, как минимум, по 2 метода в каждом классе-наследнике (всего не меньше 6) и хотя бы по 2 свойства (всего не меньше 6). Методы и свойства должны быть связаны с особенностями класса-наследника.

3. Сделать, как минимум, 3 метода основного класса переопределёнными в классах наследниках. Сделать, как минимум, 1 метод, который нельзя переопределить.

4. Скомпилировать программу, демонстрирующую работу каждого из методов каждого из классов.

#### Варианты на лабораторную работу

В ходе лабораторной работы нужно разработать программу.

1,11,21. Основной класс «Автомобили», 3 наследника «Грузовики», «Легковые» и «Автобусы».

2,12,22. Основной класс «Одежда», 3 наследника «Куртки», «Шубы» и «Пуховики».

3,13,23. Основной класс «Еда», 3 наследника «Супы», «Закуски» и «Напитки».

4,14,24. Основной класс «Гаджеты», 3 наследника «Планшеты», «Смартфоны» и «Нетбуки».

5,15,25. Основной класс «Овощи», 3 наследника «Морковь», «Свёкла» и «Лук».

6,16,26. Основной класс «Песни», 3 наследника «Частушки», «Баллады» и «Романсы».

7,17,27.

Основной класс «Оружие», 3 наследника «Пистолеты», «Автоматы» и «Пулеметы».

8,18,28.

Основной класс «Документы», 3 наследника «Справки», «Приказы» и «Заявления».

9,19,29. Основной класс «Погода», 3 наследника «Пасмурно», «Ясно» и «Ураган».

10,20,30. Основной класс «Звери», 3 наследника «Зайцы», «Волки» и «Лисы».

**Форма отчета:** отчет, защита работы

### **Практическое занятие № 26**

**Цель:** получение практических навыков тестирования и отладки программ

**Задание 1:** ознакомиться с материалом

**Теоретические основы.** Тестирование – процесс выполнения программы на наборе тестов с целью выявления ошибок.

Локализацией называют процесс определения оператора программы, выполнение которого вызвало нарушение нормального вычислительного процесса. Для исправления ошибки необходимо определить ее причину, т.е. определить оператор или фрагмент, содержащие ошибку. Причины ошибок могут быть как очевидны, так и очень глубоко скрыты. В целом сложность отладки обусловлена следующими причинами:

- требует от программиста глубоких знаний специфики управления используемыми техническими средствами, операционной системы, среды и языка программирования, реализуемых процессов, природы и специфики различных ошибок, методик отладки и соответствующих программных средств;
- психологически дискомфортна, так как необходимо искать собственные ошибки и, как правило, в условиях ограниченного времени;
- возможно взаимовлияние ошибок в разных частях программы, например, за счет затирания области памяти одного модуля другим из-за ошибок адресации;
- отсутствуют четко сформулированные методики отладки.

Отладка программы в любом случае предполагает обдумывание и логическое осмысление всей имеющейся информации об ошибке. Большинство ошибок можно обнаружить по косвенным признакам посредством тщательного анализа текстов программ и результатов тестирования без получения дополнительной информации. При этом используют различные методы:

- ручного тестирования;
- индукции;
- дедукции;
- обратного прослеживания.

#### **Метод ручного тестирования**

Это - самый простой и естественный способ данной группы. При обнаружении ошибки необходимо выполнить тестируемую программу вручную, используя тестовый набор, при работе с которыми была обнаружена ошибка. Метод очень эффективен, но не применим для больших программ, программ со сложными вычислениями и в тех случаях, когда ошибка связана с неверным представлением программиста о выполнении некоторых операций. Данный метод часто используют как составную часть других методов отладки.

### **Метод индукции**

Метод основан на тщательном анализе симптомов ошибки, которые могут проявляться как неверные результаты вычислений или как сообщение об ошибке. Если компьютер просто "зависает", то фрагмент проявления ошибки вычисляют, исходя из последних полученных результатов и действий пользователя. Полученную таким образом информацию организуют и тщательно изучают, просматривая соответствующий фрагмент программы. В результате этих действий выдвигают гипотезы об ошибках, каждую из которых проверяют. Если гипотеза верна, то детализируют информацию об ошибке, иначе - выдвигают другую гипотезу. Последовательность выполнения отладки методом индукции показана на рисунке в виде схемы алгоритма.

Самый ответственный этап - выявление симптомов ошибки. Организуя данные об ошибке, целесообразно записать все, что известно о её проявлениях, причем фиксируют, как ситуации, в которых фрагмент с ошибкой выполняется нормально, так и ситуации, в которых ошибка проявляется. Если в результате изучения данных никаких гипотез не появляется, то необходима дополнительная информация об ошибке. Дополнительную информацию можно получить, например, в результате выполнения схожих тестов. В процессе доказательства пытаются выяснить, все ли проявления ошибки объясняет данная гипотеза, если не все, то либо гипотеза не верна, либо ошибок несколько.

### **Метод дедукции**

По методу дедукции вначале формируют множество причин, которые могли бы вызвать данное проявление ошибки. Затем анализируя причины, исключают те, которые противоречат имеющимся данным. Если все причины исключены, то следует выполнить дополнительное тестирование исследуемого фрагмента. В противном случае наиболее вероятную гипотезу

пытаются доказать. Если гипотеза объясняет полученные признаки ошибки, то ошибка найдена, иначе - проверяют следующую причину.

### **Метод обратного прослеживания**

Для небольших программ эффективно применение метода обратного прослеживания. Начинают с точки вывода неправильного результата. Для этой точки строится гипотеза о значениях основных переменных, которые могли бы привести к получению имеющегося результата. Далее, исходя из этой гипотезы, делают предложения о значениях переменных в предыдущей точке. Процесс продолжают, пока не обнаружат причину ошибки.

## **Задание 2:**

### **Задание.**

1. Составить в виде блок-схемы алгоритм решения задачи.
2. Создать программу решения задачи на любом алгоритмическом языке программирования.
3. Отладить программу.

**Задача:** составить список учебной группы, включающей 25 человек. Для каждого учащегося указать дату рождения, год поступления в колледж, курс, группу, оценки каждого года обучения.

Назначение задачи: получить значение определённого критерия и упорядочить список студентов по нему.

Достижимая цель: упорядочить список студентов по среднему баллу и получить его.

**Форма отчета:** отчет, защита работы.

## **Практическое занятие № 27**

**Цель:** получение практических навыков разработки модулей программной системы и интеграции этих модулей

**Задание 1:** ознакомиться с материалом

**Теоретические сведения** Термин «интеграция» относится к такой операции в процессе разработки ПО, при которой вы объединяете отдельные программные компоненты в единую систему. В небольших проектах интеграция может занять одно утро и заключаться в объединении горстки классов. В больших — могут потребоваться недели или месяцы, чтобы связать воедино весь набор программ. Независимо от размера задач в них применяются одни и те же принципы.

Тема интеграции тесно переплетается с вопросом последовательности конструирования. Порядок, в котором вы создаете классы или компоненты, влияет на порядок их интеграции: вы не можете интегрировать то, что еще не было создано. Последовательности интеграции и конструирования имеют большое значение.

Поскольку интеграция выполняется после того, как разработчик завершил модульное тестирование, и одновременно с системным тестированием, ее иногда считают операцией, относящейся к тестированию. Однако она достаточно сложна, и поэтому ее следует рассматривать как независимый вид деятельности.

Аккуратная интеграция обеспечивает:

- упрощенную диагностику дефектов;
- меньшее число ошибок;
- меньшее количество «лесов»;
- раннее создание первой работающей версии продукта;
- уменьшение общего времени разработки;
- лучшие отношения с заказчиком;
- улучшение морального климата;
- увеличение шансов завершения проекта;
- более надежные оценки графика проекта;
- более аккуратные отчеты о состоянии;
- лучшее качество кода;
- меньшее количество документации.

Интеграция программ выполняется посредством *поэтапного* или *инкрементного* подхода.

**Поэтапная интеграция** состоит из этапов, перечисленных ниже:

1. «Модульная разработка»: проектирование, кодирование, тестирование и отладка каждого класса.
2. «Системная интеграция»: объединение классов в одну огромную систему.
3. «Системная дезинтеграция»: тестирование и отладка всей системы.

Проблема поэтапной интеграции в том, что, когда классы в системе впервые соединяются вместе, неизбежно возникают новые проблемы и их причины могут быть в чем угодно. Поскольку у вас масса классов, которые никогда раньше не работали вместе, виновником может быть плохо протестированный класс, ошибка в интерфейсе между двумя классами или ошибка, вызванная взаимодействием двух классов. Все классы находятся под подозрением.

Неопределенность местонахождения любой из проблем сочетается с тем фактом, что все эти проблемы вдруг проявляют себя одновременно. Это заставляет вас иметь дело не только с проблемами, вызванными взаимодействием классов, но и другими ошибками, которые трудно диагностировать, так как они взаимодействуют.

Поэтому поэтапную интеграцию называют еще «интеграцией большого взрыва»

Поэтапную интеграцию нельзя начинать до начала последних стадий проекта, когда будут разработаны и протестированы все классы. Когда классы, наконец, будут объединены и проявится большое число ошибок, программисты тут же ударятся в паническую отладку вместо методического определения и исправления ошибок.

Для небольших программ — нет, а для крошечных — поэтапная интеграция может быть наилучшим подходом. Если программа состоит из двух-трех классов, поэтапная интеграция может сэкономить ваше время, если вам повезет. Но в большинстве случаев инкрементный подход будет лучше.

При **инкрементной интеграции** вы пишете и тестируете маленькие участки программы, а затем комбинируете эти кусочки друг с другом по одному. При таком подходе — по одному элементу за раз — вы выполняете перечисленные далее действия:

1. Разрабатываете небольшую, функциональную часть системы. Это может быть наименьшая функциональная часть, самая сложная часть, основная часть или их комбинация. Тщательно тестируете и отлаживаете ее. Она послужит скелетом, на котором будут наращиваться мускулы, нервы и кожа, составляющие остальные части системы.
2. Проектируете, кодируете, тестируете и отлаживаете класс.
3. Прикрепляете новый класс к скелету. Тестируете и отлаживаете соединение скелета и нового класса. Убеждаетесь, что эта комбинация работает, прежде чем переходить к добавлению нового класса. Если дело сделано, повторяете процесс, начиная с п. 2.

Инкрементный подход имеет массу преимуществ перед традиционным поэтапным подходом независимо от того, какую инкрементную стратегию вы используете:

**Ошибки можно легко обнаружить** Когда во время инкрементной интеграции возникает новая проблема, то очевидно, что к этому причастен новый класс. Либо его интерфейс с остальной частью программы неправилен, либо его взаимодействие с ранее интегрированными классами приводит к ошибке. В любом случае вы точно знаете, где искать проблему.

**В таком проекте система раньше становится работоспособной** Когда код интегрирован и способен выполняться, даже если система еще не пригодна к использованию, это выглядит так, будто это скоро произойдет. При инкрементной интеграции программисты раньше видят результаты своей работы, поэтому их моральное состояние лучше, чем в том случае, когда они подозревают, что их проект может никогда не сделать первый вдох.

**Вы получаете улучшенный мониторинг состояния** При частой интеграции реализованная и нереализованная функциональность видна с первого взгляда. Менеджеры будут иметь лучшее представление о состоянии проекта, видя, что 50% системы уже работает, а не слыша, что кодирование «завершено на 99%».

**Вы улучшаете отношения с заказчиком** Если частая интеграция влияет на моральное состояние разработчиков, то она также оказывает влияние и на моральное состояние заказчика. Клиенты любят видеть признаки прогресса, а инкрементная интеграция предоставляет им такую возможность достаточно часто.

**Системные модули тестируются гораздо полнее** Интеграция начинается на ранних стадиях проекта. Вы интегрируете каждый класс по мере его готовности, а не ожидая одного внушительного мероприятия по интеграции в конце разработки. Программист тестирует классы в обоих случаях, но в качестве элемента общей системы они используются гораздо чаще при инкрементной, чем при поэтапной интеграции.

**Вы можете создать систему за более короткое время** Если интеграция тщательно спланирована, вы можете проектировать одну часть системы в то время, когда другая часть уже кодируется. Это не уменьшает общее число человеко-часов, требуемых для полного проектирования и кодирования, но позволяет выполнять часть работ параллельно, что является преимуществом в тех случаях, когда время имеет критическое значение.

При поэтапной интеграции вам не нужно планировать порядок создания компонентов проекта. Все компоненты интегрируются одновременно, поэтому вы можете разрабатывать их в любом порядке — главное, чтобы они все были готовы к часу X.

При инкрементной интеграции вы должны планировать более аккуратно. Большинство систем требует интеграции некоторых компонентов перед интеграцией других. Так что планирование интеграции влияет на планирование конструирования — порядок, в котором конструируются компоненты, должен обеспечивать порядок, в котором они будут интегрироваться.

### **Нисходящая интеграция**

При нисходящей интеграции класс на вершине иерархии пишется и интегрируется первым. Вершина иерархии — это главное окно, управляющий цикл приложения, объект, содержащий метод main() в программе на Java, функция WinMain() в программировании для Microsoft Windows или аналогичные. Для работы этого верхнего класса пишутся заглушки. Затем, по мере интеграции классов сверху вниз, классы заглушек заменяются реальными.

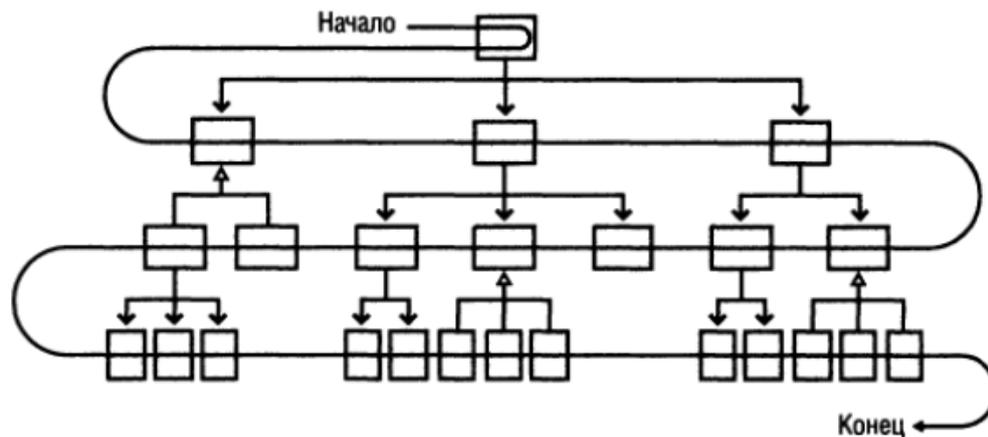
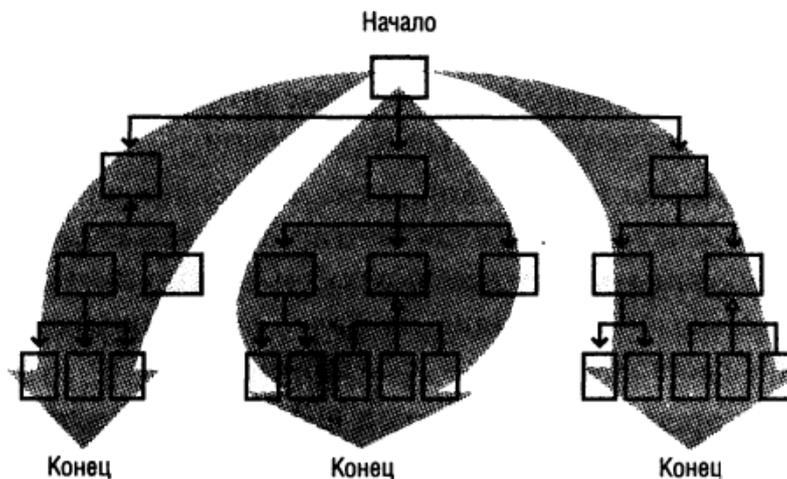


Рис.3 Нисходящая интеграция

При нисходящей интеграции вы создаете те классы, которые находятся на вершине иерархии, первыми, а те, что внизу, — последними.

Хорошей альтернативой нисходящей интеграции в чистом виде может стать подход с вертикальным секционированием.

Рис. 4 Вертикальное секционирование



При этом систему реализуют сверху вниз по частям, возможно, по очереди выделяя функциональные области и переходя от одной к другой.

#### Восходящая интеграция

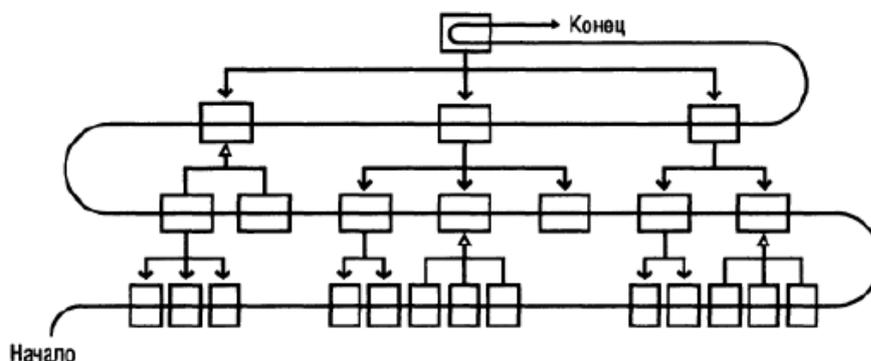


Рис.5 Восходящая интеграция

При восходящей интеграции вы пишете и интегрируете сначала классы, находящиеся в низу иерархии. Добавление низкоуровневых классов по одному, а не всех одновременно — вот что делает восходящую интеграцию инкрементной стратегией. Сначала вы пишете тестовые драйверы для выполнения низкоуровневых классов, а затем добавляете эти классы к тестовым драйверам, пристраивая их по мере готовности. Добавляя класс более высокого уровня, вы заменяете классы драйверов реальными.

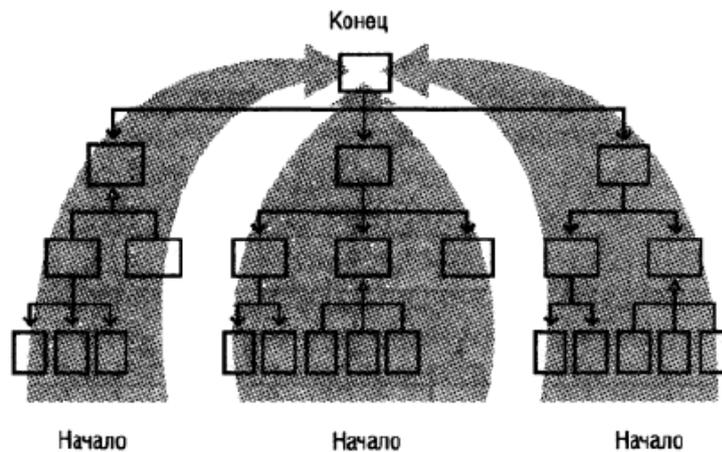


Рис. 6 Гибридный подход при восходящей интеграции

Как и нисходящую, восходящую интеграцию в чистом виде используют редко — вместо нее можно применять гибридный подход, реализующий секционную интеграцию.

#### Сэндвич-интеграция

Проблемы с нисходящей и восходящей интеграциями в чистом виде привели к тому, что некоторые эксперты стали рекомендовать сэндвич-подход.

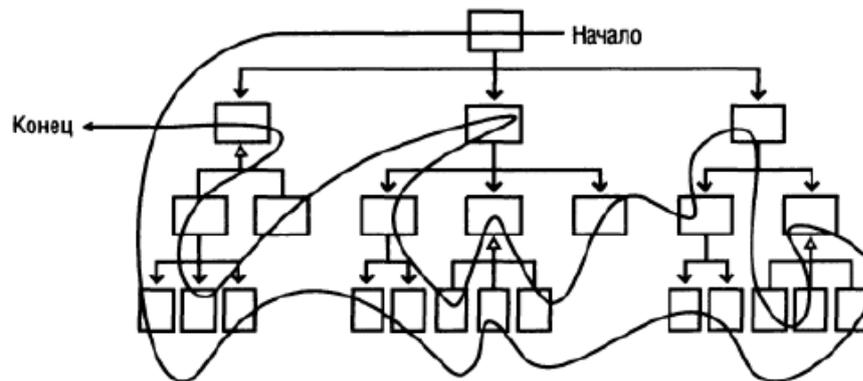


Рис. 7 Сэндвич-интеграция

Сначала вы объединяете высокоуровневые классы бизнес-объектов на вершине иерархии. Затем добавляете классы, взаимодействующие с аппаратной частью, и широко используемые вспомогательные классы в низу иерархии.

Напоследок вы оставляете классы среднего уровня.

#### Риск-ориентированная интеграция

Риск-ориентированную интеграцию, которую также называют «интеграцией, начиная с самых сложных частей» (hard part first integration), похожа на сэндвич-интеграцию тем, что пытается избежать проблем, присущих нисходящей или восходящей интеграциям в чистом виде. Кроме того, в ней также есть тенденция к объединению классов верхнего и нижнего уровней в первую очередь, оставляя классы среднего уровня напоследок. Однако суть в другом.

При риск-ориентированной интеграции вы определяете степень риска, связанную с каждым классом. Вы решаете, какие части системы будут самыми трудными, и реализуете их первыми.

#### Функционально-ориентированная интеграция

Еще один подход — интеграция одной функции в каждый момент времени. Под «функцией» понимается не нечто расплывчатое, а какое-нибудь поддающееся определению свойство системы, в которой выполняется интеграция.

Когда интегрируемая функция превышает по размерам отдельный класс, то «единица приращения» инкрементной интеграции становится больше отдельного класса. Это немного снижает преимущество инкрементного подхода в том плане, что уменьшает вашу уверенность об источнике новых ошибок. Однако если вы тщательно тестировали классы, реализующие эту функцию, перед интеграцией, то это лишь небольшой недостаток. Вы можете использовать стратегии инкрементной интеграции рекурсивно, сформировав сначала из небольших кусков отдельные свойства, а затем инкрементно объединив их в систему.

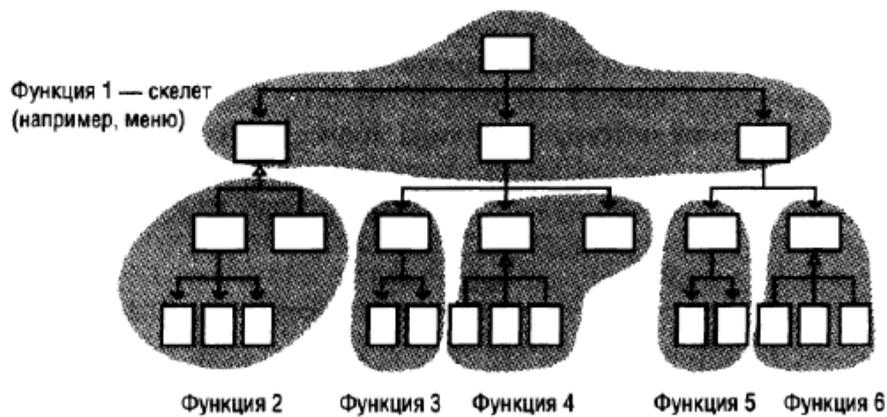


Рис. 8 Функционально-ориентированная интеграция

Обычно процесс начинается с формирования скелета, поскольку он способен поддерживать остальную функциональность. В интерактивной системе такой изначальной опцией может стать система интерактивного меню. Вы можете прикреплять остальную функциональность к той опции, которую интегрировали первой.

#### Т-образная интеграция

Последний подход, который часто упоминается в связи с проблемами нисходящей и восходящей методик, называется «Т-образной интеграцией». При таком подходе выбирается некоторый вертикальный слой, который разрабатывается и интегрируется раньше других. Этот слой должен проходить сквозь всю систему от начала до конца и позволять выявлять основные проблемы в допущениях, сделанных при проектировании системы. Реализовав этот вертикальный участок (и устранив все связанные с этим проблемы), можно разрабатывать основную канву системы (например, системное меню для настольного приложения). Этот подход часто комбинируют с риск-ориентированной и функционально-ориентированной интеграциями.

#### Г-образная интеграция

Последний подход, который часто упоминается в связи с проблемами нисходящей и восходящей методик, называется «Г-образной интеграцией». При таком подходе выбирается некоторый вертикальный слой, который разрабатывается и интегрируется раньше других. Этот слой должен проходить сквозь всю систему от начала до конца и позволять выявлять основные проблемы в допущениях, сделанных при проектировании системы. Реализовав этот вертикальный участок (и устранив все связанные с этим проблемы), можно разрабатывать основную канву системы (например, системное меню для настольного приложения). Этот подход часто комбинируют с риск-ориентированной и функционально-ориентированной интеграциями.

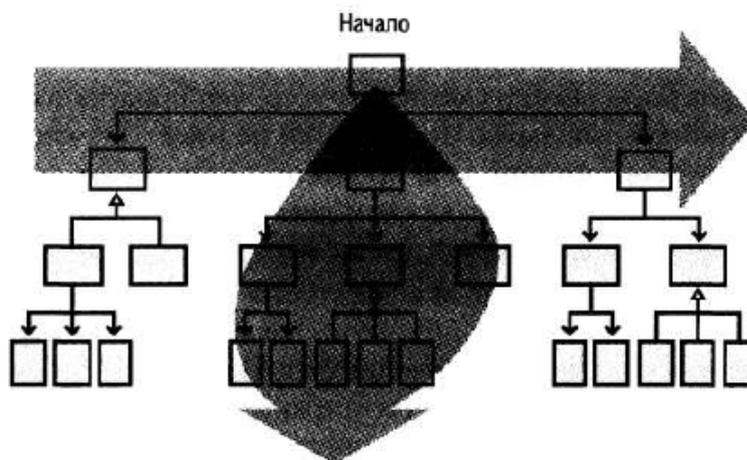


Рис. 9 Т-образная интеграция

**Задание 2:**

1. Оформить внешнюю спецификацию.
2. Составить в виде блок-схемы алгоритм решения задачи.
3. Спроектировать и разработать модули программы для решения задачи на любом алгоритмическом языке программирования.
4. Выполнить отладку и тестирование модулей программы.
5. Выполнить инкрементную интеграцию модулей с использованием одного из подходов.
6. Выполнить системное тестирование программы.

**Задача.** Задан двумерный массив размерности  $n \times m$ . Отсортировать элементы строк массива по возрастанию значений, а затем отсортировать строки массива по возрастанию среднего арифметического элементов строк.

Реализовать сортировку разными способами и сравнить эффективность этих способов для разных исходных данных.

**Контрольные вопросы.**

1. Значение фазы интеграции программных модулей.
  2. Подходы к интегрированию программных модулей.
- Эффективность и оптимизация программ.

**Форма отчета:** отчет, защита работы.

**Практическое занятие № 28**

**Цель:** получение практических навыков автоматической генерации теста на основе формального описания

**Задание 1:** ознакомиться с материалом

Практически все программные системы предусматривают интерфейс с оператором. Практически всегда этот интерфейс – графический (GUI – Graphical User's Interface). Соответственно, актуальна и задача тестирования создаваемого графического интерфейса.

Вообще говоря, задача тестирования создаваемых программ возникла практически одновременно с самими программами. Известно, что эта задача очень трудоёмка как в смысле усилий по созданию достаточного количества тестов (отвечающих заданному критерию тестового покрытия), так и в смысле времени прогона всех этих тестов. Поэтому решение этой задачи стараются автоматизировать (в обоих смыслах).

Для решения задачи тестирования программ с программным интерфейсом (API – Application Program Interface: вызовы методов или процедур, пересылки сообщений) известны подходы – методы и инструменты – хорошо зарекомендовавшие себя в индустрии создания программного обеспечения. Основа этих подходов следующая: создается формальная спецификация программы, и по этой спецификации генерируются как сами тесты, так и тестовые оракулы – программы, проверяющие правильность поведения тестируемой программы. Спецификации, как набор требований к создаваемой программе, существовали всегда. Ключевым словом здесь является формальная спецификация. Формальная спецификация – это спецификация в форме, допускающей её формальные же преобразования и обработку компьютером. Это позволяет анализировать набор требований с точки зрения их полноты, непротиворечивости и т.п. Для задачи автоматизации тестирования эта формальная запись должна также обеспечивать возможность описания формальной связи между понятиями, используемыми в спецификации, и сущностями языка реализации программы.

Правильность функционирования системы определяется соответствием реального поведения системы эталонному поведению. Для того чтобы качественно определять это соответствие, нужно уметь формализовать эталонное поведение системы. Распространённым способом описания поведения системы является описание с помощью диаграмм UML (Unified Modeling Language). Стандарт UML предлагает использование трех видов диаграмм для описания графического интерфейса системы:

- ✓ Диаграммы сценариев использования (Use Case).
- ✓ Диаграммы конечных автоматов (State Chart).
- ✓ Диаграммы действий (Activity).

С помощью UML/Use Case diagram можно описывать на высоком уровне наборы сценариев использования, поддерживаемых системой. Данный подход имеет ряд преимуществ и недостатков по отношению к другим подходам, но существенным с точки зрения автоматизации генерации тестов является недостаточная формальная строгость описания.

Широко распространено использование UML/State Chart diagram для спецификации поведения системы, и такой подход очень удобен с точки зрения генерации тестов. Но составление спецификации поведения современных систем с помощью конечных автоматов есть очень трудоёмкое занятие, так как число состояний системы очень велико. С ростом функциональности системы спецификация становится всё менее и менее наглядной.

Перечисленные недостатки этих подходов к описанию поведения системы преодолеваются с помощью диаграмм действий (Activity). С одной стороны нотация UML/Activity diagram является более строгой, чем у сценариев использования, а с другой стороны предоставляет более широкие возможности по сравнению с диаграммами конечных автоматов.

Для создания прототипа работающей версии данного подхода используется инструмент Rational Rose. В первую очередь для спецификации графического интерфейса пользователя при помощи диаграмм действий UML.

Для прогона сгенерированных по диаграмме состояний тестов используется инструмент Rational Robot. Из возможностей инструмента в работе мы использовали следующие:

1. Возможность выполнять тестовые воздействия, соответствующие переходам между состояниями в спецификации.

2. Возможность проверять соответствие свойств объектов реальной системы и эталонных свойств, содержащихся в спецификации. Из тех возможностей, которые доступны с помощью этого инструмента, используется проверка следующих свойств объектов:

- Наличие и состояние окон (заголовок, активность, доступность, статус).
- Наличие и состояние таких объектов, как PushButton, CheckBox, RadioButton, List, Tree и др. (текст, доступность, размер).
- Значение буфера обмена.
- Наличие в оперативной памяти запущенных процессов.
- Существование файлов.

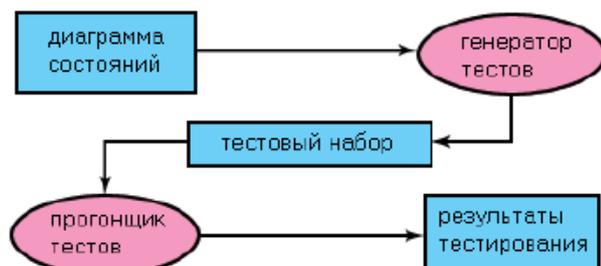


Рис. 2 Общая схема генерации и прогона тестов

Генератор строит по диаграмме состояний набор тестов. Условием окончания работы генератора является выполнение тестового покрытия. В данном случае в качестве покрытия было выбрано условие прохода по всем рёбрам графа состояний, то есть выполнения каждого доступного тестового воздействия из каждого достижимого состояния.

Автоматическая генерация тестов по диаграммам действий имеет следующие преимущества перед остальными подходами к тестированию графического интерфейса:

Генератор тестов есть программа (script), написанная на языке 'Rational Rose Scripting language' (расширение к языку Summit BasicScriptLanguage). В Rational Rose есть встроенный интерпретатор инструкций, написанных на этом языке, посредством которого можно обращаться ко всем объектам модели (диаграммы состояний).

- Спецификация автоматически интерпретируется (тем самым она проверяется и компилируется в набор тестов).
- Если какая-то функциональность системы изменилась, то диаграмму состояний достаточно изменить в соответствующем месте, и затем сгенерировать новый тестовый набор. Фактически, это снимает большую часть проблем, возникающих при организации регрессионного тестирования.
- Гарантия тестового покрытия. Эта гарантия даётся соответствующим алгоритмом обхода графа состояний.

В процессе построения обхода, генератор тестов компилирует набор тестов - инструкции на языке SQABasic. Эти инструкции есть чередование тестовых воздействий и оракула свойств объектов, соответствующих данному состоянию.

### Задание 2:

1. Сформировать диаграмму вариантов использования для задачи
2. Сгенерировать набор тестов.

**Форма отчета:** отчет, защита работы.

### Практическое занятие № 29

**Цель:** получение практических навыков использования средств автоматизации тестирования

**Задание 1:** ознакомиться с материалом

Для того чтобы продолжать тестирование, когда один тест не прошёл, в генератор тестов встроена возможность выбора – генерировать один большой тест или набор атомарных тестов. Атомарный тест – тот, который не требует приведения системы в состояние, отличное от начального состояния.

В связи с наличием ограничения инструмента прогона тестов на тестовую длину, в тесты после каждой законченной инструкции вставляется строка разреза. Во время прогона по этим строкам осуществляется разрез теста в случае, если его длина превышает допустимое ограничение. После прохождения части теста до строки разреза продолжается выполнение теста с первой инструкции, следующей за строкой разреза. Нарезку и сам прогон тестов осуществляет прогонщик тестов.

В качестве прогонщика тестов мы используем Rational Robot, который выполняет сгенерированные наборы инструкций. В случае удачного выполнения всех инструкций выносится вердикт – тест прошёл. В противном случае, если на каком-то этапе выполнения теста, поведение системы не соответствует требованиям, Robot прекращает его выполнение, вынося соответствующий вердикт – тест не прошёл.

## **Задание 2:**

1. Выполнить тестовый набор л
2. Проанализировать отчёт о прохождении тестов.

**Форма отчета:** отчет, защита работы.

### **Практическое занятие № 30**

**Цель:** получение практических навыков разработки тестов на основе внешней спецификации программы

#### **Задание 1: ознакомиться с материалом**

Программа в случае тестирования с управлением по данным рассматривается как "черный ящик", и целью тестирования является выяснение обстоятельств, в которых поведение программы не соответствует спецификации. Различают следующие методы формирования тестовых наборов:

- эквивалентное разбиение;
- анализ граничных значений;
- анализ причинно-следственных связей;
- предположение об ошибке.

#### **Эквивалентное разбиение.**

Область всех возможных наборов входных данных программы по каждому параметру разбивают на конечное число групп - *классов эквивалентности*. Наборы данных такого класса объединяют по принципу обнаружения одних и тех же ошибок. Для составления классов эквивалентности нужно перебрать ограничения, установленные для каждого входного значения в техническом задании или при уточнении спецификации. Каждое ограничение разбивают на две или более групп.

#### **Граничные значения.**

Граничные значения - это значения на границах классов эквивалентности входных значений или около них.

#### **Анализ причинно-следственных связей.**

Метод *анализа причинно-следственных связей* позволяет системно выбирать тесты, используя алгебру логики. *Причиной* называют отдельное входное условие или класс эквивалентности. *Следствием* - выходное условие или преобразование системы. Идея заключается в отнесении всех следствий к причинам, то есть в уточнении причинно-следственных связей.

#### **Предположение об ошибке.**

Метод основан на интуиции программиста с большим опытом работы. Составляется список, в котором перечисляются возможные ошибки или ситуации, в которых они могут появиться, а затем на основе списка составляются тесты.

## **Задание 2:**

На основе внешней спецификации задачи составить набор тестов на основе подхода черного ящика  
Произвести тестирование программы

**Форма отчета:** отчет, защита работы.

### **Практическое занятие № 31**

**Цель:** получение практических навыков отладки программ с помощью отладчика среды программирования

### **Задание 1:** ознакомиться с материалом

Отладка — это процесс определения и устранения причин ошибок. Этим она отличается от тестирования, направленного на обнаружение ошибок. В некоторых проектах отладка занимает до 50% общего времени разработки. Многие программисты считают отладку самым трудным аспектом программирования.

Для сокращения времени отладки необходимо пользоваться научным подходом.

Классический научный подход включает следующие этапы:

1. Сбор данных при помощи повторяющихся экспериментов.
2. Формулирование гипотезы, объясняющей релевантные данные.
3. Разработка эксперимента, призванного подтвердить или опровергнуть гипотезу.
4. Подтверждение или опровержение гипотезы.
5. Повторение процесса в случае надобности.

Эффективный метод поиска дефектов при отладке с использованием научного подхода может быть описан следующими шагами:

1. Стабилизация ошибки.
2. Определение источника ошибки.
  - a. Сбор данных, приводящих к дефекту.
  - b. Анализ собранных данных и формулирование гипотезы, объясняющей дефект.
  - c. Определение способа подтверждения или опровержения гипотезы, основанного или на тестировании программы, или на изучении кода.
  - d. Подтверждение или опровержение гипотезы при помощи процедуры, определенной в п. 2(c).
3. Исправление дефекта.
4. Тестирование исправления.
5. Поиск похожих ошибок.

Способ подтверждения или опровержения гипотезы может быть одним из следующего списка:

1. сокращение подозрительной области кода;
2. проверка классов и методов, в которых дефекты обнаруживались ранее;
3. проверка кода, который изменялся недавно.

Отладка — это тот этап разработки программы, от которого зависит возможность ее выпуска. Конечно, лучше всего вообще избегать ошибок. Однако потратить время на улучшение навыков отладки все же стоит, потому что эффективность отладки, выполняемой лучшими и худшими программистами, различается минимум в 10 раз.

Систематичный подход к поиску и исправлению ошибок — неперенное условие успешности отладки. Организуйте отладку так, чтобы каждый тест приближал вас к цели. Используйте Научный Метод Отладки.

Прежде чем приступать к исправлению программы, поймите суть проблемы. Случайные предположения о причинах ошибок и случайные исправления только ухудшат программу.

Установите в настройках компилятора самый строгий уровень диагностики и устраняйте причины всех ошибок и предупреждений.

Инструменты отладки значительно облегчают разработку ПО. Найдите их и используйте. Большинство современных сред программирования (Delphi, C++ Builder, Visual Studio и т.д.) включают средства отладки, которые обеспечивают максимально эффективную отладку. Они позволяют:

- выполнять программу по шагам, причем как с заходом в подпрограммы, так и выполняя их целиком;
- предусматривать точки останова;
- выполнять программу до оператора, указанного курсором;
- отображать содержимое любых переменных при пошаговом выполнении;
- отслеживать поток сообщений и т.п.

### **Задание 2:**

1. Составить в виде блок-схемы алгоритм решения задачи.
2. Создать программу решения задачи на любом алгоритмическом языке программирования.

3. Отладить программу с использованием инструментальных средств.

**Задача:** Имеется матрица размера  $N \times M$ . Определить в какой строке количество положительных элементов наибольшее.

**Контрольные вопросы.**

1. Что такое тестирование программы?
2. Что такое отладка программы?
3. Какие стадии тестирования выделяют при разработке программного обеспечения?
4. Какие различают подходы в формировании тестовых наборов?
5. В чем суть тестирования методом “покрытия операторов”?
6. В чем суть тестирования методом “покрытия решений”?
7. В чем суть тестирования методом “покрытия условий”?
8. В чем суть тестирования методом “комбинаторного покрытия условий”?
9. В чём суть метода эквивалентных разбиений?
10. В чём суть метода анализа граничных значений?
11. В чём суть метода анализа причинно-следственных связей?

**Форма отчета:** отчет, защита работы.

### Раздел 3. Моделирование в программных системах

#### Практическое занятие № 1

**Тема:** построение простейших математических моделей

**Цель:** закрепить практические навыки по построению простейших математических и простейших статистических моделей

**Оборудование:** тетрадь, ручка

**Методические указания:** приступите к выполнению задания

**Ход выполнения:**

**1 вариант.**

**Задача 1.** Составить математическую модель следующей задачи. Предположим, что для производства продукции вида А и В можно использовать материал трех сортов. При этом на изготовление единицы изделия вида А расходуется  $a_1$  кг первого сорта,  $a_2$  кг второго сорта и  $a_3$  кг третьего сорта. На изготовление продукции вида В расходуется  $b_1$  кг первого сорта,  $b_2$  кг второго сорта,  $b_3$  кг третьего сорта. На складе фабрики имеется всего материала первого сорта  $c_1$  кг, второго сорта  $c_2$  кг, третьего сорта  $c_3$  кг. От реализации единицы готовой продукции вида А фабрика имеет прибыль вида  $\alpha$  руб., а от реализации единицы готовой продукции вида В фабрика имеет прибыль вида  $\beta$  руб. Определить максимальную прибыль от реализации всей продукции видов А и В.

$$a_1=19, a_2=16, a_3=19, b_1=26, b_2=17, b_3=8, c_1=868, c_2=638, c_3=853, \\ \alpha=5, \beta=4.$$

**Задача 2.** Имеются три пункта поставки однородного груза А1, А2, А3 и пять пунктов В1, В2, В3, В4, В5 потребления этого груза. На пунктах А1, А2 и А3 находится груз соответственно в количестве  $a_1, a_2$  и  $a_3$  тонн. В пункты В1, В2, В3, В4, В5 требуется доставить соответственно  $b_1, b_2, b_3, b_4, b_5$  тонн груза. Расстояние между пунктами поставки и пунктами потребления приведено в таблице:

Пункты поставки	Пункты потребления				
	В1	В2	В3	В4	В5
А1	D11	D12	D13	D14	D15
А2	D21	D22	D23	D24	D25
А3	D31	D32	D33	D34	D35

Найти такой план закрепления потребителей за поставщиками однородного груза, чтобы общие затраты по перевозкам были минимальными.

$$a_1=300, a_2=250, a_3=200, \\ b_1=210, b_2=150, b_3=120, b_4=135, \\ b_5=135.$$

$$D = \begin{pmatrix} 4 & 8 & 13 & 2 & 7 \\ 9 & 4 & 11 & 9 & 17 \\ 3 & 16 & 10 & 1 & 4 \end{pmatrix}$$

## 2 вариант.

**Задача 1.** Составить математическую модель следующей задачи. Предположим, что для производства продукции вида А и В можно использовать материал трех сортов. При этом на изготовление единицы изделия вида А расходуется  $a_1$  кг первого сорта,  $a_2$  кг второго сорта и  $a_3$  кг третьего сорта. На изготовление продукции вида В расходуется  $b_1$  кг первого сорта,  $b_2$  кг второго сорта,  $b_3$  кг третьего сорта. На складе фабрики имеется всего материала первого сорта  $c_1$  кг, второго сорта  $c_2$  кг, третьего сорта  $c_3$  кг. От реализации единицы готовой продукции вида А фабрика имеет прибыль вида  $\alpha$  руб., а от реализации единицы готовой продукции вида В фабрика имеет прибыль вида  $\beta$  руб. Определить максимальную прибыль от реализации всей продукции видов А и В.

$$a_1=14, a_2=15, a_3=20, b_1=40, b_2=27, b_3=4, c_1=1200, c_2=993, c_3=1097, \\ \alpha=5, \beta=13.$$

**Задача 2.** Имеются три пункта поставки однородного груза А1, А2, А3 и пять пунктов В1, В2, В3, В4, В5 потребления этого груза. На пунктах А1, А2 и А3 находится груз соответственно в количестве  $a_1, a_2$  и  $a_3$  тонн. В пункты В1, В2, В3, В4, В5 требуется доставить соответственно  $b_1, b_2, b_3, b_4, b_5$  тонн груза. Расстояние между пунктами поставки и пунктами потребления приведено в таблице:

Пункты поставки	Пункты потребления				
	В1	В2	В3	В4	В5
А1	D11	D12	D13	D14	D15
А2	D21	D22	D23	D24	D25
А3	D31	D32	D33	D34	D35

Найти такой план закрепления потребителей за поставщиками однородного груза, чтобы общие затраты по перевозкам были минимальными.

$$a_1=350, a_2=200, a_3=300, \\ b_1=170, b_2=140, b_3=200, b_4=195, \\ b_5=145, \quad D = \begin{pmatrix} 22 & 14 & 16 & 28 & 30 \\ 19 & 17 & 26 & 36 & 36 \\ 37 & 30 & 31 & 39 & 41 \end{pmatrix}$$

**Форма отчета:** тетрадь, защита работы.

### Практическое занятие № 2

**Тема:** построение простейших статистических моделей

**Цель:** определить оптимальное решение однокритериальных и многокритериальных задач в простейших случаях

**Оборудование:** тетрадь, ручка

**Методические указания:** приступите к выполнению задания

**Ход выполнения:**

### 1 вариант.

**Задача 1.** Составить математическую модель следующей задачи. Предположим, что для производства продукции вида А и В можно использовать материал трех сортов. При этом на изготовление единицы изделия вида А расходуется  $a1$  кг первого сорта,  $a2$  кг второго сорта и  $a3$  кг третьего сорта. На изготовление продукции вида В расходуется  $b1$  кг первого сорта,  $b2$  кг второго сорта,  $b3$  кг третьего сорта. На складе фабрики имеется всего материала первого сорта  $c1$  кг, второго сорта  $c2$  кг, третьего сорта  $c3$  кг. От реализации единицы готовой продукции вида А фабрика имеет прибыль вида  $\alpha$  руб., а от реализации единицы готовой продукции вида В фабрика имеет прибыль вида  $\beta$  руб. Определить максимальную прибыль от реализации всей продукции видов А и В.

$$a1=19, a2=16, a3=19, b1=26, b2=17, b3=8, c1=868, c2=638, c3=853, \\ \alpha=5, \beta=4.$$

**Задача 2.** Фирме необходимо выбрать наилучший вариант закупки оборудования, если задана закупочная цена каждого из вариантов оборудования и время изготовления и доставки. Под наилучшим вариантом понимается вариант с минимальными закупочной стоимостью и временем доставки.

А) Для заданной двухкритериальной задачи, задавшись коэффициентами  $\alpha$  и  $\beta$  провести линейную свертку критериев  $F_1(x)$  и  $F_2(x)$  и определить минимальное решение.

Б) Для заданной двухкритериальной задачи найти множество Парето в случае двух критериев вида  $F_1(x) \rightarrow \min$  и  $F_2(x) \rightarrow \min$ .

Значения  $F_1(x_i)$  и  $F_2(x_i)$  заданы таблицей:

$x_i$	1	2	3	4	5	6	7	8
$F_1(x_i)$	2	2	2	3	3	4	4	5
$F_2(x_i)$	4	5	6	3	4	4	5	3

### 2 вариант.

**Задача 1.** Составить математическую модель следующей задачи. Предположим, что для производства продукции вида А и В можно использовать материал трех сортов. При этом на изготовление единицы изделия вида А расходуется  $a1$  кг первого сорта,  $a2$  кг второго сорта и  $a3$  кг третьего сорта. На изготовление продукции вида В расходуется  $b1$  кг первого сорта,  $b2$  кг второго сорта,  $b3$  кг третьего сорта. На складе фабрики имеется всего материала первого сорта  $c1$  кг, второго сорта  $c2$  кг, третьего сорта  $c3$  кг. От реализации единицы готовой продукции вида А фабрика имеет прибыль вида  $\alpha$  руб., а от реализации единицы готовой продукции вида В фабрика имеет прибыль вида  $\beta$  руб. Определить максимальную прибыль от реализации всей продукции видов А и В.

$$a1=14, a2=15, a3=20, b1=40, b2=27, b3=4, c1=1200, c2=993, c3=1097, \\ \alpha=5, \beta=13.$$

**Задача 2.** Фирме необходимо выбрать наилучший вариант закупки оборудования, если задана закупочная цена каждого из вариантов оборудования и время изготовления и доставки. Под наилучшим вариантом понимается вариант с минимальными закупочной стоимостью и временем доставки.

А) Для заданной двухкритериальной задачи, задавшись коэффициентами  $\alpha$  и  $\beta$  провести линейную свертку критериев  $F_1(x)$  и  $F_2(x)$  и определить минимальное решение.

Б) Для заданной двухкритериальной задачи найти множество Парето в случае двух критериев вида  $F_1(x) \rightarrow \min$  и  $F_2(x) \rightarrow \min$ .

Значения  $F_1(x_i)$  и  $F_2(x_i)$  заданы таблицей:

$x_i$	1	2	3	4	5	6	7	8
$F_1(x_i)$	3	3	3	4	4	5	5	6
$F_2(x_i)$	5	6	7	4	5	5	6	4

### Практическое занятие № 3

**Тема:** Задача Коши для уравнения теплопроводности

**Цель:** определить оптимальное решение однокритериальных и многокритериальных задач в простейших случаях

**Оборудование:** тетрадь, ручка

**Методические указания:** приступите к выполнению задания

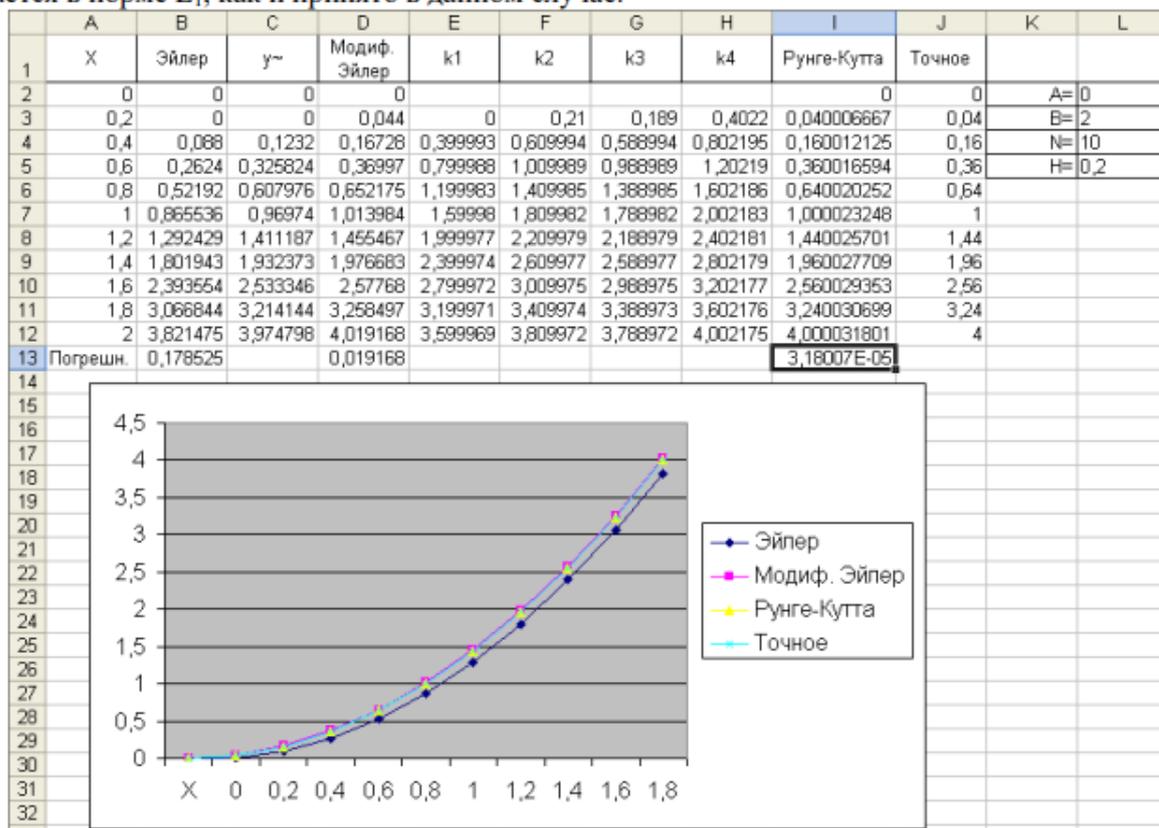
**Ход выполнения:**

Решение обыкновенных дифференциальных уравнений (ОДУ)

В приведённом примере решается задача Коши, то есть, ищется решение дифференциального уравнения первого порядка вида  $dy/dx = f(x,y)$  на интервале  $x \in [x_0, x_n]$  при условии  $y(x_0)=y_0$  и равномерном шаге сетки по  $x$ .

Решение выполняется методами Эйлера, "предиктор-корректор" (он же модифицированный метод Эйлера) и методом Рунге-Кутты 4 порядка точности. Пример может служить образцом для Ваших решений, правда, функцию придётся перепрограммировать несколько раз при различных значениях аргумента - поскольку без применения макросов на VBA Excel не позволяет создать полноценную функцию, которую было бы удобно вызывать с разными значениями аргументов.

Здесь решается уравнение  $dy/dx = 2x-y+x^2$  на интервале  $[0,2]$ , начальное значение  $y(0)=0$ , для оценки точности задано также точное решение в виде функции  $u(x)=x^2$ . Оценка погрешности делается в норме  $L_1$ , как и принято в данном случае.



**Форма отчета:** тетрадь, защита работы.

### Практическое занятие № 4

**Тема:** Решение задач линейного программирования симплекс - методом

**Цель:** научиться сводить произвольную задачу линейного программирования к основной задаче линейного программирования. Решить задачу линейного программирования симплекс-методом.

**Оборудование:** тетрадь, ручка

**Методические указания:** приступите к выполнению задания

**Ход выполнения:**

### 1 вариант.

**Задача 1. а)** Привести к канонической форме задачу линейного программирования.

$$\begin{cases} x_1 - 2x_2 + x_3 \geq 4, \\ x_1 + x_2 - 3x_3 \leq 9 \\ x_1 + 3x_2 + 2x_3 = 10 \end{cases}$$

$$x_1 \geq 0, x_2 \geq 0, x_3 \geq 0$$

$$F = 2x_1 + x_2 - x_3 \rightarrow \max$$

**б)** Напишите задачу в стандартной форме.

**Задача 2.** Предположим, что для производства продукции вида А и В можно использовать материал трех сортов. При этом на изготовление единицы изделия вида А расходуется  $a1$  кг первого сорта,  $a2$  кг второго сорта и  $a3$  кг третьего сорта. На изготовление продукции вида В расходуется  $b1$  кг первого сорта,  $b2$  кг второго сорта,  $b3$  кг третьего сорта. На складе фабрики имеется всего материала первого сорта  $c1$  кг, второго сорта  $c2$  кг, третьего сорта  $c3$  кг. От реализации единицы готовой продукции вида А фабрика имеет прибыль вида  $\alpha$  руб., а от реализации единицы готовой продукции вида В фабрика имеет прибыль вида  $\beta$  руб. Определить максимальную прибыль от реализации всей продукции видов А и В симплекс-методом.

$$a1=19, a2=16, a3=19, b1=31, b2=9, b3=1, c1=1121, c2=706, c3=1066, \\ \alpha=16, \beta=19.$$

### 2 вариант.

**Задача 1. а)** Привести к канонической форме задачу линейного программирования.

$$\begin{cases} 4x_1 + 2x_2 + 5x_3 \leq 12, \\ 6x_1 - 3x_2 + 4x_3 = 18 \\ 3x_1 + 3x_2 - 2x_3 \geq 16 \end{cases}$$

$$x_1 \geq 0, x_2 \geq 0, x_3 \geq 0$$

$$F = -2x_1 + x_2 + 5x_3 \rightarrow \max$$

**б)** Напишите задачу в стандартной форме.

**Задача 2.** Предположим, что для производства продукции вида А и В можно использовать материал трех сортов. При этом на изготовление единицы изделия вида А расходуется  $a1$  кг первого сорта,  $a2$  кг второго сорта и  $a3$  кг третьего сорта. На изготовление продукции вида В расходуется  $b1$  кг первого сорта,  $b2$  кг второго сорта,  $b3$  кг третьего сорта. На складе фабрики имеется всего материала первого сорта  $c1$  кг, второго сорта  $c2$  кг, третьего сорта  $c3$  кг. От реализации единицы готовой продукции вида А фабрика имеет прибыль вида  $\alpha$  руб., а от реализации единицы готовой продукции вида В фабрика имеет прибыль вида  $\beta$  руб. Определить максимальную прибыль от реализации всей продукции видов А и В симплекс-методом.

$$a1=14, a2=15, a3=20, b1=40, b2=27, b3=4, c1=1200, c2=993, c3=1097, \\ \alpha=5, \beta=13.$$

**Форма отчета:** тетрадь, защита работы.

### Практическое занятие № 5

**Тема:** Решение задач линейного программирования симплекс - методом

**Цель:** научиться решать задачи геометрическим методом, научиться решать задачи симплексным методом, закрепить навыки записи взаимосвязи показателей задачи в виде математической модели

**Оборудование:** тетрадь, ручка.

**Методические указания:** приступите к выполнению задания

**Ход выполнения:**

**Задание 1**

На предприятии имеется сырье видов I, II, III. Из него можно изготавливать изделия типов А и В. Пусть запасы видов сырья на предприятии составляют  $b_1, b_2, b_3$  ед. соответственно, изделие типа А дает прибыль  $c_1$  ден.ед., а изделие типа В -  $c_2$  ден.ед. Расход сырья на изготовление одного изделия задан в словных единицах таблицей.

Составить план выпуска изделий, при котором предприятие имеет наибольшую прибыль. Решить задачу графическим методом.

<b>1 вариант</b>								
Изделие	Сырье			$b_1$	$b_2$	$b_3$	$c_1$	$c_2$
	I	II	III					
А	6	3	2	102	91	105	5	9
В	3	4	5					

<b>2 вариант</b>								
Изделие	Сырье			$b_1$	$b_2$	$b_3$	$c_1$	$c_2$
	I	II	III					
А	1	1	3	20	36	40	2	5
В	3	2	1					

<b>3 вариант</b>								
Изделие	Сырье			$b_1$	$b_2$	$b_3$	$c_1$	$c_2$
	I	II	III					
А	2	1	3	40	34	46	1	2
В	2	2	1					

<b>4 вариант</b>								
Изделие	Сырье			$b_1$	$b_2$	$b_3$	$c_1$	$c_2$
	I	II	III					
А	3	4	3	300	520	600	6	3
В	1	3	4					

## Задание 2

Предприятие выпускает три вида изделий (N1, N2, N3), используя три вида ресурсов (P1, P2, P3). Запасы ресурсов (З) ограничены. Прибыль от реализации (П) единицы изделия и нормы расхода ресурсов представлены в таблице. Определить ассортимент и объемы выпуска продукции, получаемую прибыль, величину остатков. Найти решение задачи симплексным методом с представлением всех симплексных таблиц и проанализировать полученные результаты.

Вариант 1				
	N1	N2	N3	З
P1	1	8	5	43
P2	4	1	6	74
P3	5	2	2	35
П	5	7	6	

Вариант 2				
	N1	N2	N3	З
P1	3	5	4	81
P2	6	1	3	74
P3	1	5	2	33
П	4	8	7	

Вариант 3				
	N1	N2	N3	З
P1	6	7	2	57
P2	6	6	1	97
P3	3	7	8	63
П	5	6	8	

**Форма отчета:** тетрадь, защита работы.

### Практическое занятие № 6-10

**Тема:** Графический метод решения задач линейного программирования; нахождение начального решения транспортной задачи; решение транспортной задачи методом потенциалов

**Цель:** найти начальное решение транспортной задачи двумя методами: методом северо-западного угла и методом наименьшей стоимости; найти оптимальное решение транспортной задачи методом потенциалов.

**Оборудование:** тетрадь, ручка

**Методические указания:** приступите к выполнению задания

**Ход выполнения:**

#### 1 вариант.

**Задача.** Имеются три пункта поставки однородного груза **A1, A2, A3** и пять пунктов **B1, B2, B3, B4, B5** потребления этого груза. На пунктах **A1, A2 и A3** находится груз соответственно в количестве **a1, a2 и a3** тонн. В пункты **B1, B2, B3, B4, B5** требуется доставить соответственно **b1, b2, b3, b4, b5** тонн груза. Расстояние между пунктами поставки и пунктами потребления приведено в таблице:

Пункты поставки	Пункты потребления				
	B1	B2	B3	B4	B5
A1	D11	D12	D13	D14	D15

<b>A2</b>	D21	D22	D23	D24	D25
<b>A3</b>	D31	D32	D33	D34	D35

Найти такой план закрепления потребителей за поставщиками однородного груза, чтобы общие затраты по перевозкам были минимальными.

$$\begin{aligned}
 & a1=200, \quad a2=350, \quad a3=300, \\
 & b1=270, \quad b2=130, \quad b3=190, \quad b4=150, \\
 & b5=110.
 \end{aligned}
 \quad D = \begin{pmatrix} 24 & 50 & 55 & 27 & 16 \\ 50 & 47 & 23 & 17 & 21 \\ 35 & 59 & 55 & 27 & 41 \end{pmatrix}$$

**2 вариант.**

**Задача.** Имеются три пункта поставки однородного груза **A1, A2, A3** и пять пунктов **B1, B2, B3, B4, B5** потребления этого груза. На пунктах **A1, A2 и A3** находится груз соответственно в количестве *a1, a2 и a3* тонн. В пункты **B1, B2, B3, B4, B5** требуется доставить соответственно *b1, b2, b3, b4, b5* тонн груза. Расстояние между пунктами поставки и пунктами потребления приведено в таблице:

Пункты поставки	Пункты потребления				
	<b>B1</b>	<b>B2</b>	<b>B3</b>	<b>B4</b>	<b>B5</b>
<b>A1</b>	D11	D12	D13	D14	D15
<b>A2</b>	D21	D22	D23	D24	D25
<b>A3</b>	D31	D32	D33	D34	D35

Найти такой план закрепления потребителей за поставщиками однородного груза, чтобы общие затраты по перевозкам были минимальными.

$$\begin{aligned}
 & a1=200, \quad a2=300, \quad a3=250, \\
 & b1=210, \quad b2=150, \quad b3=120, \quad b4=135, \\
 & b5=135.
 \end{aligned}
 \quad D = \begin{pmatrix} 20 & 10 & 13 & 13 & 18 \\ 27 & 19 & 20 & 16 & 22 \\ 26 & 17 & 19 & 21 & 23 \end{pmatrix}$$

**Форма отчета:** тетрадь, защита работы.

### Практическое занятие № 11-12

**Тема:** Применение метод стрельбы для решения линейной краевой задачи

**Цель:** научиться применять метод стрельбы для решения линейной краевой задачи

**Оборудование:** тетрадь, ручка

**Методические указания:** приступите к выполнению задания

**Ход выполнения:**

### Задание

1. Начните выполнение работы с темы «Линейная краевая задача». Выбрав с помощью меню один из методов решения линейной краевой задачи, перейдите к пункту меню «Параметры». Наберите следующую краевую задачу:

$$y'' - py = -p, \quad 0 < x < 1, \\ y(0) = 1, \quad y(1) = 1.$$

для  $p = \text{const} > 0$ . Решением этой задачи является функция  $y \equiv 1$ . Установите значение шага сетки  $h = 0,05$ .

1.1. Найдите решение этой задачи методом построения общего решения и методом прогонки для разных  $p$ , начиная с умеренных значений и увеличивая их до величины порядка 1200. Сравните получаемые решения с точным и объясните наблюдаемые эффекты. Попробуйте найти решение этой же задачи методом стрельбы. Проанализируйте, как влияет при разных  $p$  точность задания недостающего начального условия на левом конце интервала на успешное решение задачи методом стрельбы.

1.2. Объясните полученные результаты. Замените левое краевое условие (положите, например,  $y'(0) = 0$ ) и посмотрите, как изменится характер решения.

1.3. Выполните п. 1.1, 1.2 для задачи:

$$y'' + py = p, \quad 0 < x < 1, \\ y(1) = 1, \quad y'(0) = 0.$$

Ее точное решение  $y \equiv 1$ . Объясните полученные результаты. Найдите условие устойчивости метода прогонки для данной задачи.

2. Получите численное решение следующих нелинейных краевых задач:

2.1.  $y'' + px \cos y = 0, \quad 0 < x < 1,$   
 $y'(0) = 0, \quad y(1) = 0, \quad p = 1, 4, 7, 25, 50, 100;$

2.2.  $y'' + \frac{0,5}{1-0,5y} y'^2 = 0, \quad 0 < x < 1,$   
 $y(0) = y_0, \quad y(1) = 0,$

$y_0 = 0,25; 0,5; 1; 1,5; 1,8; 1,9; 1,95;$

2.3.  $y'' + \sin y = 0, \quad 0 < x < x_k,$   
 $y(0) = 0, \quad y(x_k) = \pi,$   
 $x_k = 0,5; 1; 2; 4; 6.$

3. Рассмотрите следующие краевые задачи:

3.1.  $y'' = e^y, \quad 0 < x < 1,$   
 $y(0) = 1, \quad y(1) = a,$

3.2.  $y'' = -e^y, \quad 0 < x \leq 1,$   
 $y(0) = 1, \quad y(1) = a.$

Параметр  $a$  меняется от 0 до 2. Что при этом происходит с решением задач? Почему в задаче 3.2 при значениях  $a > 1,4999\dots$  не работает метод линеаризации?

Замечание. Задача 3 подробно рассмотрена в [29].

4. Рассмотреть две сингулярно-возмущенные задачи (с малым параметром при старших производных):

$$\varepsilon y'' = y(y^2 - 1), \quad -1 < x < 1, \\ y(-1) = y(1) = \sqrt{2}$$

и

$$\varepsilon y'' = -y(y^2 - 1), \quad -1 < x < 1, \\ y(-1) = y(1) = \sqrt{2}.$$

Считаем  $\varepsilon = 10^{-3}$ . Какие численные методы позволят получить решение каждой из этих задач? Почему?

**Форма отчета:** тетрадь, защита работы.

**Практическое занятие № 13-14**

**Тема:** Задача о распределении средств между предприятиями

**Цель:** решить простейшие задачи методом динамического программирования.

**Оборудование:** тетрадь, ручка

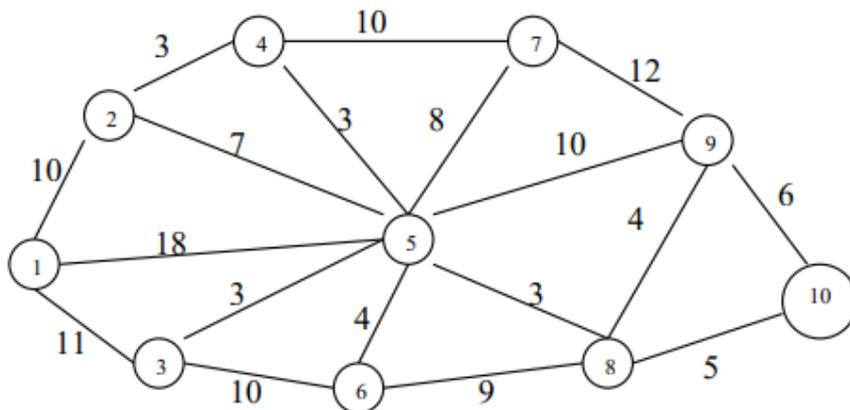
**Методические указания:** приступите к выполнению задания

**Ход выполнения:**

**1 вариант.**

**Задача 1.** Планируется работа двух отраслей производства А и В на 4 года. Количество  $x$  средств, вложенных в отрасль А, позволяет получить доход  $2x$  и уменьшается до  $0,6x$ . Количество  $y$  средств, вложенных в отрасль В, позволяет получить доход  $3y$  и уменьшается до  $0,2y$ . Необходимо распределить выделенные ресурсы в количестве  $S_0 = 850$  единиц между отраслями по годам планируемого периода для получения максимальной прибыли за весь период.

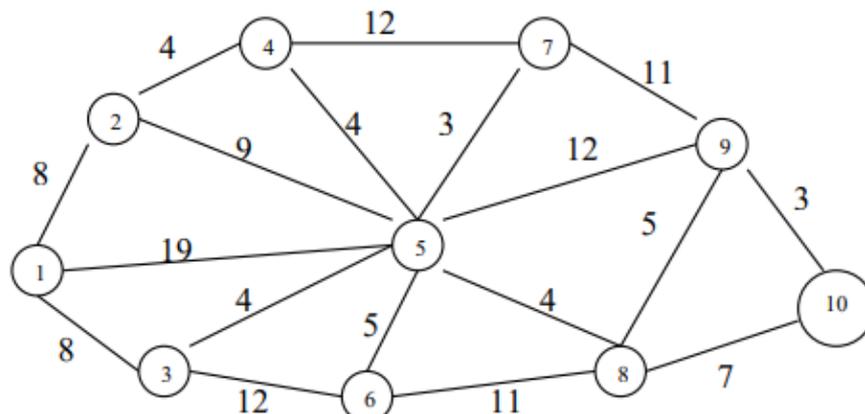
**Задача 2.** По заданной схеме, соединяющей 10 точек, найти кратчайшее расстояние от 1 точки до 10.



**2 вариант.**

**Задача 1.** Двум предприятиям А и В на 4 квартала выделено  $S_0 = 900$  единиц средств. Каждый квартал предприятие А получает  $x$  средств, предприятие В -  $y$  средств. При этом от выделенных средств предприятие А получает  $4x$  единиц и остаток средств  $0,3x$  единиц, а предприятие В - доход  $5y$  единиц и остаток выделенных средств  $0,1y$  единиц. Необходимо распределить средства между предприятиями поквартально таким образом, чтобы за весь год оба предприятия получили максимальный доход.

**Задача 2.** По заданной схеме, соединяющей 10 точек, найти кратчайшее расстояние от 1 точки до 10.



**Форма отчета:** тетрадь, защита работы.

**Практическое занятие № 15-16**

**Тема:** Задача о замене оборудования

**Цель:** научиться решать задачи динамического программирования, научиться разбивать весь процесс решения задачи на этапы, научиться выбирать оптимальную стратегию поведения.

**Оборудование:** тетрадь, ручка

**Методические указания:** приступите к выполнению задания

**Ход выполнения:**

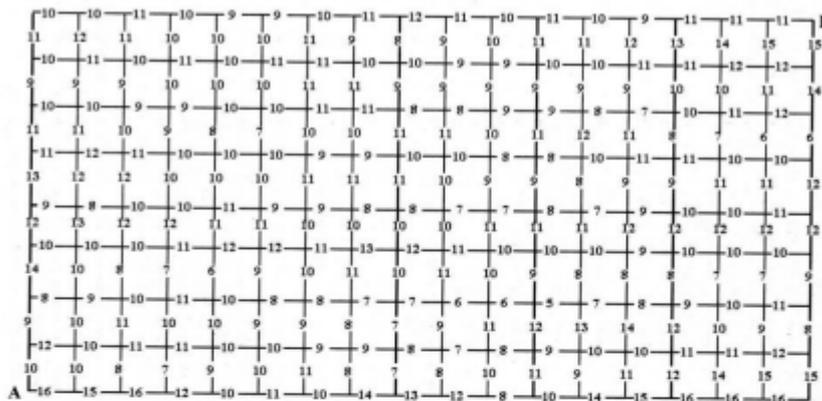
**Задание 1.** Распределите оптимальным образом денежные средства инвестора величиной 5 у.е. между четырьмя предприятиями. Доход каждого предприятия от вложения в него  $u$  у.е. определяется функцией дохода  $f(u)$ . Эти функции приведены в таблице.

u	Прибыль от внедрения по предприятиям			
	f4(u)	f3(u)	f2(u)	f1(u)
1	f4(1)	6	3	4
2	10	f3(2)	4	6
3	11	11	f2(3)	8
4	12	13	11	f1(4)
5	18	15	18	16

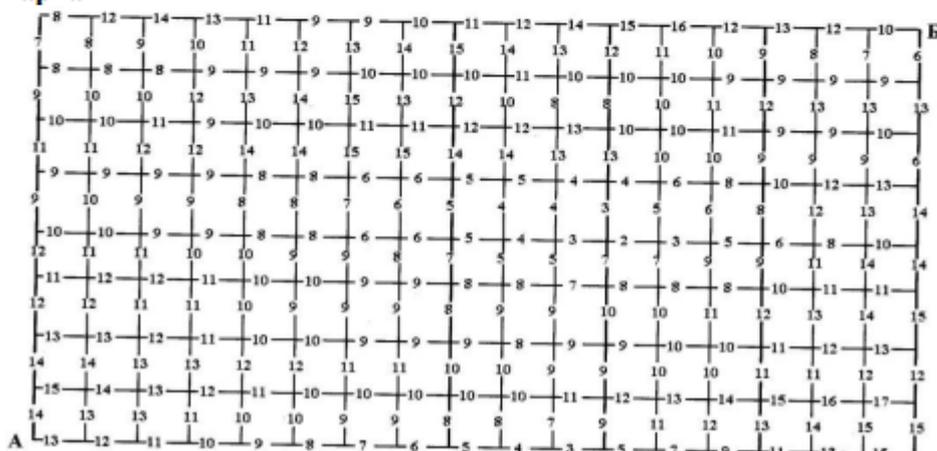
Вариант	1	2	3	4
f4(1)	9	5	6	8
f3(2)	10	10	7	7
f2(3)	7	5	8	9
f1(4)	10	9	13	15

**Задание 2.** Из пункта А в пункт В необходимо проложить автомобильную трассу по самому экономичному пути.

**Вариант 1**



**Вариант 2**



**Форма отчета:** тетрадь, защита работы.

## Практическое занятие № 17-18

**Тема:** Нахождение кратчайших путей на графе

**Цель:** отработать навыки нахождения кратчайшего пути на графе

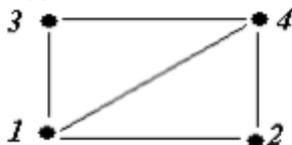
**Оборудование:** тетрадь, ручка

**Методические указания:** приступите к выполнению задания

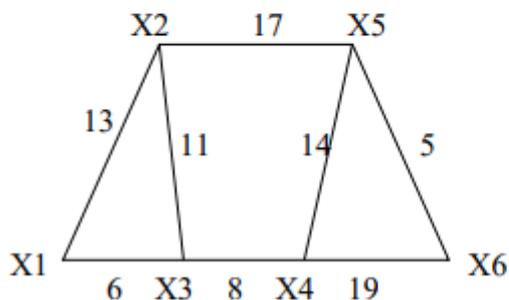
**Ход выполнения:**

**1 вариант**

**Задача 1.** Составить матрицы инцидентности и смежности для графа:

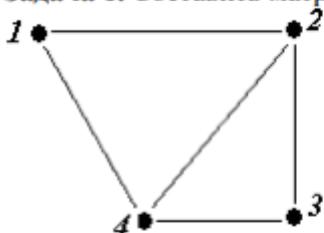


**Задача 2.** На представленном графе найдите: а) минимальный остов дерева, б) найдите кратчайший путь от начальной точки X1 до всех остальных точек.

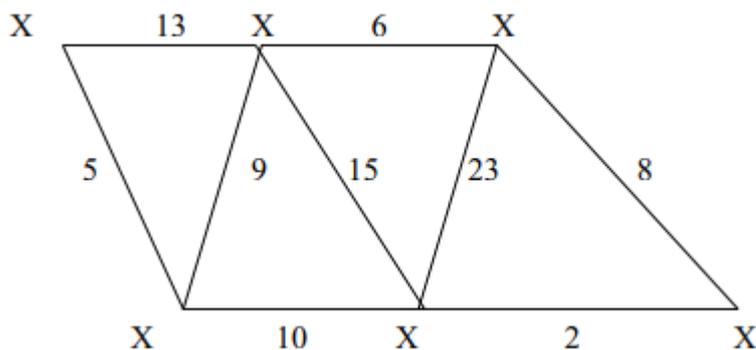


**2 вариант**

**Задача 1.** Составить матрицы инцидентности и смежности для графа:



**Задача 2.** На представленном графе найдите: а) минимальный остов дерева, б) найдите кратчайший путь от начальной точки X1 до всех остальных точек.



**Форма отчета:** тетрадь, защита работы.

## Практическое занятие № 19-23

**Тема:** Составление систем уравнений Колмогорова. Нахождение финальных вероятностей

**Цель:** отработать и закрепить умения составлять системы уравнений Колмогорова, отработать и закрепить умения находить финальные вероятности, отработать и закрепить умения определить основные показатели СМО.

**Оборудование:** тетрадь, ручка

**Методические указания:** приступите к выполнению задания

**Ход выполнения:**

**Форма отчета:** тетрадь, защита работы.

## Практическое занятие № 24

**Тема:** нахождение характеристик простейших систем массового обслуживания

**Цель:** учиться оценивать надежность простейших систем методом Монте-Карло; научиться рассчитывать СМО с отказами методом Монте-Карло.

**Оборудование:** тетрадь, ручка

**Методические указания:** приступите к выполнению задания

**Ход выполнения:**

### Вариант 1

1. Система состоит из двух блоков, соединенных последовательно. Первый блок содержит три элемента: А, В, С, а второй- два элемента: D, E. Элементы каждого блока соединены параллельно.
  - а) Найти методом Монте-Карло оценку  $P^*$  надежности системы, зная вероятности безотказной работы элементов:  $P(A)=0,8$ ;  $P(B)=0,9$ ;  $P(C)=0,85$ ;  $P(D)=0,7$ ;  $P(E)=0,6$ ;
  - б) найти абсолютную погрешность  $|P-P^*|$ , где P- надежность системы, вычисленная аналитически. Произвести 15 испытаний.
2. В двухканальную систему массового обслуживания с отказом поступает пуассоновский поток заявок. Время между поступлениями двух последовательных заявок распределено по показательному закону  $f(\tau)=4e^{-4\tau}$ . Длительность обслуживания каждой заявки равна 1 мин. Найти методом Монте-Карло математическое ожидание  $a$  числа обслуженных заявок за время  $T=8$  мин.

### Вариант 2

1. Система состоит из двух блоков, соединенных последовательно. Первый блок содержит два элемента: А, В, второй- три элемента: С, D, E. Элементы первого и второго блоков соединены параллельно.
  - а) Найти методом Монте-Карло оценку  $P^*$  надежности системы, зная вероятности безотказной работы элементов:  $P(A)=0,8$ ;  $P(B)=0,9$ ;  $P(C)=0,7$ ;  $P(D)=0,75$ ;  $P(E)=0,8$ ;
  - б) найти абсолютную погрешность  $|P-P^*|$ , где P - надежность системы, вычисленная аналитически. Произвести 15 испытаний.
2. В трехканальную СМО с отказами поступает пуассоновский поток заявок. Время между моментами поступления двух последовательных заявок распределено по закону  $f(\tau)=0,8e^{-0,8\tau}$ ; время обслуживания заявок 1,5 мин. Найти методом Монте-Карло математическое ожидание  $a$  числа обслуженных заявок за время  $T=10$  мин.

**Форма отчета:** тетрадь, защита работы.

## Практическое занятие № 25

**Тема:** построение прогнозов

**Цель:** научиться применять МНК для линейного сглаживания данные, научиться сглаживать данные с помощью квадратичной функции

**Оборудование:** тетрадь, ручка.

**Методические указания:** приступите к выполнению задания

**Ход выполнения:**

**Задание 1.** С помощью МНК подобрать параметры  $a$  и  $b$  линейной функции  $y = ax + b$ , приближенно описывающей следующие опытные данные. Построить полученную прямую и исходные точки в одной системе координат.

вариант							
1	$x_i$	1	3	4	5	6	8
	$y_i$	6	4	4	2	3	2
2	$x_i$	2	3	4	5	7	8
	$y_i$	1	3	4	6	6	9
3	$x_i$	1	2	4	6	7	8
	$y_i$	7	6	4	5	3	3
4	$x_i$	2	3	4	5	7	8
	$y_i$	2	6	6	7	8	10

**Задание 2.** С помощью МНК подобрать параметры  $a$  и  $b$  квадратичной функции  $y = a^2x + bx + c$ , приближенно описывающей следующие опытные данные. Построить полученную линию и исходные точки в одной системе координат.

вариант							
1	$x_i$	2	2,5	3	3,5	4	4,5
	$y_i$	4,2	2,5	6,2	7,5	2,6	1
2	$x_i$	1	1,5	2	2,6	2,9	3,1
	$y_i$	2,6	5,6	4,3	1,6	2,6	3,4
3	$x_i$	2	3	3,6	3,8	4,2	4,6
	$y_i$	0	2,3	2,5	2,9	1	4,5
4	$x_i$	5	5,5	6	6,5	7	7,5

**Форма отчета:** тетрадь, защита работы.

### Практическое занятие № 26-27

**Тема:** алгоритм поиска решения матричной антагонистической игры

**Цель:** научиться выбирать оптимальную стратегию игры

**Оборудование:** тетрадь, ручка

**Методические указания:** приступите к выполнению задания

**Ход выполнения:**

Борьба двух предприятий за рынок в регионе (N – номер варианта)

Две компании, занимающиеся производством антивирусного программного обеспечения, практически полностью делят рынок некоторого региона. Разрабатывая новую версию программного продукта для мобильных телефонов, каждая из компаний может использовать один из четырех вариантов продвижения нового программного продукта на рынок, который влияет на конечную стоимость продукции.

В зависимости от сделанного выбора компании могут установить цену реализации единицы продукции на уровне 25, 22, 19 и 16 условных единиц соответственно. Соотношение цен реализации и себестоимость представлены в таблице:

Вариант продвижения нового продукта	Цена реализации единицы продукции, у.е.	Полная себестоимость единицы продукции, у.е.	
		Компания А	Компания В
1	25	17	$21 - 0,1 * N$
2	22	15	$10 + 0,1 * N$
3	19	$10 + 0,1 * N$	10

N – номер варианта, предложенный преподавателем.

В результате маркетингового исследования рынка была определена функция спроса на программные продукты:

$$Y = 20 - 0,5 * X,$$

где Y – количество продукции, которое будет реализовано в регионе (тыс. ед.), а X – средняя цена продукции компаний, у.е.

Значения долей продукции, реализованной компанией А, зависят от соотношения цен на продукцию компании А и компании В. Маркетинговое исследование позволило установить эту зависимость:

Цена реализации 1 ед. продукции, у.е.		Доля реализованной продукции компании А
Компания А	Компания В	
25	25	0,31
25	22	0,33
25	19	0,25
25	16	0,2
22	25	0,4
22	22	0,35
22	19	0,32
22	16	0,28
19	25	0,52
19	22	0,48
19	19	0,4
19	16	0,35
16	25	0,6

16	22	0,58
16	19	0,55
16	16	0,5

1. Существует ли в данной задаче ситуация равновесия при выборе варианта продвижения продукта на рынок обоими компаниями?
2. Существуют ли варианты, которые компании заведомо не будут выбирать вследствие невыгодности?
3. Сколько продукции будет реализовано в ситуации равновесия? Какая компания получит больше прибыль в ситуации равновесия? Какая компания будет иметь большую долю рынка в ситуации равновесия? Дайте краткую экономическую интерпретацию результатов решения задачи.

**Форма отчета:** тетрадь, защита работы.

### Практическое занятие № 28

**Тема:** выбор оптимального решения с помощью дерева решений

**Цель:** научиться производить выбор оптимального решения с помощью дерева решений

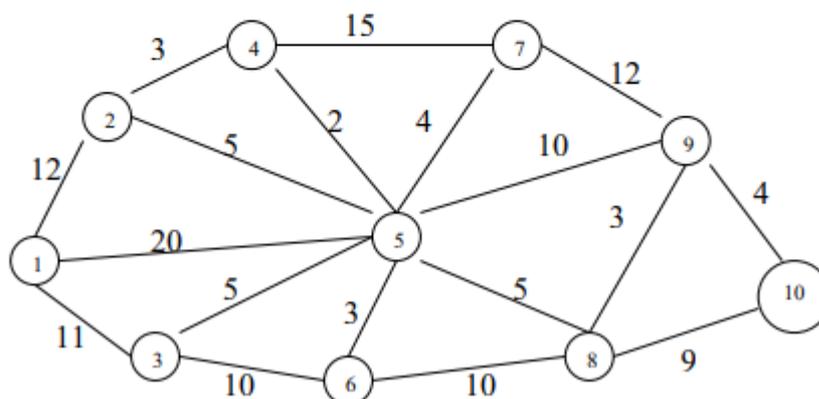
**Оборудование:** тетрадь, ручка

**Методические указания:** приступите к выполнению задания

**Ход выполнения:**

**Задача 1.** Планируется работа двух отраслей производства А и В на 4 года. Количество  $x$  средств, вложенных в отрасль А, позволяет получить доход  $5x$  и уменьшается до  $0,1x$ . Количество  $y$  средств, вложенных в отрасль В, позволяет получить доход  $3y$  и уменьшается до  $0,5y$ . Необходимо распределить выделенные ресурсы в количестве  $S_0 = 1100$  единиц между отраслями по годам планируемого периода для получения максимальной прибыли за весь период.

**Задача 2.** По заданной схеме, соединяющей 10 точек, найти кратчайшее расстояние от 1 точки до 10.



**Форма отчета:** тетрадь, защита работы.

### Практическое занятие № 29-31

**Тема:** Моделирование прогноза

**Цель:** изучение возможностей и формирование умения использования универсальной компьютерной технологии для решения задач выявления тенденций и прогнозирования развития процесса на основе моделирования рядов динамики (с помощью табличного процессора Excel)

**Оборудование:** тетрадь, ручка, ПК

**Методические указания:** приступите к выполнению задания

**Ход выполнения:**

## 4. ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ

### 4.1 Основные печатные и (или) электронные издания:

О-1. Зубкова, Т. М. Технология разработки программного обеспечения / Т. М. Зубкова. — 3-е изд., стер. — Санкт-Петербург: Лань, 2023. — 252 с. — ISBN 978-5-507-45571-3. — Текст: электронный // Лань: электронно-библиотечная система. — URL: <https://e.lanbook.com/book/276419> (дата обращения: 03.05.2025). — Режим доступа: для авториз. пользователей.

#### **4.2 Дополнительные печатные и (или) электронные издания (электронные ресурсы):**

Д-1. Немцова, Т.И. Практикум по информатике: учебное пособие / Т.И. Немцова, Ю.В. Назарова. — М: ИД "ФОРУМ"-ИНФРА-М, 2009. — 437 с.

**ЛИСТ ИЗМЕНЕНИЙ И ДОПОЛНЕНИЙ, ВНЕСЕННЫХ В  
МЕТОДИЧЕСКИЕ УКАЗАНИЯ**

<b>№ изменения, дата внесения, № страницы с изменением</b>	
<b>Было</b>	<b>Стало</b>
<b>Основание:</b>	
<b>Подпись лица, внесшего изменения</b>	

## Приложение А

### Перечень вариантов к практическим работам № 1-6

1. Создание экспертной информационной системы
2. Разработка системы электронного документооборота
3. Разработка электронного магазина для предприятия
4. Разработка корпоративной сети предприятия
5. Разработка информационной подсистемы учета выпуска продукции фермерского хозяйства
6. Разработка информационной подсистемы автоматизированной обработки документов коммерческого предприятия
  7. Разработка подсистемы управления кадрами предприятия
  8. Разработка подсистемы учета операций по импорту товаров
  9. Разработка системы автоматизации учета расчетов за проживание в общежитии
  10. Разработка системы автоматизации учета реализации и затрат на доставку мебели
  11. Разработка подсистемы учета дебиторов банка
  12. Разработка информационной подсистемы интернет-магазина
  13. Разработка информационной подсистемы банкомата
  14. Разработка информационной системы по организации учебного процесса
  15. Разработка подсистемы регистрации командировочных удостоверений в информационной системе
16. Разработка ИС автотранспортного предприятия
17. Разработка ИС учета договоров и контроля за их исполнением
18. Разработка ИС учета и оптимизации транспортных расходов на предприятии
19. Разработка информационной подсистемы страховой фирмы
20. Разработка ИС учета материальных ресурсов предприятия
21. Разработка подсистемы автоматизации складского учета
22. Разработка подсистемы автоматизации учета платежей по договорам
23. Разработка подсистемы учета реализации товаров в оптовой торговле
24. Разработка системы регистрации и обработки медицинской информации на примере тестов на артериальное давление и анализы крови
25. Разработка ИС учета обмена валют
26. Разработка информационной системы склада косметики и парфюмерии
27. Разработка ИС учета запасов предприятия
28. Разработка ИС ведения реестра акционеров в банке
29. Разработка ИС кинотеатра
30. Разработка ИС библиотеки
31. Разработка ИС аэропорта
32. Разработка ИС автовокзала



\_\_\_\_\_/\_\_\_\_\_/

подпись

расшифровка

«\_\_» \_\_\_\_\_ 20\_\_ г.

## СОДЕРЖАНИЕ

1	ОБЩИЕ СВЕДЕНИЯ.....	180
1.1	Полное наименование системы и ее условное обозначение .....	180
1.2	Номер договора.....	180
1.3	Наименования организации-заказчика и организации-исполнителя .....	180
1.4	Перечень документов, на основании которых создается система .....	181
1.5	Плановые сроки начала и окончания работы по созданию системы.....	181
1.6	Порядок оформления и предъявления заказчику результатов работ по созданию системы 181	
1.7	Перечень нормативно-технических документов, методических материалов, использованных при разработке ТЗ.....	181
1.8	Определения, обозначения и сокращения.....	181
2	НАЗНАЧЕНИЕ И ЦЕЛИ СОЗДАНИЯ СИСТЕМЫ .....	183
2.1	Назначение системы.....	183
2.2	Цели создания системы.....	183
3	ХАРАКТЕРИСТИКА ОБЪЕКТА АВТОМАТИЗАЦИИ.....	184
3.1	Краткие сведения об объекте автоматизации или ссылки на документы, содержащие такую информацию.....	184
3.2	Сведения об условиях эксплуатации объекта автоматизации и характеристиках окружающей среды.....	184
4	ТРЕБОВАНИЯ К СИСТЕМЕ.....	185
4.1	Требования к системе в целом .....	185
4.2	Требования к функциям, выполняемым системой.....	185
4.3	Требования к видам обеспечения .....	186
4.3.1	Требования к математическому обеспечению системы .....	186
4.3.2	Требования информационному обеспечению системы .....	186
4.3.3	Требования к лингвистическому обеспечению системы.....	186
4.3.4	Требования к методическому обеспечению системы .....	187
4.3.5	Требования организационному обеспечению системы .....	187
4.3.6	Требования к правовому обеспечению системы .....	187
4.3.7	Требования к программному обеспечению системы .....	187
4.3.8	Требования к техническому обеспечению .....	188
4.3.9	Требования к эргономическому обеспечению .....	188
5	СОСТАВ И СОДЕРЖАНИЕ РАБОТ ПО СОЗДАНИЮ СИСТЕМЫ .....	189
6	ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ.....	191
7	ТРЕБОВАНИЯ К СОСТАВУ И СОДЕРЖАНИЮ РАБОТ ПО ПОДГОТОВКЕ ОБЪЕКТА АВТОМАТИЗАЦИИ К ВВОДУ СИСТЕМЫ В ДЕЙСТВИЕ.....	192
8	ТРЕБОВАНИЯ К ДОКУМЕНТИРОВАНИЮ.....	193
9	ИСТОЧНИКИ РАЗРАБОТКИ .....	194

## 1 ОБЩИЕ СВЕДЕНИЯ

### 1.1 Полное наименование системы и ее условное обозначение

Полное наименование системы: \_\_\_\_\_.

Условное обозначение: \_\_\_\_\_.

### 1.2 Номер договора

Настоящее Техническое задание разработано в рамках выполнения работ по договору № \_\_\_\_\_, заключенному «\_\_» \_\_\_\_\_ 20\_\_ года.

### 1.3 Наименования организации-заказчика и организации-исполнителя

**Заказчик:** \_\_\_\_\_

Место нахождения: 344000, г. Ростов-на-Дону, Иванушкино, 15

Телефон: +7 863 256-08-00

Банковские реквизиты: ООО «Торгаш», ИНН 8495037287, Р/Сч № 40292710471192030000 в АКБ Сбербанк России, БИТ 194026400, Корр. Счет № 38192047506172030000

**Исполнитель:** \_\_\_\_\_

Место нахождения: 344113, г. Ростов-на-Дону, ул. Космонавтов 27/3

Телефон: +7 999 693-24-11

Банковские реквизиты: ООО «Техник», ИНН 8496031111, Р/Сч №40382710471192055111 в АКБ Сбербанк России, БИТ 194026111, Корр. Счет № 37792047506172034111

#### **1.4 Перечень документов, на основании которых создается система**

Система создается на основании договора № \_\_\_\_\_, от «\_\_\_» \_\_\_\_\_ 20\_\_ года.

#### **1.5 Плановые сроки начала и окончания работы по созданию системы**

Плановый срок начала работ – 1 день после заключения договора. Плановый срок окончания работ – «\_\_\_» \_\_\_\_\_ 20\_\_ года.

#### **1.6 Порядок оформления и предъявления заказчику результатов работ по созданию системы**

Система передается в виде функционирующего комплекса на базе средств вычислительной техники Заказчика и Исполнителя в сроки, установленные договором. Приемка системы осуществляется комиссией в составе уполномоченных представителей Заказчика и Исполнителя. Порядок предъявления системы, ее испытаний и окончательной приемки определен в п.6 настоящего Технического задания. Сдача разработанного Исполнителем комплекта документации определена на I этапе и обеспечивается согласно п.8 настоящего Технического задания.

#### **1.7 Перечень нормативно-технических документов, методических материалов, использованных при разработке ТЗ**

При разработке автоматизированной системы и создании проектно-эксплуатационной документации Исполнитель должен руководствоваться требованиями следующих нормативных документов:

- договор № \_\_\_\_\_, от «\_\_\_» \_\_\_\_\_ 20\_\_ года;
- ГОСТ 34.602-89. Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы;
- ГОСТ 34.601-90. Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания;
- ГОСТ 34.201-89. Информационная технология. Комплекс стандартов на автоматизированные системы. Виды, комплексность и обозначение документов при создании автоматизированных систем.

#### **1.8 Определения, обозначения и сокращения**

Таблица 1 – Определения, обозначения и сокращения

№	Сокращение	Расшифровка
1	НСД	Несанкционированный доступ
2	ИТ	Информационные технологии
3	ЭВМ	Электронно-вычислительная машина
4	ОС	Операционная система
5	СУБД	Система управления базами данных
6	ИС	Информационная система

## **2 НАЗНАЧЕНИЕ И ЦЕЛИ СОЗДАНИЯ СИСТЕМЫ**

### **2.1 Назначение системы**

АИС \_\_\_\_\_ предназначена для автоматизации процессов в \_\_\_\_\_:

- формирование единой базы торговой компании;
- обеспечение записи покупок;
- мониторинг в режиме реального времени всех товаров, ожидающих доставки.

### **2.2 Цели создания системы**

Основными целями проекта являются:

- возможность регистрации/изменения/удаления сотрудников;
- возможность поиска сотрудника по его ФИО, должности;
- возможность добавления/изменения/удаления клиентов со скидкой;
- обеспечение записи поступления новых товаров;
- регистрация заказов товаров от клиентов;
- возможность выводить отчеты о продажах/заказах/доставке.

### 3 ХАРАКТЕРИСТИКА ОБЪЕКТА АВТОМАТИЗАЦИИ

#### 3.1 Краткие сведения об объекту автоматизации или ссылки на документы, содержащие такую информацию

Торговая компания «\_\_\_\_\_» работает с 2006 года и обладает большим штатом опытных сотрудников. Принцип работы торговой компании заключается в постоянном развитии и освоении новых сфер деятельности, обеспечивающих разностороннее обслуживание покупателей, индивидуальный подход к каждому клиенту. Качество продукции и приемлемые цены – основной критерий выбора поставщиков. Весь товар имеет сертификат качества, действует гарантия производителя, дифференцированная система скидок, любая форма оплаты.

Торговая компания «\_\_\_\_\_» предлагает:

- продажу товаров с проверенными поставщиками;
- низкие цены;
- большой ассортимент товаров;
- качественную продукцию;
- быструю доставку.

В торговой компании получение товара производится путем доставки или самостоятельно на пункте выдачи.

#### 3.2 Сведения об условиях эксплуатации объекта автоматизации и характеристиках окружающей среды

АИС «\_\_\_\_\_» должна эксплуатироваться на операционных системах Microsoft Windows XP, Microsoft Windows Vista, Microsoft Windows 7 и выше, на которых установлен программный продукт.

Серверная часть ИС «\_\_\_\_\_» должна работать на серверах под управлением ОС Linux или Windows с установленным веб-сервером apache с поддержкой php, а также СУБД MySQL.

## 4 ТРЕБОВАНИЯ К СИСТЕМЕ

### 4.1 Требования к системе в целом

Система должна быть разделена на два контура, информационно связанные между собой, но разделенных по типу информационного пространства. Информационное пространство АИС «\_\_\_\_\_» разделяется на следующие контуры или подсистемы, имеющие уровень доступа:

– контур публичного доступа. Разделы АИС «\_\_\_\_\_», к функциональности которых имеют доступ все пользователи АИС «\_\_\_\_\_», как зарегистрированные, так и не зарегистрированные;

– контур персонифицированного доступа. Разделы и сервисы АИС «\_\_\_\_\_», к функциональности и/или информационному наполнению которых имеют доступ пользователи АИС «\_\_\_\_\_», прошедшие процедуру регистрации.

Каждый пользователь должен функционировать в рамках своего информационного пространства. Доступ любого пользователя к Системе должен осуществляться в области подпространства публичного доступа.

Для получения доступа в пространство персонифицированного доступа, пользователь должен пройти процедуру аутентификации для предотвращения НСД. В случае успешной аутентификации пользователя всего его действия в системе будут однозначно ассоциироваться с его учетной записью.

Процедура аутентификации пользователя Системы должна выполняться с помощью логина и пароль пользователя.

### 4.2 Требования к функциям, выполняемым системой

Система должна осуществлять следующие функции.

1 Функции редактирования данных. Перечислить конкретные функции, которые должна выполнять система.

2 Функции получения информации из информационного хранилища или поисковые функции. Перечислить конкретные функции, которые должна выполнять система.

3 Функции безопасности. Существуют 3 составляющие функции безопасности: защита доступности данных, защита целостности данных, защита конфиденциальности информации. Перечислить конкретные функции, которые должна выполнять система.

4 Расчетные функции. Перечислить конкретные функции, которые должна выполнять система.

5 Технологические функции. Перечислить конкретные функции, которые должна выполнять система.

6 Аналитические функции. **Перечислить конкретные функции, которые должна выполнять система.**

### **4.3 Требования к видам обеспечения**

#### **4.3.1 Требования к математическому обеспечению системы**

Требования не предъявляются.

#### **4.3.2 Требования информационному обеспечению системы**

Состав, структура и способы организации данных в системе должны быть определены на этапе технического проектирования.

Уровень хранения данных в системе должен быть построен на основе современных реляционных или объектно-реляционных СУБД. Для обеспечения целостности данных должны использоваться встроенные механизмы СУБД.

Средства СУБД, а также средства используемых операционных систем должны обеспечивать документирование и протоколирование обрабатываемой в системе информации.

Структура базы данных должна поддерживать кодирование хранимой и обрабатываемой информации в соответствии с общероссийскими классификаторами (там, где они применимы).

Доступ к данным должен быть предоставлен только авторизованным пользователям с учетом их служебных полномочий, а также с учетом категории запрашиваемой информации.

Структура базы данных должна быть организована рациональным способом, исключающим единовременную полную выгрузку информации, содержащейся в базе данных системы.

Технические средства, обеспечивающие хранение информации, должны использовать современные технологии, позволяющие обеспечить повышенную надежность хранения данных и оперативную замену оборудования (распределенная избыточная запись/считывание данных; зеркалирование; независимые дисковые массивы; кластеризация).

При проектировании и развертывании системы необходимо рассмотреть возможность использования накопленной информации из уже функционирующих информационных систем.

Перечень функционирующих информационных систем приведен в разделе 3 настоящего документа.

Требования к процедуре придания юридической силы документам, продуцируемым техническими средствами АС (в соответствии с ГОСТ 6.10.4) не предъявляются.

#### **4.3.3 Требования к лингвистическому обеспечению системы**

Все обозначения, названия элементов управления Системы, тексты должны быть изложены на русском языке без применения терминов, непонятных пользователю.

#### **4.3.4 Требования к методическому обеспечению системы**

Требования не предъявляются.

#### **4.3.5 Требования организационному обеспечению системы**

В ходе выполнения работ должно обеспечиваться постоянное взаимодействие между сторонами, для чего ими должны быть сформированы рабочие группы по данному проекту. Члены рабочих групп должны иметь необходимый уровень компетенции, в том числе, для принятия оперативных решений по вопросам разработки.

К работе с системой должны допускаться сотрудники, имеющие навыки работы на персональном компьютере, ознакомленные с правилами эксплуатации и прошедшие обучение работы с системой.

#### **4.3.6 Требования к правовому обеспечению системы**

Главной целью правового обеспечения является укрепление законности. В состав правового обеспечения входят законы, указы, постановления государственных органов власти, приказы, инструкции и другие нормативные документы министерств, ведомств, местных органов власти.

Специалисты и другие пользователи, работающие с системой, прежде всего, должны знать действующие в стране законы, регламентирующие области работ, с которыми они соприкасаются.

Им должны быть хорошо известны основные положения таких законов Российской Федерации, как:

- ФЗ «Об информации, информационных технологиях и о защите информации» от 27 июля 2006 г. № 149-ФЗ;
- ФЗ «О стандартизации в РФ» от 29 июня 2015 г. № 162-ФЗ;
- ФЗ «О техническом регулировании» от 27 декабря 2002 г. № 184-ФЗ;
- Гражданский кодекс Российской Федерации (часть четвертая) от 18.12.2006 № 230-ФЗ;
- ФЗ «Об электронной цифровой подписи» от 06 апреля 2011 г. № 63-ФЗ;
- Указ Президента РФ от 06.03.1997 N 188 (ред. от 13.07.2015) «Об утверждении Перечня сведений конфиденциального характера».

#### **4.3.7 Требования к программному обеспечению системы**

Состав программных средств:

- операционная система семейства Windows не ниже 7;
- реляционная СУБД Microsoft Access не ниже 2007 года;

- Microsoft .NET Framework не ниже 4.0;
- драйверы внешних устройств (мышь, клавиатура, принтер).

#### **4.3.8 Требования к техническому обеспечению**

Состав технических средств:

- 32/64-разрядный процессор с частотой выше 2.9 ГГц;
- видеокарта не менее 1 Гб;
- оперативная память объемом не менее 8 Гб;
- свободное дисковое пространство не менее 5000 Мб;
- клавиатура для ввода информации, мышь и монитор;

#### **4.3.9 Требования к эргономическому обеспечению**

Все визуальные интерфейсы ИС «\_\_\_\_\_» должны быть выполнены согласно современным стандартам в области разработки интерфейсов. Все существенно отличающиеся дизайн-макеты должны быть в обязательном порядке утверждены у Заказчика в рабочем порядке на этапе их графического представления.

## 5 СОСТАВ И СОДЕРЖАНИЕ РАБОТ ПО СОЗДАНИЮ СИСТЕМЫ

Состав и содержание работ по созданию системы приведены на рисунке 1.

	Название задачи	Длительность	Начало	Окончание	Наз рес
0	<b>Проектирование и разработка БД</b>	<b>22 дней</b>	<b>Сб 20.04.19</b>	<b>Пт 17.05.19</b>	
1	<b>1. Разработка ТЗ</b>	<b>5 дней</b>	<b>Сб 20.04.19</b>	<b>Чт 25.04.19</b>	
2	Постановка задачи	1 день	Сб 20.04.19	Сб 20.04.19	Ме
3	Определение и уточнение требований к техническим средствам	1 день	Пн 22.04.19	Пн 22.04.19	Ме
4	Определение требований к системе	1 день	Вт 23.04.19	Вт 23.04.19	Ме
5	Определение стадий, этапов и сроков разработки системы и документации	1 день	Ср 24.04.19	Ср 24.04.19	Ме
6	Согласование и утверждение ТЗ	1 день	Чт 25.04.19	Чт 25.04.19	Ме
7	<b>2. Рабочее проектирование</b>	<b>16 дней</b>	<b>Пт 26.04.19</b>	<b>Чт 16.05.19</b>	
8	Разработка системы	12 дней	Пт 26.04.19	Сб 11.05.19	Ме
9	Разработка программной документации	2 дней	Пн 13.05.19	Вт 14.05.19	Ме
10	Испытания системы	2 дней	Ср 15.05.19	Чт 16.05.19	Ме
11	<b>3. Внедрение</b>	<b>1 день</b>	<b>Пт 17.05.19</b>	<b>Пт 17.05.19</b>	
12	Подготовка и передача системы	1 день	Пт 17.05.19	Пт 17.05.19	Ме

Рисунок 3 – Календарный график

Информационные объекты ИС сведены в таблицу 3.

Таблица 3 – Информационные объекты

Объект	Атрибуты	Ключевой атрибут
Заказы	ИД заказа, ИД сотрудника, ИД клиента, Дата размещения, Дата отгрузки, ИД грузоотправителя, Получатель, Адрес получателя, Город получателя, Область получателя, Индекс доставки, Страна или регион доставки, Стоимость доставки, Налоги, Тип платежа, Дата оплаты, Примечания, Налоговая ставка, Налоговый статус, ИД состояния	ИД заказа
Сотрудники	ИД сотрудника, Организация, Фамилия, Имя, Адрес электронной почты, Должность, Рабочий телефон, Домашний телефон, Мобильный телефон, Факс, Адрес, Город, Область, Индекс, Страна или регион, Веб-страница, Примечания, Вложения	ИД сотрудника
Заказы на приобретение	ИД заказа на приобретение, ИД поставщика, Создано, Дата отправки, Дата создания, ИД состояния, Расчетная дата, Стоимость доставки, Налоги, Дата платежа, Сумма, Форма оплаты, Примечания, Утверждено, Дата утверждения, Отправлено	ИД заказа на приобретение

## Окончание таблицы 3

Доставка	ИД доставка, Организация, Фамилия, Имя, Адрес электронной почты, Должность, Рабочий телефон, Домашний телефон, Мобильный телефон, Факс, Адрес, Город, Область, Индекс, Страна или регион, Веб-страница, Примечания, Вложения	ИД доставка
Сведения о заказе	ИД сведения о заказе, ИД заказа, ИД товара, Количество, Цена за единицу, Скидка, ИД состояния, Дата размещения, ИД заказа на приобретение, Инвентарный номер	ИД сведения о заказе
Налоговый статус заказов	ИД налоговый статус заказов, Название налогового статуса	ИД налоговый статус заказов
Сведения о заказе на приобретение	ИД сведений о заказе на приобретение, ИД заказа на приобретение, ИД товара, Количество, Стоимость единицы, Дата получения, Отправлено на склад, Инвентарный номер	ИД сведений о заказе на приобретение
Отчет о продажах	Группировка, Вывод, Должность, Источник строк фильтра, По умолчанию	Группировка
Роли	ИД роли, Имя роли	ИД роли
Роли сотрудников	ИД сотрудника, ИД роли	ИД роли сотрудника
Состояние заказа на приобретение	ИД состояния, Состояние	ИД состояния
Состояние заказов	ИД состояния заказов, Название состояния	ИД состояния заказов
Состояние сведений о заказе	ИД состояния сведений о заказе, Название состояния	ИД состояния сведений о заказе
Операции с запасами	ИД операции, Тип операции, Дата создания операции, Дата изменения операции, ИД товара, Количество, ИД заказа на приобретение, ИД заказа клиента, Примечания	ИД операции
Счета	ИД счета, ИД заказа, Дата счета, Срок, Налог, Доставка, Остаток	ИД счета
Строки	ИД строки, Данные строки	ИД строки
Тип операций с запасами	ИД тип операций, Название типа	ИД тип операций
Товары	ИД поставщиков, ИД товара, Код товара, Наименование, Описание, Стандартная стоимость, Цена по прейскуранту, Минимальный запас, Желаемый запас, Количество в позиции, Поставки_прекращены, Минимальное количество для повтора заказа, Категория, Вложения	ИД товара
Поставщик	ИД поставщиков, Организация, Фамилия, Имя, Адрес электронной почты, Должность, Рабочий телефон, Домашний телефон, Мобильный телефон, Факс, Адрес, Город Область, Индекс, Страна или регион, Веб-страница, Примечания, Вложения	ИД поставщиков

## 6 ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ

Порядок и контроль приемки определяются Заказчиком.

Главным требованием к приемке является наличие правильно работающей ИС с тестовым примером и набора документации, предоставленного в печатном виде.

Сдача-приёмка работ производится поэтапно, в соответствии с рабочей программой и календарным планом, являющимися приложениями к договору № \_\_\_\_\_ от « \_\_\_\_ » \_\_\_\_\_ 20\_\_ года.

Приемочные испытания АИС « \_\_\_\_\_ » должны проводиться в соответствии с разработанной Исполнителем Программой и методикой испытаний, предварительно согласованной с Заказчиком. Результаты приемочных испытаний должны быть зафиксированы в Протоколе приемочных испытаний.

Протокол приемочных испытаний должен содержать заключение о соответствии АИС « \_\_\_\_\_ » Техническому заданию и выводы о возможности передачи АИС « \_\_\_\_\_ » в эксплуатацию.

По итогам испытаний должны быть устранены все замечания к работе АИС « \_\_\_\_\_ » и ее функциям, соответствующим образом доработан Технический проект АИС « \_\_\_\_\_ ».

Приемка АИС « \_\_\_\_\_ » в эксплуатацию оформляется Актом приемки АИС « \_\_\_\_\_ » в опытную эксплуатацию.

## 7 ТРЕБОВАНИЯ К СОСТАВУ И СОДЕРЖАНИЮ РАБОТ ПО ПОДГОТОВКЕ ОБЪЕКТА АВТОМАТИЗАЦИИ К ВВОДУ СИСТЕМЫ В ДЕЙСТВИЕ

В перечень основных мероприятий включают:

- приведение поступающей в систему информации (в соответствии с требованиями к информационному и лингвистическому обеспечению) к виду, пригодному для обработки с помощью ЭВМ;
- изменения, которые необходимо осуществить в объекте автоматизации;
- создание условий функционирования объекта автоматизации, при которых гарантируется соответствие создаваемой системы требованиям, содержащимся в ТЗ;
- определить подразделение и ответственных должностных лиц, ответственных за внедрение и проведение опытной эксплуатации АИС «\_\_\_\_\_»;
- обеспечить присутствие пользователей на обучении работе с системой, проводимом Исполнителем;
- обеспечить соответствие помещений и рабочих мест пользователей системы в соответствии с требованиями, изложенными в настоящем ЧТЗ;
- обеспечить выполнение требований, предъявляемых к программно-техническим средствам, на которых должно быть развернуто программное обеспечение АИС «\_\_\_\_\_»;
- совместно с Исполнителем подготовить план развертывания системы на технических средствах Заказчика;
- провести опытную эксплуатацию АИС «\_\_\_\_\_».

## 8 ТРЕБОВАНИЯ К ДОКУМЕНТИРОВАНИЮ

Виды, наименование, комплектность и обозначения документов, разрабатываемых на различных стадиях создания Программного комплекса, должны определяться в соответствии с ГОСТ 34.201-89 и согласованы с Заказчиком.

Таблица 4 – Виды предоставляемой документации

№	Стадия создания	Название документа	Дополнительные указания
1	ТЗ	Техническое задание на разработку	ГОСТ 34.602-89
2	ТП	Технический проект	ГОСТ 2.120-2013
3	РП	Руководство пользователя ИС	ГОСТ 2.610-2006 ГОСТ 34.201-89

## **9 ИСТОЧНИКИ РАЗРАБОТКИ**

Исходными документами для разработки настоящего технического задания и Системы являются действующие законодательные и нормативные правовые акты, в рамках которых функционирует объект автоматизации, нормативно-техническая документация Заказчика, ГОСТ 34.602-89, образцы рабочих документов, полученных в процессе обследования, информационные материалы и проектная документация на аналогичные автоматизированные системы.

## Приложение В

### Перечень вариантов к практическим работам № 2, 7, 8

- 1 Вывести на экран все целые числа от 100 до 200, кратные трем.
- 2 Вывести на экран все целые числа от  $a$  до  $b$ , кратные некоторому числу  $c$ .
- 3 Найти сумму положительных нечетных чисел, меньших 50.
- 4 Найти сумму целых положительных чисел из промежутка от  $a$  до  $b$ , кратных четырем.
- 5 Составить программу поиска четырехзначных чисел, которые при делении на 47 дают в остатке 43, а при делении на 43 дают в остатке 37.
- 6 Составить программу поиска четырехзначных чисел, которые при делении на 133 дают в остатке 125, а при делении на 134 дают в остатке 111.
- 7 Определить количество натуральных чисел из интервала от 100 до 500, сумма цифр которых равна 15. Подсказка.
- 8 Определить количество трехзначных натуральных чисел, сумма цифр которых равна целому числу  $n$  ( $0 < n \leq 27$ ).
- 9 Найти:
  - а) все двузначные числа, сумма квадратов цифр которых делится на 13;
  - б) все двузначные числа, обладающие следующим свойством: если к сумме цифр числа прибавить квадрат этой суммы, то получится снова искомое число.
- 10 Найти все двузначные числа, которые делятся на  $n$  или содержат цифру  $n$ .
- 11 Найти:
  - а) все трехзначные числа, чьи квадраты оканчиваются тремя цифрами, которые и составляют искомые числа;
  - б) все трехзначные числа, кратные семи и у которых сумма цифр также кратна семи.
- 12 Найти сумму целых положительных чисел, больших 30 и меньших 100, кратных трем и оканчивающихся на 2, 4 и 8.
- 13 Дано натуральное число.
  - а) Получить все его делители.
  - б) Найти сумму его делителей.
  - в) Найти сумму его четных делителей.
  - г) Определить количество его делителей.
  - д) Определить количество его нечетных делителей.
  - е) Определить количество его делителей. Сколько из них четных?
  - ж) Найти количество его делителей, больших  $d$ .
- 14 Даны вещественные числа  $a_1, a_2, \dots, a_{12}$ . Определить сумму тех из них, которые больше 10,75.
- 15 Даны натуральное число  $n$  и вещественные числа  $a_1, a_2, \dots, a_n$ . Определить сумму тех вещественных чисел, которые больше  $p$ .
- 16 Даны целые числа  $a_1, a_2, \dots, a_{10}$ . Определить сумму тех из них, которые являются четными.
- 17 Даны натуральное число  $m$  и целые числа  $a_1, a_2, \dots, a_m$ . Определить сумму тех целых чисел, которые кратны числу  $n$ .
- 18 Даны целые числа  $a_1, a_2, \dots, a_{20}$ . Найти сумму  $a_2 + a_4 + a_6 + \dots$ . Оператор цикла с шагом, отличным от 1 и  $-1$ , не использовать.
- 19 Даны вещественные числа  $a_1, a_2, \dots, a_{15}$ . Найти  $a_1 - a_3 + a_5 - \dots$ .
- 20 Даны натуральное число  $n$  и целые числа  $a_1, a_2, \dots, a_n$ . Получить:
  - а)  $a_1 - a_2 + a_3 - \dots$
  - б)  $a_1 + a_n$ ;
  - в)  $a_1 - a_2$ .
- 21 Известны данные о стоимости каждого товара из группы. Найти общую стоимость тех товаров, которые стоят дороже 1000 рублей (количество таких товаров неизвестно).
- 22 Известны данные о количестве страниц в каждой из нескольких газет и в каждом из нескольких журналов. Число страниц в газете не более 16. Найти общее число страниц во всех журналах (количество журналов неизвестно, но известно, что объем любого журнала превышает объем любой газеты).

23 Известны данные о количестве осадков, выпавших за каждый день месяца. Определить общее количество осадков, выпавших второго, четвертого и т. д. числа этого месяца. Оператор цикла с шагом, отличным от 1 и  $-1$ , не использовать.

24 Известно число детей, учащихся во всех первых классах, во всех вторых, ... и во всех одиннадцатых. Определить общее число детей, учащихся в первых, третьих, пятых и т. д. классах школы. Оператор цикла с шагом, отличным от 1 и  $-1$ , не использовать.

25 Известны оценки по информатике каждого ученика класса. Определить количество пятерок.

26 Известны данные о температуре воздуха в течение месяца. Определить, сколько раз температура опускалась ниже  $0\text{ C}$ .

27 Даны вещественные числа  $a_1, a_2, \dots, a_9$ . Определить количество тех из них, которые меньше 100.

28 Даны натуральное число  $n$  и целые числа  $a_1, a_2, \dots, a_n$ . Определить:

а) количество чисел  $a_i$ , которые больше  $p$ ;

б) количество чисел  $a_i$ , которые оканчиваются цифрой 5;

в) количество чисел  $a_i$ , которые кратны числу  $k$ .

29 Известны оценки по химии каждого ученика класса. Определить количество пятерок и количество двоек.

30 Известен год рождения каждого человека из группы. Определить число людей, родившихся до 1985 года, и число людей, родившихся после 1990 года.

31 Для каждой команды-участницы чемпионата по футболу известно ее количество выигрышей и количество проигрышей. Определить, сколько команд имеют больше выигрышей, чем проигрышей.

32 Известны оценки каждого студента из группы по двум экзаменам. Определить количество студентов группы, получивших на экзамене двойку.

33 Даны натуральное число  $n$  и вещественные числа  $a_1, a_2, \dots, a_n$ . Определить количество отрицательных и количество положительных вещественных чисел.

34 Даны натуральное число  $m$  и целые числа  $a_1, a_2, \dots, a_m$ . Определить количество чисел  $x_i$ , кратных трем, и количество чисел  $x_i$ , кратных семи.

35 Даны натуральное число  $n$  и целые числа  $a_1, a_2, \dots, a_n$ . Найти:

а) количество пар «соседних» чисел  $a_i$ , равных между собой;

б) количество пар «соседних» чисел  $a_i$ , равных нулю;

в) количество пар «соседних» чисел  $a_i$ , являющихся четными числами;

г) количество пар «соседних» чисел  $a_i$ , оканчивающихся на цифру 5.

36 Даны натуральное число  $n$  и вещественные числа  $x_1, x_2, \dots, x_n$ . Найти количество вещественных чисел, которые больше своих «соседей», т.е. предшествующего и последующего.

37 Дана последовательность ненулевых целых чисел. Определить, сколько раз в этой последовательности меняется знак. Например, в последовательности 10,  $-4$ , 12, 56,  $-4$  знак меняется 3 раза.

38 Задано  $n$  троек целых чисел  $a, b, c$  ( $a \leq b \leq c$ ). Определить, сколько троек может быть использовано для построения треугольника со сторонами  $a, b, c$ . Z2.26. В ходе хоккейного матча игроки обеих команд удалялись в общей сложности 24 раза. По каждому удалению известен номер команды удаленного игрока и продолжительность удаления (2, 5 или 10 мин.). Для каждой команды определить общее число удалений и общее время всех удалений.

39 Известны оценки каждого из учеников класса по физике. Посчитать количество пятерок, количество четверок, количество троек и количество двоек.

40 В чемпионате по футболу команде за выигрыш дается 3 очка, за проигрыш — 0, за ничью — 1. Известно число очков, полученных командой за каждую из проведенных игр. Определить количество выигрышей, количество проигрышей и количество ничьих.

41 Даны вещественные числа  $b_1, b_2, \dots, b_9$ . Определить среднее арифметическое тех из них, которые больше 10. Известно, что числа, большие 10, среди заданных имеются.

42 Даны натуральное число  $x$  и целые числа  $a_1, a_2, \dots, a_x$ . Определить среднее арифметическое тех чисел  $a_i$ , которые больше некоторого числа  $n$ . Известно, что числа, большие  $n$ , среди заданных имеются.

## Приложение Г

### Перечень вариантов к практической работе

#### Вариант №1.

```
main()
{float tr[4][2]; float xe;float ye; float xa, ya, xb, yb,xc, yc, xd, yd; float a, b, c, d, f, h; int
i;clrscr();printf("Введите координаты точек параллелограмма"); for (i=0; i<4; i++)
{printf("\nкоординаты: %d-й точки: x=",i+1); scanf("%f", &tr[i][0]); printf("y="); scanf("%f", &tr[i][1]);
} xa=tr[0][0]; ya=tr[0][1]; xb=tr[1][0]; yb=tr[1][1]; xc=tr[2][0]; yc=tr[2][1]; xd=tr[3][0]; yd=tr[3][1];
printf("%f, %f\n %f %f\n %f %f\n %f %f\n",xa,ya,xb,yb,xc,yc,xd,yd); a=sqrt((xa-xb)*(xa-xb)+(ya-
yb)*(ya-yb)); b=sqrt((xb-xc)*(xb-xc)+(yb-yc)*(yb-yc)); c=sqrt((xc-xd)*(xc-xd)+(yc-yd)*(yc-yd));
d=sqrt((xd-xa)*(xd-xa)+(yd-ya)*(yd-ya)); f=sqrt((xa-xc)*(xa-xc)+(ya-yc)*(ya-yc)); h=sqrt((xb-xd)*(xb-
xd)+(yb-yd)*(yb-yd)); if((a+b>f)&&(b+c>h)&&(d+c>f)&&(a+d>h)) { xe=(xa+xc)/2; ye=(ya+yc)/2;
printf("xe=%f, ye=%f",xe,ye); } else printf("Данный параллелограмм ((%f, %f),(%f, %f),(%f, %f),"
"%f, %f) вырожденный\n",xa,ya,xb,yb,xc,yc,xd,yd); getch(); }
```

#### Вариант №2.

```
main() { int n, m, r; float a[50][50]; float y[50]; float b; float s; int i,j,k; clrscr(); printf("vvedite dannie
matrici"); scanf(" %d %d", &m, &n); while( m<=1 && n<=1 ) { printf("Vi owiblis, povtorite vvod");
scanf(" %d %d", &m, &n); printf("n=", "m="); } for(i=0; i<n; i++) { printf("vvesti koord vekt"); scanf("%f",
&y[i]); } for(i=0; i<m; i++) for(j=0; j<n; j++) a[i][j]=pow(y[j], i); for(i=0; i<m; i++) for(j=0; j<n; j++)
printf("a[%d,%d]=%f\n", i,j,a[i][j]); }
```

#### Вариант №3.

```
#define L 100 main() { clrscr(); char s1[L]; char s2[L]; char seps[] = " \t\n"; char *token; printf("Enter
the sentence\n"); gets(s1); token= strtok(s1,seps); while (token !=NULL) { if (strstr(token, "аь") ||
strstr(token, "яь") || strstr(token, "уь") || strstr(token, "юь")) {strcat(s2, "не"); strcat(s2, token);} else
strcat(s2, token); strcat(s2, " "); token= strtok(NULL,seps); } printf("%s\n", s2); getch(); }
```

#### Вариант №4.

```
#define PI float form(int k, float, float, float); main() { float x; int k=1; float e; float s=0; printf("Enter the
X:\n"); scanf("%f", &x); while (x<-0 || x>PI) { printf("Error\n"); scanf("%f", &x); } printf("\nEnter the
e\n"); scanf("%f", &e); while (e<0) { printf("Error"); scanf("%f", &e); } printf("%f \n", form(k,s,x,e));
getch(); } float form(int k, float y, float x, float e) { float a; a=cos((2*k-1)*x)/(2*k-1) ; y=y+a;
printf("%f\n",y); if (a<e) return(y); else return(form(k+1,y, x, e)); }
```

#### Вариант №5.

```
main() { float min=999999; struct icx { char kultura[20]; float p38; float p57; float z38; float z57; } a; struct
abc { char kultura[20]; float izmen; } b; FILE *filein; clrscr(); if((filein=fopen("TABL1.txt","r"))==NULL)
{printf("owibka\n");exit(-1);}
while(fscanf(filein,"%s%f%f%f%f",&a.kultura,&a.p38,&a.p57,&a.z38,&a.z57)!=EOF) { if(a.z57-
a.z38<min) { min=a.z57-a.z38; strcpy(b.kultura, a.kultura); b.izmen = a.p57/a.p38*100 - 100; } printf(
"%15s|%10.2f|%10.2f|%10.2f|%10.2f\n",a.kultura, a.p38, a.p57, a.z38, a.z57); getch(); }
printf("Минимальное увеличение сбора культуры:\n"); printf("| Kultura |Izmenenie|\n");
printf("%13s|%9.2f\n", b.kultura, b.izmen); getch(); }
```

#### Вариант №6.

```
main() { float tr[3][2]; float x,y; float xa,ya,xb,yb,xc,yc; float a1,b1,c1,a2,b2,c2,a3,b3,c3,a4,b4,c4,a5,b5,c5;
float a,b,c; float r; int i; clrscr(); printf("Enter coordinats :\n"); for(i=0;i<3;i++) { printf("\ncoordinats:%d:
x=", i+1); scanf("%f",&r); tr[i][0]=r; printf("y="); scanf("%f", &r); tr[i][1]=r; xa=tr[0][0]; ya=tr[0][1];
xb=tr[1][0]; yb=tr[1][1]; xc=tr[2][0]; yc=tr[2][1]; a=sqrt((xa-xb)*(xa-xb)+(ya-yb)*(ya-yb)); b=sqrt((xb-
xc)*(xb-xc)+(yb-yc)*(yb-yc)); c=sqrt((xa-xc)*(xa-xc)+(ya-yc)*(ya-yc));
if(!(a+b>c)&&(b+c>a)&&(a+c>b)) printf("Triangle ((%f,%f), (%f,%f),(%f,%f)) virog\n",
```

```

xa,ya,xb,yb,xc,yc); else { a1=yb-ya; b1=xa-xb; c1=-a1*xa-b1*ya; a2=yc-yb; b2=xb-xc; c2=-a2*xb-b2*yb;
a3=yc-ya; b3=xa-xc; c3=-a3*xa-b3*ya; a4=b2; b4=-a2; c4=-a4*xa-b4*ya; a5=b3; b5=-a3; c5=-a5*xb-
b5*yb;      x=((-c4*b5)-(b4*(-c5)))/((a4*b5)-(b4*a5));      y=((a4*(-c5))-(-c4*a5))/((a4*b5)-(b4*a5));
printf("( (%f,%f), )\n", x,y); getch(); } }

```

#### Вариант №7.

```

void main () { const int size= 10; int a[size]; srand(time(NULL)); for (int i = 0; i < size; i++) a[i] = rand()
% 11 - 5; for (i = 0; i < size; i++) cout << a[i] << " "; for (i = 0; i < size; i++) for (int j = i+1; j < size; j++)
if (a[i] < a[j]) { int buf = a[i]; a[i] = a[j]; a[j] = buf; } cout << endl << endl; for (i = 0; i < size; i++) cout <<
a[i] << " "; getch(); return 0; }

```

#### Вариант №8.

```

float f(float z) { return pow(z,3)+6*pow(z,2)+6*z-7; } void main() { float a=-3.0, b=2.0, e=0.001, x;//
объявление переменных while (fabs(a-b)>=e) { if((f(a)>0&&f((a+b)/2)<0)||f(a)<0&&f((a+b)/2)>0))
b=(a+b)/2; else if ((f((a+b)/2)>0&&f(b)<0)||f((a+b)/2)<0&&f(b)>0)) a=(a+b)/2; else { printf("! Net kornej
!"); return; getch(); } } x=(a+b)/2; printf("x=%f F(x)=%f |a-b|=%f",x,f(x),fabs(a-b)); getch(); }

```

#### Вариант №9.

```

int main() { clrscr(); int i; float x[10], max, min; for (i=0;i<10;i++) { printf("x[%d]=",i+1); scanf(
"%f",&x[i]); } max=x[0]; min=x[0]; for(i=1;i<10;i++) { if (x[i]>max) max = x[i]; if (x[i]<min) min=x[i];
} x[0] = max+min; printf("\nmax=%f ",max); printf("\nmin=%f \n",min); for(i=0;i<10;i++)
printf("\nx[%d]=%f ",i+1,x[i]); getch(); return 0;}

```

#### Вариант №10.

```

void main() { cout<<"Sluchaino zapolnenii massiv 8x8:"<<endl; int mass[8][8] = {}; for (int i=0; i<=7;
i++) { for(int j=0; j<=7; j++) { mass[i][j]=rand()%500-100; if ((mass[i][j]<10) && (mass[i][j]>=0)) {
cout<<" "; } else if ((mass[i][j]>=10) && (mass[i][j]<=99)) { cout<<" "; } else if ((mass[i][j]>=100) &&
(mass[i][j]<=999)) { cout<<" "; } else if ((mass[i][j]<0) && (mass[i][j]>-10)) { cout<<" "; } else if
((mass[i][j]<=-10) && (mass[i][j]>=-99)) { cout<<" "; } cout<<mass[i][j]<<" "; } cout<<endl; } int s, max,
s1, k, l, t; s=0; for (int i=0; i<=7; i++) { s1=s1+abs(mass[i][0]); } for (int j=0; j<=7; j++) { for (int i=0;
i<=7; i++) { s=s+abs(mass[i][j]); } if (s>s1) { max=s; s1=max; k=j+1; } s=0; } l=mass[0][k-1]; for (int i=0;
i<=7; i++) { t=mass[i][k-1]; if (t<l) { l=t; } } cout<<"Maximal'naya summa elementov v
stolbike:"<<k<<endl<<"Minimal'noe znachenie v stolbike:"<<l<<endl; getch(); }

```