

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«КАРАЧАЕВО-ЧЕРКЕССКИЙ ГОСУДАРСТВЕННЫЙ  
УНИВЕРСИТЕТ ИМЕНИ У.Д. АЛИЕВА»**



Узденова А.М.

РАБОЧАЯ ТЕТРАДЬ

# **Проектирование информационных систем**

студента \_\_\_\_\_  
(номер группы, факультет)

\_\_\_\_\_  
(Ф.И.О.)

**Карачаевск-2021**

Печатается по решению редакционно-издательского совета Карачаево-Черкесского государственного университета им. У.Д. Алиева

**УДК 004.415.2**

**ББК 32.973.233-018я7**

**УЗ4**

Рабочая тетрадь предназначена для активизации самостоятельной работы студентов по изучению дисциплины «Проектирование информационных систем» направления подготовки 09.03.01 – Информатика и вычислительная техника.

**Узденова, А.М.** Рабочая тетрадь по курсу «Проектирование информационных систем». Учебно-методическое пособие. / А.М. Узденова. – Карачаевск: КЧГУ, 2021. - 168 с.

Рецензенты:

*М.Х. Уртенов*, д. ф.-м. н., проф.,

*Р.А. Бостанов*, к. ф.-м. н., доц.

© Узденова А.М., 2021

© Карачаево-Черкесский государственный университет им. У.Д. Алиева, 2021

# Предисловие

*Посвящается Узденову М.И.*

Перед Вами рабочая тетрадь «Проектирование информационных систем», разработанная специально для студентов направления подготовки бакалавров 09.03.01 Информатика и вычислительная техника.

Целью данного учебно-методического пособия является помощь в самостоятельной работе студента по освоению учебной дисциплины в аудитории и дома. Рабочая тетрадь может быть использована студентами в самостоятельном освоении теоретического материала и формировании практических умений и навыков, а также при подготовке к промежуточной аттестации.

Выполнение заданий рабочих тетрадей создает прочную базу для постижения и усвоения основного материала дисциплины и является одним из наиболее результативных видов самостоятельной работы студента.

Разделы рабочей тетради соответствуют логике расположения учебного материала в учебной программе. Каждая тема содержит необходимые теоретические сведения, методику выполнения заданий, задания для самостоятельного выполнения по вариантам и контрольные вопросы. Рабочая тетрадь также содержит области занесения отчета студентами о выполнении заданий.

Балльная оценка заданий в рабочей тетради позволяют студентам самим отслеживать свою успеваемость и ориентируют на успех. Тетрадь позволяет повысить эффективность обучения благодаря тому, что преподаватель может производить мониторинг освоения каждой темы в отдельности и курса в целом.

Желаю успехов в изучении проектирования информационных систем!

# Тема 1. Компоненты ввода и редактирования данных Delphi

**Цель:** Изучение компонентов ввода-вывода данных Delphi и формирование навыков их использования для создания форм. По завершению выполнения заданий данной темы студент должен уметь организовать ввод-вывод данных с помощью библиотеки визуальных компонентов (VCL) Delphi.


## Теоретические сведения

Практически любая программа предназначена для обработки данных. В процессе работы программа получает исходные данные и возвращает результат обработки. Наиболее часто используются два способа получения исходных данных: ввод данных пользователем или считывание данных, хранящихся на носителе информации. Результаты расчета могут отображаться на экране (в виде строк, таблиц, графиков, диаграмм и т.п.) либо записываться в файл. Информация, с которой работают программы, часто хранится в базах данных.

Компоненты Delphi, применяемые для ввода и редактирования информации, условно можно разделить на две группы: компоненты, не ориентированные на работу с базами данных, и компоненты, применяемые при работе с базами данных. А также не визуальные компоненты, предназначенные для работы с файлами: диалоговые окна открытия и сохранения файлов [14, 21].

## 1. Стандартные элементы интерфейса


### 1.1. Кнопки TButton

Свойства и методы компонента «кнопка» инкапсулированы в классе **TButton**  (см. рис. 1.1). Свойства кнопки:

- *Caption: TCaption* – строка текста, отображаемого на кнопке;
- *Enable: Boolean* – при установке данного свойства равным false кнопка становится недоступной, отображаясь серым цветом;
- *Visible: Boolean* – при установке данного свойства false кнопка становится невидимой.

Основное событие кнопки – *OnClick*, оно вызывается при нажатии на кнопку и используется для программирования реакции на нажатие [1, 6, 14, 21].


## 1.2. Надписи TLabel

Надписи используются для отображения на форме текста без возможности редактирования. Свойства и методы компонента «надпись» инкапсулированы в классе **TLabel** . Основное свойство надписей – *Caption*, в котором задается выводимый текст. Изменять значение этого свойства можно как во время разработки программы в окне (Object Inspector), так и во время выполнения программы. Следует иметь в виду, что свойство *Caption* имеет строковый тип и ему может быть присвоено только строковое значение. Для вывода числовых значений с использованием надписей необходимо воспользоваться функциями преобразования чисел в строки:

- **IntToStr** – преобразует целое число, заданное параметром, в строку;
- **FloatToStr** – преобразует действительное число в соответствующее ему строковое представление.


Свойство *WordWrap: Boolean* определяет будет ли выполняться автоматический перенос (если его значение true) или нет (если его значение false) [1, 6, 14, 21].

## 1.3. Флажки TCheckBox

Флажки используются для выбора одного из двух вариантов. Элемент «флажок» может находиться в одном из двух состояний: установлен (включен) или снят (выключен). Установленный флажок помечается крестиком. Компоненту «флажок» соответствует класс **TCheckBox** . Флажки имеют следующее основное свойство:


- *Checked: Boolean* – показывает, установлен флажок или нет. Если данное свойство имеет значение true, то флажок установлен, если false, то флажок снят;
- *Caption: TCaption* – поясняющая надпись [1, 6, 14, 21].

## 1.4. Переключатели TRadioButton

Переключатели предназначены для выбора одного из нескольких альтернативных вариантов. Свойства и методы данного компонента инкапсулированы в классе **TRadioButton** . Основным свойством переключателя является свойство *Checked* типа Boolean, которое показывает, выбран данный переключатель или нет. Все переключатели, помещенные в один контейнер (форма, фрейм и т.п.), считаются входящими в одну группу, из которой может быть выбран только один переключатель. Свойство *Caption* класса

TRadioButton используется для задания поясняющей надписи [1, 6, 14, 21].

### 1.5. Текстовое поле TEdit


Текстовое поле – стандартное поле ввода, которое позволяет отображать и редактировать текст. Возможность ввода текста с клавиатуры реализована в классе TEdit .

Свойства:

- *AutoSelect: Boolean* – определяет, будет ли выделяться текст в поле ввода при получении полем фокуса ввода;
- *Text: String* – содержит текст, отображаемый в поле ввода;
- *ReadOnly: Boolean* – если данное свойство true, то при выполнении программы отсутствует возможность редактирования текста, отображаемого в поле ввода [1, 6, 14, 21].


## 2. Стандартные компоненты Delphi для ввода и редактирования данных

### 2.1. Компонент TMemo

Компонент TMemo  предназначен для отображения и редактирования нескольких строк текста. Рассмотрим основные свойства этого элемента:

- *Alignment: TAlignment* – задает способ выравнивания текста в поле ввода TMemo (taLeftJustify – выравнивание по левому краю, taCenter – выравнивание по центру, taRightJustify – выравнивание по правому краю);
- *CaretPos: TPoint* – определяет координаты курсора в поле ввода TMemo;
- *Lines: TStrings* – массив строк, содержащихся в поле TMemo;
- *ScrollBars: TScrollStyle* – полосы прокрутки [1, 6, 14, 21].


### 2.2. Списки TListBox

Компонент TListBox  предназначен для работы со списком из нескольких вариантов. Пользователь может просмотреть весь этот перечень и выбрать одну или несколько строк для последующей обработки. Напрямую редактировать содержимое списка нельзя. Если все строки не помещаются в списке, в него автоматически добавляется вертикальная строка прокрутки.

Свойства и методы, обеспечивающую работу со списками инкапсулированы в классе TListBox. Основные свойства:


- *Columns: Integer* – позволяет создавать списки из нескольких столбцов. Если задано значение 0, то список состоит из одной колонки. В случае *Columns*>0 список будет состоять из нескольких колонок, причем значение *columns* определяет количество отображаемых столбцов;
- *ItemIndex: Integer* – порядковый номер выделенного элемента списка;
- *Items: TString* – массив строк списка. Используется для добавления, вставки, перемещения и удаления элементов списка;
- *MultiSelect: Boolean* – задает возможность выбора в списке нескольких элементов;
- *SelCount: Integer* – количество выбранных элементов списке;
- *Selected [Index: Integer]: Boolean* – определяет, выделен элемент списка с порядковым номером *Index* или нет;
- *Sorted: Boolean* – позволяет задать сортировку элементов списка по алфавитному порядку [1, 6, 14, 21].

### 2.3. Комбинированные поля TComboBox

объединяют возможности списка и текстового поля. Данный элемент управления позволяет пользователю выбрать компонент из списка заранее определенную строку или ввести строку, которой нет в списке. Список комбинированного поля может быть раскрывающимся. Для описания комбинированных полей используется класс **TComboBox** . Перечислим его основные свойства:

- *DropDownCount: Integer* – количество строк, отображаемых в выпадающей части списка;
- *DroppedDown: Boolean* – определяет, развернут выпадающий список или нет;
- *ItemIndex: Integer* – определяет выделенный элемент в выпадающем списке;
- *Items: TString* – массив строк выпадающего списка;
- *MaxLength: Integer* – максимальное количество символов, которое пользователь может ввести в текстовом поле;
- *SelText: String* – выделенный фрагмент строки в текстовом поле;
- *Sorted: Boolean* – определяет, сортировать элементы списка по алфавиту или нет [1, 6, 14, 21].

## 2.4. Таблица строк TStringGrid

Для ввода таблицы строк удобно использовать компонент **TStringGrid**. Значок компонента StringGrid  находится на вкладке Additional. Компонент представляет собой таблицу, ячейки которой содержат строки символов. Перечислим его основные свойства:

- *Name* – имя компонента;
- *ColCount* – количество колонок таблицы;
- *RowCount* – количество строк таблицы;
- *Cells* – соответствующий таблице двумерный массив. Ячейка таблицы, находящаяся на пересечении столбца номер col и строки номер row определяется элементом cells[col,row];
- *FixedCols* – количество зафиксированных слева колонок таблицы. Зафиксированные колонки выделяются цветом и при горизонтальной прокрутке таблицы остаются на месте;
- *FixedRows* – количество зафиксированных сверху строк таблицы. Зафиксированные строки выделяются цветом и при вертикальной прокрутке страницы остаются на месте;
- *Options.goEditing* – признак допустимости редактирования содержимого ячеек таблицы. True – редактирование разрешено, false – запрещено;
- *Options.goTab* – разрешает (true) или запрещает (false) использование клавиши <Tab> для перемещения курсора с следующей ячейку таблицы;
- *DefaultColWidth* – ширина колонок таблицы;
- *DefaultRowHeight* – высота строк таблицы;
- *GridLineWidth* – ширина линий, ограничивающих ячейки таблицы;
- *Left* – расстояние от левой границы поля таблицы до левой границы формы;
- *Top* – расстояние от верхней границы поля таблицы до верхней границы формы;
- *Height* – высота таблицы;
- *Width* – ширина таблицы;
- *Font* – шрифт используемый для отображения содержимого ячеек таблицы [1, 6, 14, 21].

## 2.5. Ввод даты и времени TDateTimePicker



Компонент **TDateTimePicker**  – поле ввода даты и времени.

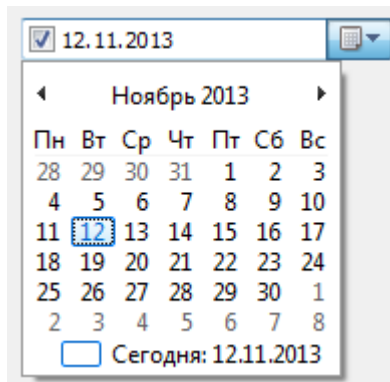


Рис. 1.1. Компонент *DateTimePicker*

Некоторые свойства компонента *DateTimePicker*:

- *DateFormat* - определение длинного (*dfLong*) или короткого (*dfShort*) формата даты;
- *DateMode* - способ работы компонента: счетчик для изменения даты (*dmUpDown*) или раскрывающийся список (*dmComboBox*);
- *Kind* - разновидность компонента: *dtkDate* - предназначен для ввода даты; *dtkTime* - предназначен для ввода времени.

При изменении даты или времени генерируется событие *OnChange* [1, 6, 14, 21].

### **TMonthCalendar**

Компонент **TMonthCalendar**  служит для быстрого выбора необходимой даты.

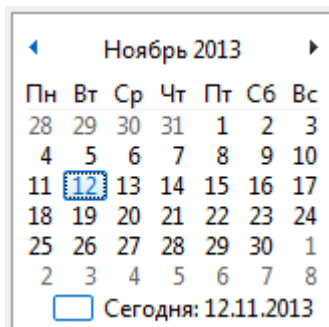


Рис. 1.2. Компонент *MonthCalendar*

Дата, выбранная щелчком мыши, выделяется синим цветом.

Некоторые свойства компонента *MonthCalendar*:

- *CalColors* - цвет элементов календаря;
- *MaxDate* - максимальная дата в календаре;
- *MinDate* - минимальная дата в календаре;
- *MultiSelect* - при значении *True* возможно выбирать диапазон дат;
- *ShowToday* - если *True*, то текущая дата отображается в нижней части календаря;

- *MaxSelectRange* - содержит максимальное количество дат в выбранном диапазоне [1, 6, 14, 21].

### 3. Диалоговые окна Delphi

#### 3.1. Диалоговые окна для вывода сообщений в Delphi (*ShowMessage*, *MessageDlg* и *MessageDlgPos*)

Рассмотрим методы Delphi для создания простых диалоговых окон вывода сообщений. Это процедура *ShowMessage*, функции *MessageDlgPos* и *MessageDlg*.

**ShowMessage(const Msg: String)** – эта процедура выводит окно с сообщением и кнопкой «Ok». В заголовке содержится название исполняемого файла, если в опциях приложения не задан параметр *Title*, если задан, то выводиться будет он. Строка *Msg* – будет выводиться как текст сообщения.

Например, команда

```
showmessage('В таблицу добавлена строка.');
```

сгенерирует окно, показанное на рис. 1.3.

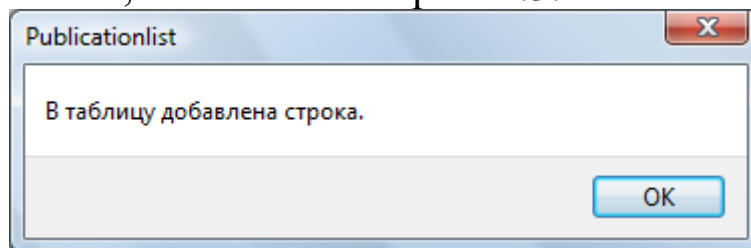


Рис. 1.3. Диалоговое окно команды *ShowMessage*

**MessageDlg(const Msg: String; AType: TMsgDlgType; Abuttons: TMsgButtons; HelpCtx: Longint): Word** – функция, показывающая диалоговое окно сообщения в центре экрана, и дающая возможность пользователю ответить на сообщение. *Msg* – параметр, отвечающий за выводимый текст сообщения.

Тип выводимого окна сообщения зависит от параметра *AType*, список возможных значений которого следующий:

- *mtError* – на фоне красного круга расположен белый крест и заголовок окна – *Error*;
- *mtWarning* – на фоне желтого треугольника расположен черный восклицательный знак – *!* и заголовок окна – *Warning*;
- *mtConfirmation* на фоне белого круга расположен синий знак *?* и заголовок окна – *Confirmation*;

- *mtInformation* – на фоне белого круга расположена синяя буква "i" и заголовок окна – Information;
- *mtCustom* – диалоговое окно не содержит рисунка, в заголовке выводится имя исполняемого файла приложения или Title свойства Application приложения.

*AButtons* – параметр, который задает набор кнопок на диалоговой форме и может принимать произвольные комбинации из значений:

- кнопка «Yes» – *mbYes*,
- кнопка «Ok» – *mbOk*,
- кнопка «No» – *mbNo*,
- кнопка «Cancel» – *mbCancel*,
- кнопка «Abort» – *mbAbort*,
- кнопка «Retry» – *mbRetry*,
- кнопка «Ignore» – *mbIgnore*,
- кнопка «All» – *mbAll*,
- кнопка «Help» – *mbHelp*.

Список из необходимых, перечисленных кнопок должен быть заключен в квадратные скобки.

```
MessageDlg('Очистить форму?', mtConfirmation, [mbYes,mbNo],0 );
```

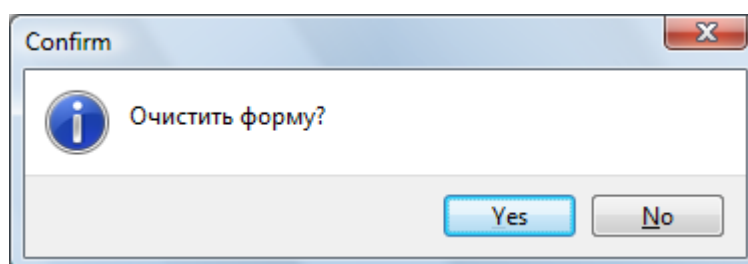


Рис. 1.4. Диалоговое окно команды *MessageDlg*

При щелчке на любой кнопке (кроме Help) результат возвращается функцией *MessageDlg* (свойство *ModalResult*), а сама форма закрывается. Результат может быть одним из значений списка:

```
mrOk      mrRetry  mrNo
mrNone    mrAbort  mrYes
mrCancel  mrIgnore mrAll
```

Для примера рассмотрим обработчик кнопки «Очистить форму»:

```

procedure TForm6.Button3Click(Sender: TObject);
var i,j:integer;
    result : TModalResult;
begin
result:=MessageDlg('Очистить форму?', mtConfirmation, [mbYes,mbNo],0);
if result = mrYes then
begin
Edit1.Text:='';
Edit2.Text:='';
Edit3.Text:='';
Edit4.Text:='';
Edit5.Text:='';
for i:= 1 to StringGrid1.RowCount do
for j:= 0 to StringGrid1.ColCount do
StringGrid1.Cells[j,i]:='';
end;
end;
end;


```

Послу запуска приложения и щелчка по этой кнопке появиться диалоговое окно, приведенное на рис. 1.4. Нажав на кнопку «Yes» мы очистим форму, «No» - никакие действия не будут выполнены.

**MessageDlgPos(const Msg: String; Atype: AMsgDlgType; Abuttons: TMsgDlgButtons; HelpCtx: Longint; X, Y: Integer):Word** - эта функция отличается от MessageDlg тем, что у нее есть два параметра X и Y, устанавливающие положение окна на экране [14].

### 3.2. Диалоговые окна выбора имени файла **OpenDialog** и **SaveDialog**

Диалоговые окна выбора имени файла используются в процессах открытия и сохранения файла. Часто эти диалоги называют также диалогами открытия/сохранения файла, хотя на самом деле они позволяют только выбрать имя файла, а все действия по открытию или сохранению файла программируются вручную.

Компонент **OpenDialog**  реализует диалог открытия файла. При запуске этого диалога появляется окно (см. рис. 1.5), в котором можно выбрать имя открываемого файла. В случае успешного

закрытия диалогового окна (нажатием кнопки «Открыть») в качестве результата возвращается выбранное имя файла.

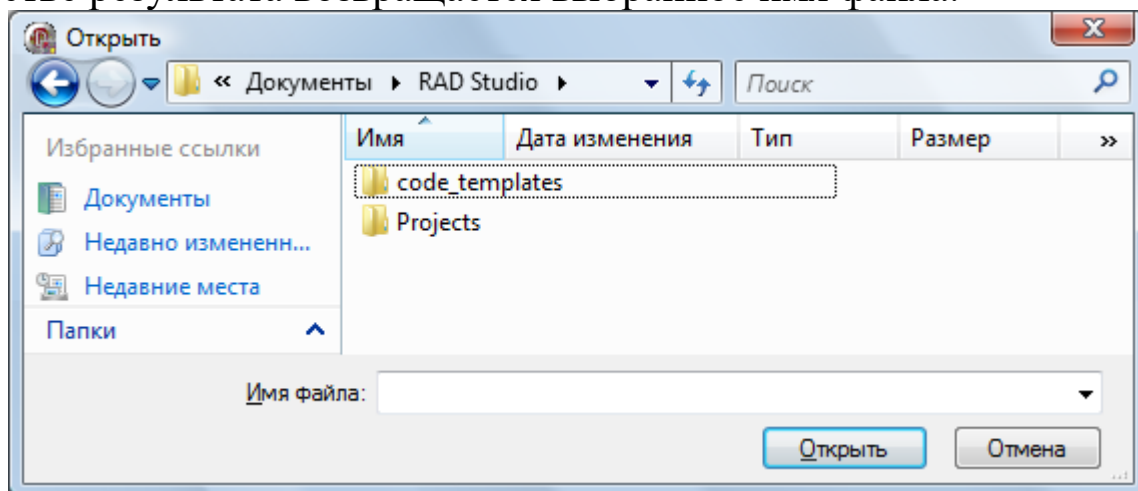



Рис. 1.5. Диалоговое окно компонента *OpenDialog*

Компонент **SaveDialog**  предлагает стандартный диалог сохранения файла, который отличается от диалога открытия файла только своим заголовком.

Основные свойства компонентов *OpenDialog* и *SaveDialog*:

- *FileName: String* – указывает имя и полный путь файла, выбранного в диалоге. Имя файла отображается в строке редактирования списка «Имя файла» и является результатом диалога.
- *Title: String* – задает заголовок окна. Если свойство *Title* не установлено, то по умолчанию используется заголовок «Открыть» для *OpenDialog* и заголовок «Сохранить» – для *SaveDialog*.
- *InitialDir: String* – определяет каталог, содержимое которого отображается при вызове диалогового окна. Если каталог не задан, то отображается содержимое текущего каталога.
- *DefaultExt: String* – задает расширение, автоматически используемое в имени файла, если пользователь не указал расширение.
- *Filter: String* – задает маски имен файлов, отображаемых в раскрывающемся списке «Тип файлов». В диалоговом окне видны имена файлов, совпадающие с указанной маской.
- *FilterIndex: Integer* – указывает, какая из масок фильтра отображается в списке. По умолчанию свойство *FilterIndex* имеет значение 1 (используется первая маска).
- *Options: TOpenOptions* – применяется для настройки параметров, управляющих внешним видом и функциональными

возможностями диалога. Каждый параметр (флажок) может быть установлен или снят. Свойство Options имеет около двух десятков параметров, наиболее важные из них перечислены ниже:

- *ofAllowMultiSelect* (из списка можно выбрать одновременно несколько файлов);
- *ofCreatePrompt* (при вводе несуществующего имени файла выдается запрос на создание файла);
- *ofNoLongNames* (имена файлов отображаются как короткие, не более 8 символов для имени и 3 символов для расширения);
- *ofOldStyleDialog* (создает диалоговое окно в стиле Windows 3.11) [1, 6, 14, 21].

### Пример: Список публикаций

Создать электронную форму списка публикаций (см. рис. 1.6).

СПИСОК  
научных и методических трудов  
старшего преподавателя кафедры информатики и вычислительной математики физико-математического факультета  
Узденовой Аминат Магоматовны

№ п/п	Наименование работы	Форма работы	Выходные данные	Объём п.д	Соавторы
1	Центральная проблема начального образования или тайны учебного труда	Печатный	Карачаево-Черкесский государственный педагогический университет. Вестник, 9/2002, С. 250-257.	0.5 0.25	Узденов М.И.

Список верен:

Сонскатель	А.М. Узденова	Зав. кафедрой	Д.Д. Маршанкулов
		Секретарь Учёного совета	М.Д. Тамбиева
		Проректор по научной работе	С.У. Пазов

(Дата, гербовая печать)

*Рис. 1.6. Образец заполнения документа*

Для этого надо выполнить последовательность следующих действий:

1. Создайте папку и дайте ей имя.

2. Создайте приложение Delphi (для этого запустите Delphi, выполните команды File → New → VCL Forms Application).
3. Сохраните приложение в созданную папку (File → Save project as... в появившемся окне выберите свою папку и введите имя приложения).
4. Разместите на форме следующие компоненты (см. рис. 1.7):
  - пять компонентов TLabel для надписей «Должность», «ФИО преподавателя», «Зав. кафедрой», «Секретарь Ученого совета», «Проректор по научной работе»;
  - пять компонентов TEdit для ввода должности и ФИО преподавателя, Зав. кафедрой, ФИО секретаря Ученого совета, ФИО проректора по научной работе;
  - компонент TStringGrid для ввода данных о публикациях (таблица на рис. 1.6);
  - три компонента TButton для добавления строки в таблицу, так как заранее неизвестно их число; сохранения в файл; очистки формы.
  - компонент TSaveDialog – диалоговое окно выбора имени файла.

Отредактируйте свойства этих компонентов (см. рис. 1.7).

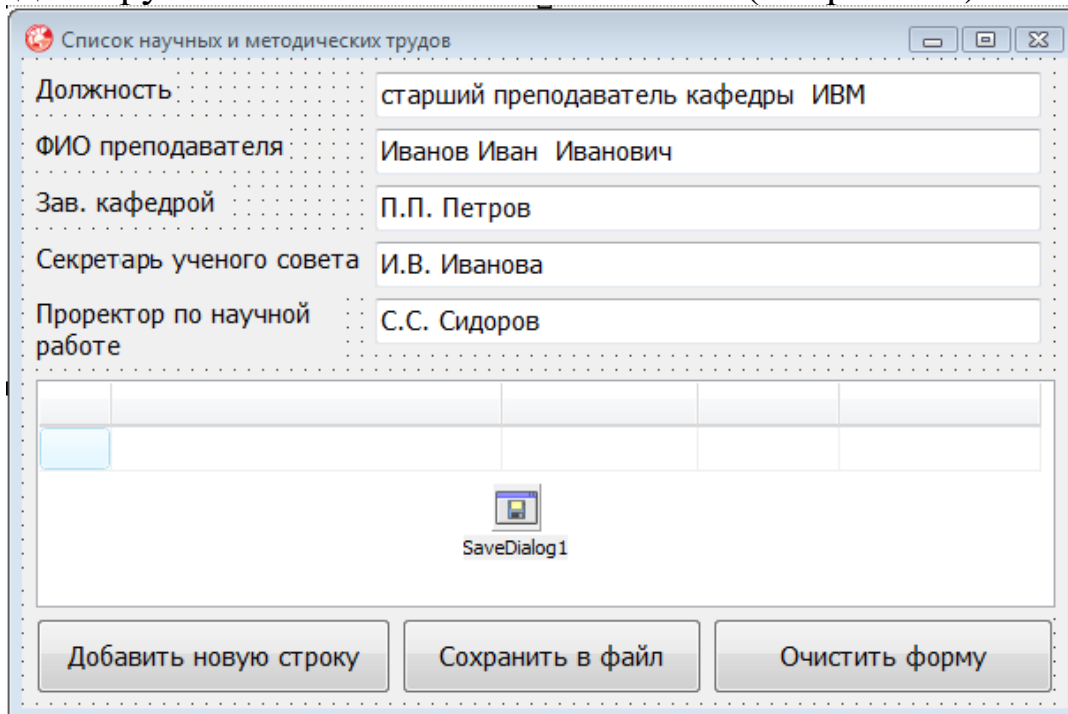


Рис. 1.7. Окно приложения

5. Далее запрограммируем кнопку «Добавить новую строку». Для этого нужно дважды щелкнуть компонент Button1 и в появившемся окне кода между *begin* и *end* ввести необходимые команды:

```

procedure TForm6.Button1Click(Sender: TObject);
begin
  stringgrid1.RowCount:=stringgrid1.RowCount+1;
  showmessage('В таблицу добавлена строка. ');
end;

```

Кнопка «Сохранить в файл»:

```

procedure TForm6.Button2Click(Sender: TObject);
  var f:textfile; i,j:integer; s:string;
begin
  if savedialog1.Execute then
  begin
    s:=savedialog1.FileName;
    assignfile(f,s);
    rewrite(f);
    writeln(f, 'Список научных и методических трудов');
    writeln(f,'должность: ', edit1.Text, ' Ф.И.О.:',Edit2.text);
    for i:= 0 to StringGrid1.RowCount do
    begin
      for j:= 0 to StringGrid1.ColCount do
        write(f, StringGrid1.Cells[j,i], ' ');
        writeln(f, '');
      end;
      writeln(f,'Соискатель ', Edit2.Text);
      writeln(f,'Зав. кафедрой ', Edit3.Text);
      writeln(f,'Секретарь ученого совета ', Edit4.Text);
      writeln(f,'Проректор по научной работе ', Edit5.Text);
      closeFile(f);
    end;
end;

```

Кнопка «Очистить форму»:

```

procedure TForm6.Button3Click(Sender: TObject);
  var i,j:integer;
  result : TModalResult;
begin
  result:=MessageDlg('Очистить форму?', mtConfirmation, [mbYes,mbNo],0);
  if result = mrYes then
  begin
    Edit1.Text:='';

```





Список научных и методических трудов

Должность: старший преподаватель кафедры ИВМ

ФИО преподавателя: Иванов Иван Иванович

Зав. кафедрой: П.П. Петров

Секретарь ученого совета: И.В. Иванова

Проректор по научной работе: С.С. Сидоров

№	Наименование работы	Форма работы	Выходные дан	Объем
1	Пограничная аномальная личность	Печатная	Ставрополь, 1	357 с.
2	Экспериментальная психология	Печатная	Москва, 2009	300 с.

Добавить новую строку      Сохранить в файл      Очистить форму

*Рис. 1.8. Тестирование*

### **Задания для самостоятельного выполнения**

**1.** Под каждой строкой, приведенных ниже процедур, описать её семантику.

**procedure TForm6.Button2Click(Sender: TObject);**

var f:textfile; i,j:integer; s:string;

**begin**

if savedialog1.Execute then

begin

s:=savedialog1.FileName;

assignfile(f,s);

rewrite(f);

writeln(f, 'Список научных и методических трудов');

writeln(f,'должность: ', edit1.Text, ' Ф.И.О.:',Edit2.text);

for i:= 0 to StringGrid1.RowCount do

begin

for j:= 0 to StringGrid1.ColCount do

write(f, StringGrid1.Cells[j,i], ' ');

writeln(f, "");

end;

writeln(f,'Соискатель ', Edit2.Text);

writeln(f,'Зав. кафедрой ', Edit3.Text);

writeln(f,'Секретарь ученого совета ', Edit4.Text);



<b>Бланк заказа электронной копии документа</b>	
Автор(ы) _____	
Тип издания _____	книга, периодическое издание
Название издания _____	
Название главы, статьи _____	
Место издания _____	
Год издания _____	Издательство _____
Том _____	Номер/Выпуск _____
Формат копирования _____	tiff, jpg
Форма получения заказа _____	по электронной почте, через FTP-сервер

*Рис. 1.9. Бланк заказа электронной копии документа в библиотеке*

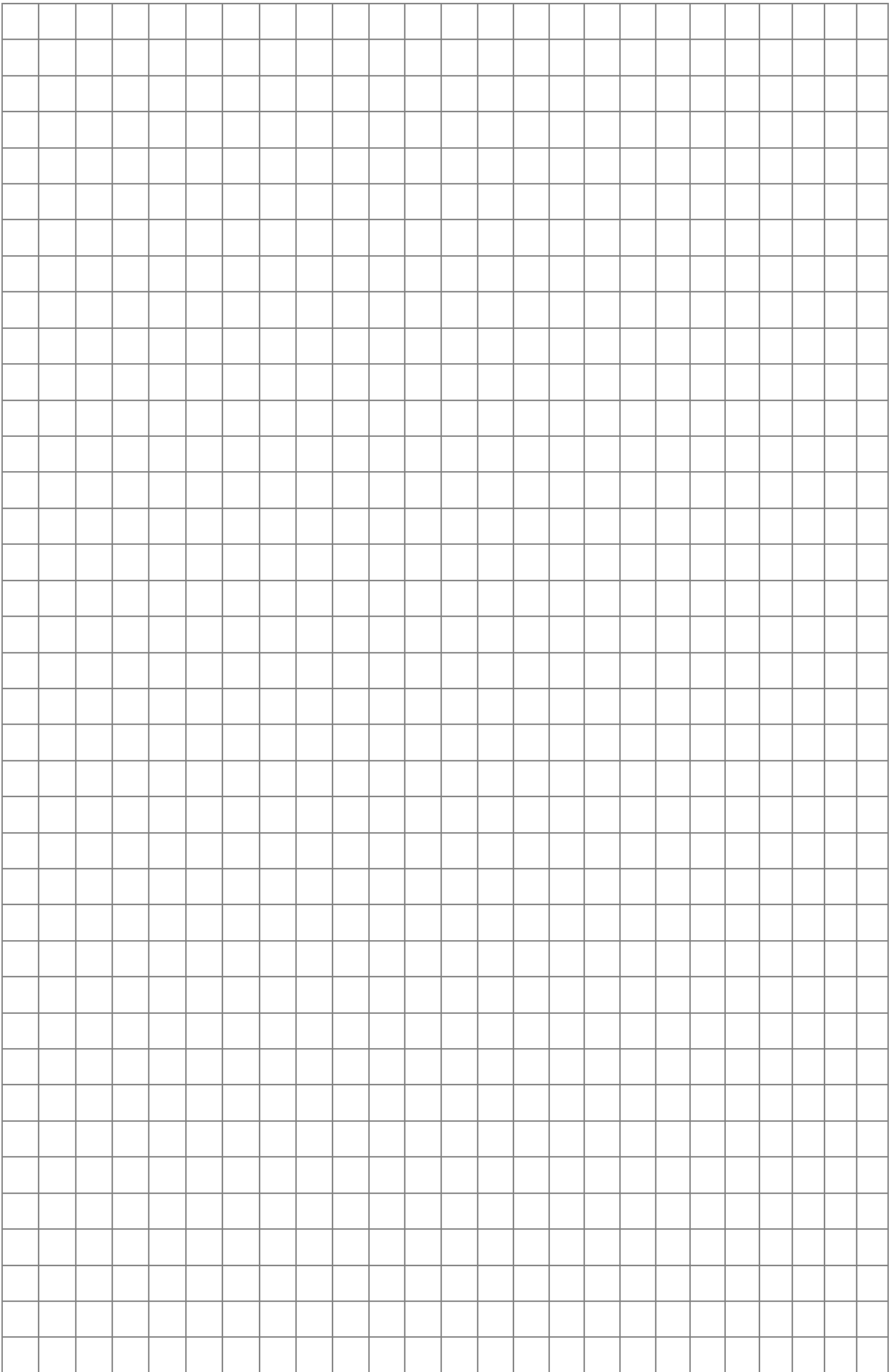
- б)** Описать компоненты, которые Вы использовали для ввода и вывода данных.
- в)** Привести программный код кнопок.

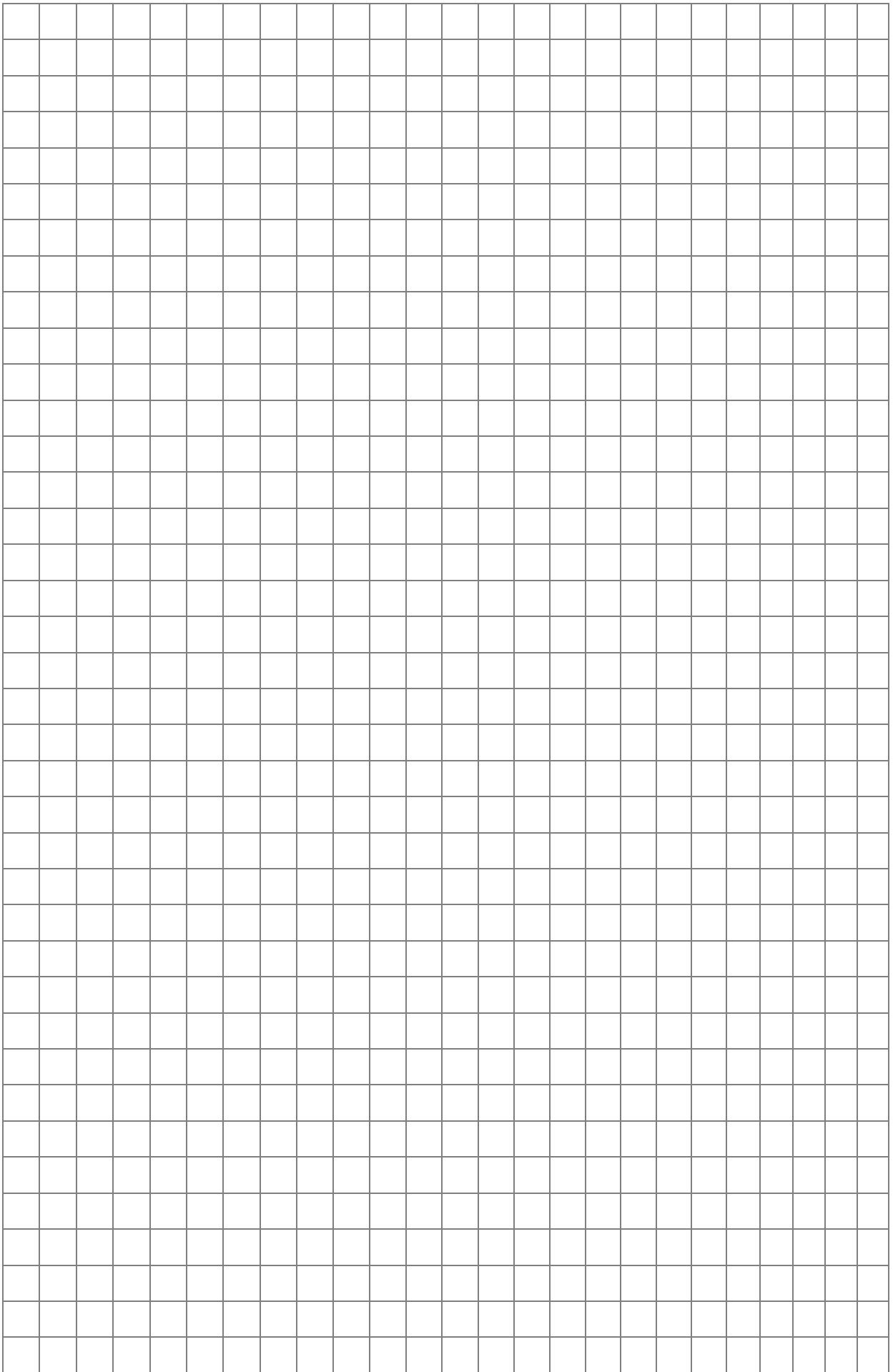
**3. Ответить на вопросы:**

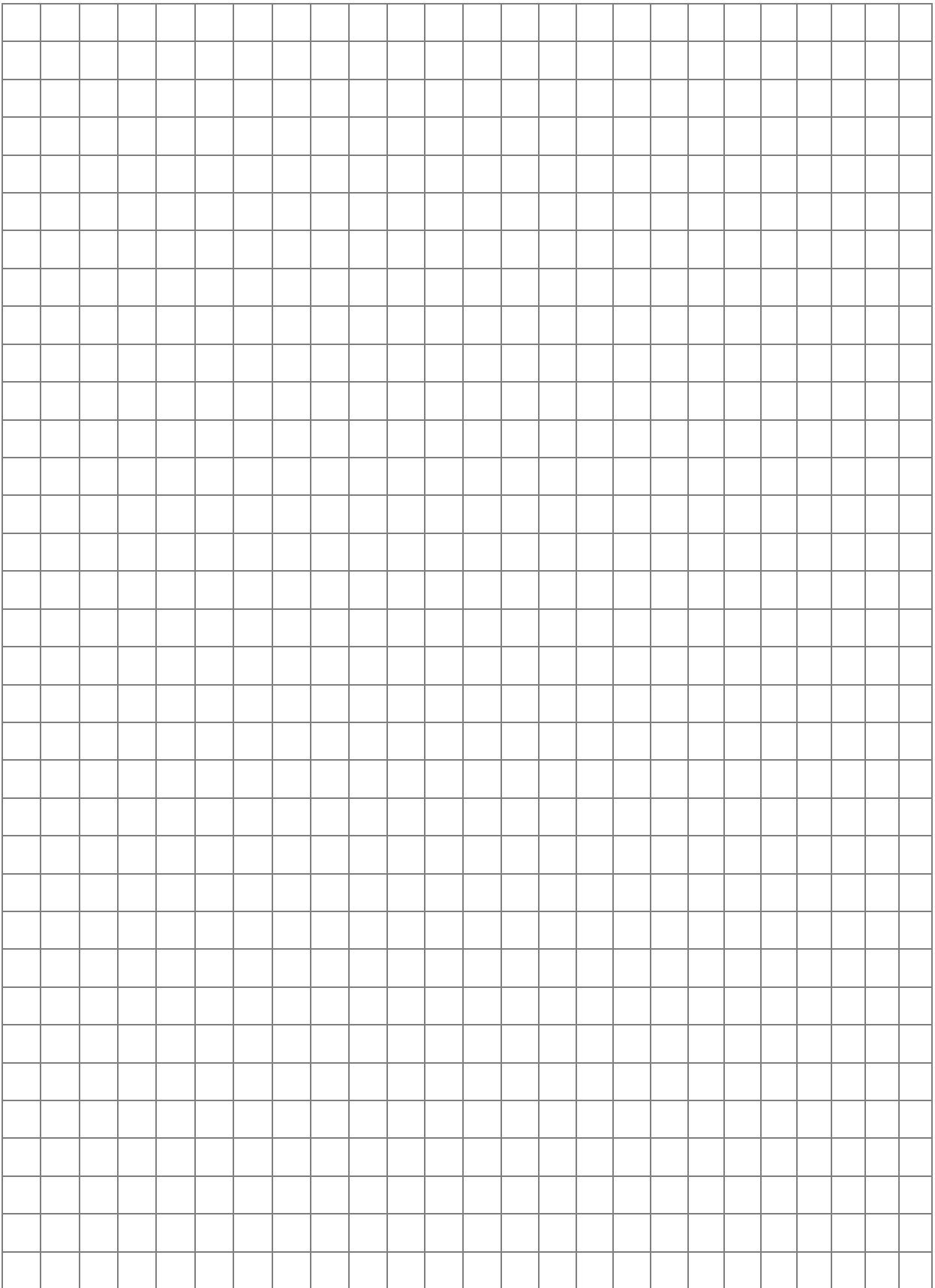
- а)** Какие компоненты можно использовать для вывода данных?
- б)** Какие компоненты можно использовать для ввода данных?
- с)** Перечислите компоненты, с помощью которых можно ввести однозначные величины?
- д)** Перечислите компоненты, с помощью которых можно ввести многозначные величины?

**4. Изучить самостоятельно свойства и методы компонента TBitBtn.**

**Письменный отчет**







## Тема 2. Создание форм для ввода и редактирования данных

**Цель:** сформировать навыки разработки форм ввода и редактирования данных. По завершению выполнения заданий данной темы студент должен уметь создавать формы ввода и редактирования данных.

### Теоретические сведения

#### 1. Формы в Delphi

Форма представляет окно приложения на этапе разработки и обеспечивает создание интерфейса пользователя, являясь контейнером для размещения элементов интерфейса. Все свойства класса **TForm** можно разделить на две группы – опубликованные свойства, то есть те свойства, которые отображаются в окне инспектора объектов во время разработки приложения, и свойства, которые можно изменять только в процессе выполнения программы [14].

Ниже приведены основные опубликованные свойства формы.

- *ActiveControl* - указывает на элемент управления, имеющий фокус ввода.
- *AutoScroll* - определяет необходимость создания автоматических полос прокрутки, если размер формы не позволяет отобразить все расположенные на ней элементы управления.
- *AutoSize* - если значение данного свойства равно true, то размеры формы автоматически изменяются таким образом, чтобы вместить все расположенные на ней элементы управления.
- *BorderIcons* - определяет набор кнопок в заголовке окна.
- *BorderStyle* - определяет тип границы окна и задает вид заголовка формы, отображение строки меню, наличие кнопок в заголовке окна.
- *BorderWidth* - ширина рамки вокруг окна.
- *Caption* - заголовок окна.
- *ClientHeight* - высота клиентской области в пикселах.
- *ClientWidth* - ширина клиентской области в пикселах.
- *Color* - цвет фона окна.
- *Constraints* - ограничения на изменения размера окна.
- *Ctl3D* - определяет внешний вид окна — «трехмерное» или «плоское».

- *Cursor* - вид указателя мыши над формой.
- *FormStyle* - стиль окна: *fsNormal* - обычное окно SDI; *fsStayOnTop* - окно SDI, всегда остающееся поверх всех других окон; *fsMDIForm* - главная форма MDI; *fsMDIChild* - дочерняя форма MDI.
- *Height* - высота окна в пикселах.
- *Icon* - значок формы, отображающийся в верхнем левом углу окна.
- *Left* - левая граница формы в координатах экрана.
- *Menu* - указатель на главное меню окна.
- *Name* - идентификатор экземпляра TForm.
- *PopupMenu* - указатель на контекстное меню окна.
- *Top* - верхняя граница формы в координатах экрана.
- *Visible* - определяет видимость формы.
- *Width* - ширина.
- *WindowState* - состояние окна: *wsMinimized* - свернуто; *wsMaximized* - развернуто на весь экран; *wsNormal* - обычное состояние.

Свойства, приведенные ниже, доступны для изменений как в инспекторе объектов во время разработки приложения, так и в процессе выполнения программы.

- *Active* - определяет, активна форма или нет (только для чтения).
- *Canvas* - используется для «рисования» на форме.
- *MDIChildCount* - количество дочерних окон (только для чтения).
- *MDIChildren [i : integer]* - указывает на i-е дочернее окно (только для чтения).

В классе TForm определен ряд методов-обработчиков событий, которые позволяют задавать реакцию экземпляра класса TForm на определенные действия. Здесь мы рассмотрим только основные из них:


- *OnActivate* – вызывается при передаче форме фокуса ввода;
- *OnClick* – вызывается при одиночном щелчке на форме;
- *OnDblClick* – вызывается при двойном щелчке на форме;
- *OnClose* – вызывается при закрытии формы;
- *OnCloseQuery* – вызывается перед закрытием формы.

Используется для задания параметра, возвращаемого методом *CloseQuery*;

- *OnCreate* – вызывается при создании формы;
- *OnDeactivate* – вызывается при потере формой фокуса ввода;
- *OnDestroy* – вызывается перед уничтожением формы;

- *OnPaint* – вызывается при перерисовке формы;
- *OnShow* – вызывается при отображении формы;
- *OnKeyPress* – вызывается при нажатии на клавишу;
- *OnMouseDown* – вызывается при нажатии левой кнопки мыши в области формы;
- *OnMouseUp* – вызывается при отпускании левой кнопки мыши в области формы;
- *OnMouseMove* – вызывается при движении указателя мыши над формой [1, 6, 14, 21].

## 2. Фреймы

Фреймы представляют собой контейнеры, предназначенные для размещения на них элементов управления. В этом плане фреймы подобны формам, однако в отличие от них фреймы могут помещаться на формы и другие фреймы. Перед размещением фрейма на форме его необходимо предварительно создать с помощью команды File → Other → Delphi Files → VCL Frame. Процедура размещения элементов управления на фрейме производится точно так же, как и на форме. Для помещения готового фрейма на форму используется компонент **TFrames**  палитры компонентов (страница Standard). При этом отображается список всех фреймов, существующих в текущем проекте. Выбранный из списка фрейм помещается на форме как обычный компонент.

Так как фреймы очень похожи на формы, то они обладают примерно теми же свойствами. Однако в отличие от форм фреймы содержат меньшее число методов и реагируют на меньшее количество событий. Так, например, у фреймов нет методов *Show* и *ShowModal*, так как они не могут отображаться вне форм [1, 6, 14, 21].

## 3. Методы работы с элементами управления

### 3.1. Размещение и удаление элементов управления

Для создания новой формы используется команда File → New → VCL Form главного меню. После создания формы на ней можно размещать элементы управления. Для этого щелкните на соответствующем значке компонента в палитре, а затем – в том месте формы, где предполагается разместить компонент. Для удаления компонента с формы выделите его щелчком мыши и

нажмите клавишу «Delete». Можно также воспользоваться командой главного меню Edit → Delete [1, 6, 14, 21].

### **3.2. Выравнивание компонентов на форме**

Для выравнивания элементов управления относительно формы, друг друга или заданной сетки в Delphi используется редактор форм. Кроме того, имеется возможность установки одинаковых размеров для группы выделенных объектов. Доступ к командам выравнивания обеспечивается с помощью подпункта «Align» пункта «Position» контекстного меню.

### **3.3. Выделение группы элементов управления**

Команды выравнивания, как правило, применяются к группе выделенных компонентов. Для выделения нескольких элементов можно воспользоваться двумя способами:

- удерживая клавишу «Shift», последовательно щелкните на нужных компонентах;
- удерживая левую кнопку мыши нажатой, обведите область формы, на которой расположены выбираемые компоненты, контуром выделения.

### **3.4. Команды выравнивания компонентов**

Для выравнивания компонентов можно использовать окно диалога Alignment, открывающееся командой Align контекстного меню редактора форм.


### **3.5. Порядок обхода элементов**

Хотя основным инструментом при работе в среде Windows является мышь, иногда пользователю приходится использовать и клавиатуру. Для передачи фокуса ввода от одного элемента управления к другому с помощью клавиатуры используется клавиша «Tab». По умолчанию порядок передачи фокуса ввода при нажатии на клавишу «Tab» определяется порядком помещения элементов на форму на стадии разработки приложения. Однако заранее довольно трудно определить предпочтительное направление обхода, чтобы в соответствии с ним размещать элементы. Поэтому в Delphi предусмотрена возможность изменения порядка обхода после размещения элементов. Для этого используется свойство *TabOrder*, имеющееся у всех визуальных элементов управления. Значение, присвоенное этому свойству,

определяет порядок передачи фокуса ввода. Компонент, для которого `TabOrder = 0`, получит фокус ввода при открытии формы.

Свойство `TabStop` определяет, может элемент получить фокус ввода (`true`) или нет (`false`).

#### 4. Компонент `TPageControl`

Компонент `TPageControl`  представляет собой многостраничный блокнот (см. рис. 2.2), причем доступ к каждой странице, содержащей свой набор элементов управления, осуществляется через корешки, на которых можно написать название, определяющее содержание страницы. Данный элемент управления удобен тем, что позволяет эффективно использовать ограниченное пространство экрана, создавая эффект книги, которую можно раскрыть на любой странице.

Основные свойства этого компонента:

- `MultiLine` – свойство, определяющее, разрешен ли вывод надписей на ярлычках в несколько строк;
- `Pages` – массив компонентов типа `TTabSheet`, которые можно создавать, выбрав пункт «New Page» из контекстного меню компонента `PageControl`;
- `PageCount` – число страниц;
- `ActivePage` – имя выбранной страницы (символьная строка).



У страниц наиболее важными являются свойства `PageIndex` (`0,1,...`), указывающее порядок, в котором появляются страницы, и `TabVisible`, указывающее, видима ли данная страница [1, 6, 14, 21].

#### *Пример: Счет-фактура*

Создать форму для ввода и редактирования данных счет-фактуры (см. рис. 2.1). Счет-фактура представляет собой документ, служащий основанием для принятия предъявленных сумм налога на

добавленную стоимость к вычету. При реализации товаров (работ, услуг) налогоплательщик налога на добавленную стоимость обязан выставить покупателю счет-фактуру не позднее пяти дней считая со дня отгрузки товара (выполнения работ, оказания услуг).

Для создания электронной формы надо выполнить последовательность следующих действий:

1. Создайте папку и дайте ей имя.
2. Создайте приложение Delphi (для этого запустите Delphi, выполнить команды File → New → VCL Forms Application).
3. Сохраните приложение в созданную папку (File → Save project as..., в появившемся окне выберите свою папку и введите имя приложения).
4. Разместите на форме следующие компоненты (см. рис. 2.2):
  - компонент TPageControl  и командой «New Page» контекстного меню этого компонента добавьте две страницы «Данные агентов» и «Таблица»;
  - на первую страницу «Данные агентов» добавьте 4 компонента TGroupBox  для объединения данных о продавце, покупателе, грузоотправителе, грузополучателе и др. компоненты (см. рис. 2.2);
  - на вторую страницу компонент TStringGrid для ввода данных о товарах (таблица на рис. 2.3) и кнопку TButton;
  - на форму три компонента TButton для очистки формы, расчета суммы, заполнения документа.

Приложение №1  
к Правилам ведения журналов учета полученных и выставленных счетов-фактур,  
книг покупок и книг продаж при расчетах по налогу на добавленную стоимость,  
утвержденным постановлением Правительства Российской Федерации от 2 декабря 2000 г. N 814  
(в редакции постановлений Правительства Российской Федерации  
от 15 марта 2001 г. N 189, от 27 июля 2002 г. N 575, от 16 февраля 2004 г. N 84, от 11 мая 2006 г. N 283)

## СЧЕТ-ФАКТУРА № 000289 от 12 июня 2007 г.

Продавец: ЗАО "Милана"  
Адрес: 129366, Москва, Ракетный бульвар, 17  
ИНН/КПП продавца: 7717027908/671010011

Грузоотправитель и его адрес: ЗАО "Милана", Адрес: 129366, Москва, Ракетный бульвар, 17  
Грузополучатель и его адрес: ООО "Чемпион", Адрес: 129366, Москва, Ракетный бульвар, 1  
К платежно-расчетному документу \_\_\_\_\_ от \_\_\_\_\_

Покупатель: ООО "Чемпион"  
Адрес: 129366, Москва, Ракетный бульвар, 1  
ИНН/КПП покупателя: 7717000408/671010011

Валюта: руб.

Наименование товара (описание выполненных работ, оказанных услуг), имущественного права	Единица измерения	Количество	Цена (тариф) за единицу измерения	Стоимость товаров (работ, услуг), имущественных прав, всего без налога	В том числе акциз	Налоговая ставка	Сумма налога	Стоимость товаров (работ, услуг), имущественных прав, всего с учетом налога	Страна происхождения	Номер таможенной декларации
1	2	3	4	5	6	7	8	9	10	11
Монитор 17" Samsung 710N (SKN) TFT	шт.	5.000	4'631.56	23'157.80	---	18%	4'168.40	27'326.20	Китай	10210130/211206/0017348/1
Принтер HP LaserJet 1020 Q5911A A4, 600x600dpi, 14ppm, USB	шт.	2.000	3'461.97	6'923.93	---	18%	1'246.31	8'170.24		
Сканер Bear Paw 2448 TA Pro (A4, 1200x2400, 48bit, USB, Slide)	шт.	1.000	1'711.64	1'711.64	---	18%	308.10	2'019.74		
<b>Всего к оплате</b>							<b>5722.81</b>	<b>37'516.18</b>		

Руководитель организации \_\_\_\_\_

Иванов В.П.

(Ф.И.О)

Главный бухгалтер \_\_\_\_\_

Петрова Г.С.

(Ф.И.О)

Индивидуальный предприниматель \_\_\_\_\_

(подпись)  
(развизиты свидетельства о государственной регистрации индивидуального предпринимателя)

ПРИМЕЧАНИЕ. Первый экземпляр - покупателю, второй экземпляр - продавцу

Рис. 2.2. Вкладка «Ввод данных агентов»

Рис. 2.3. Вкладка «Таблица» для ввода данных о товаре

5. Далее запрограммируем кнопку «Добавить новую строку». Для этого нужно дважды щелкнуть компонент Button1 и в появившемся окне кода между begin и end ввести необходимые команды:



Листинг программы:

```
unit Unit6;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, Grids, StdCtrls, ComCtrls;
type
  TForm6 = class(TForm)
    PageControl1: TPageControl;
    Label1: TLabel;
    Edit1: TEdit;
    Label2: TLabel;
    Edit2: TEdit;
    TabSheet1: TTabSheet;
    GroupBox1: TGroupBox;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Edit3: TEdit;
    Edit4: TEdit;
    Edit5: TEdit;
    GroupBox2: TGroupBox;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Edit6: TEdit;
    Edit7: TEdit;
    Edit8: TEdit;
    GroupBox3: TGroupBox;
    Label9: TLabel;
    Label10: TLabel;
    Edit9: TEdit;
    Edit10: TEdit;
    GroupBox4: TGroupBox;
    Label11: TLabel;
    Label12: TLabel;
    Edit11: TEdit;
    Edit12: TEdit;
    Label13: TLabel;
```

```

Edit13: TEdit;
Label14: TLabel;
Edit14: TEdit;
TabSheet2: TTabSheet;
StringGrid1: TStringGrid;
Button1: TButton;
Button2: TButton;
Button3: TButton;
Label15: TLabel;
Label16: TLabel;
Button4: TButton;
procedure Button4Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure Button1Click(Sender: TObject);
procedure FormShow(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
var
Form6: TForm6;
implementation
{ $R *.dfm }

procedure TForm6.FormShow(Sender: TObject);
begin
// заполнение шапки таблицы при создании формы
stringGrid1.Cells[0,0]:='Наименование товара';
stringGrid1.Cells[1,0]:='Единица измерения';
stringGrid1.Cells[2,0]:='Количество';
stringGrid1.Cells[3,0]:='Цена';
stringGrid1.Cells[4,0]:='Стоимость';
stringGrid1.Cells[5,0]:='В том числе акциз';
stringGrid1.Cells[6,0]:='Налоговая ставка';
stringGrid1.Cells[7,0]:='Сумма налога';
stringGrid1.Cells[8,0]:='Стоимость с учетом налогов';
stringGrid1.Cells[9,0]:='Страна';
stringGrid1.Cells[10,0]:='Номер должностной декларации';

```

```

end;
procedure TForm6.Button1Click(Sender: TObject);
begin
  // добавление одной строки в таблицу
  StringGrid1.RowCount:=StringGrid1.RowCount+1;
end;

procedure TForm6.Button2Click(Sender: TObject);
var i,j:integer;
begin
  // очистка формы
  edit1.Text:="";
  edit2.Text:="";
  edit3.Text:="";
  edit4.Text:="";
  edit5.Text:="";
  edit6.Text:="";
  edit7.Text:="";
  edit8.Text:="";
  edit9.Text:="";
  edit10.Text:="";
  edit11.Text:="";
  edit12.Text:="";
  edit13.Text:="";
  edit14.Text:="";
  for i:=1 to StringGrid1.RowCount do
  for j:=0 to stringgrid1.ColCount do
  StringGrid1.Cells[j,i]:="";
  StringGrid1.RowCount:=2;
end;
procedure TForm6.Button4Click(Sender: TObject);
var i,j:integer;
    s:real;
begin
  // вычисление суммы к оплате
  for i:=1 to StringGrid1.RowCount do
    if StringGrid1.cells[8,i]<>" then
      s:=s+StrToFloat(StringGrid1.Cells[8,i]);
  label16.Caption:=FloatToStr(s);

```

end;  
end.

Счет-фактура

Счет фактура № 000289 от 12.09.2010

Данные агентов Таблица

Единица измерения	Количество	Цена	Стоимость	В том числе акц	Налоговая ставка	Сумма налога	Стоимость с уч	Страна
шт.	5	4631.56	23157.80		18%	4168.40	27326	Китай
шт.	10	150	1500		18%		177	

Всего к оплате **27503**

Добавить строку

Очистить форму Рассчитать сумму к оплате Заполнить документ

Рис. 2.4. Тестирование приложения



2. Создать форму для ввода и редактирования данных согласно Вашему варианту (см. ниже). При создании формы определите поля, которые вводятся пользователем, какого они типа, какие значения могут принимать. В согласии с этим выберите компоненты для ввода данных, их размер и расположение на форме. При необходимости разместить большое число компонентов, используйте компоненты TPageControl или добавьте еще формы в приложение.

Варианты:

- 1) Авансовый отчет.
- 2) Академическая справка.
- 3) Акт о неисполнении трудовых обязанностей.
- 4) Акт о приемке выполненных работ.
- 5) Акт приема-передачи автомобиля.
- 6) Акт сдачи-приемки товара.
- 7) Анкета поступающего на работу.
- 8) Балльно-рейтинговый лист студентов.
- 9) Банковская книжка.
- 10) Брачный договор.
- 11) Бухгалтерский баланс.
- 12) Генеральная доверенность.
- 13) График отпусков.
- 14) Доверенность на ведение дел в суде.
- 15) Доверенность на получение материальных ценностей.
- 16) Доверенность на получение пенсии.
- 17) Договор банковского вклада.
- 18) Договор дарения квартиры.
- 19) Договор купли-продажи квартиры.
- 20) Договор купли-продажи транспортного средства.
- 21) Договор мены квартир.
- 22) Договор на реализацию продукции покупателям.
- 23) Договор о займе денег.
- 24) Договор о полной индивидуальной материальной ответственности.
- 25) Журнал регистрации амбулаторных больных.

- 26) Журнал регистрации приходных и расходных кассовых документов.
- 27) Журнал учета пропущенных и замещенных уроков.
- 28) Журнал факультативных занятий.
- 29) Записка-расчет о предоставлении отпуска.
- 30) Записка-расчет при прекращении трудового договора с работником.
- 31) Заявление о выдаче водительского удостоверения.
- 32) Заявление о выдаче паспорта.
- 33) Заявление о государственной регистрации транспортного средства.
- 34) Заявление о расторжении договора банковского счета.
- 35) Инвентаризационная опись основных средств.
- 36) Исковое заявление о взыскании алиментов на ребенка.
- 37) Исковое заявление о взыскании заработной платы.
- 38) Исковое заявление о возврате вклада и защите прав потребителя.
- 39) Исковое заявление о возмещении ущерба, причиненного заливом квартиры.
- 40) Исковое заявление о расторжении брака.
- 41) Исковое заявление о снятии дисциплинарного взыскания.
- 42) Исполнительская надпись.
- 43) Карта учета диспансеризации.
- 44) Квитанция о принятии денежных средств в депозит.
- 45) Книга покупок.
- 46) Книга продаж.
- 47) Командировочное удостоверение.
- 48) Лист нетрудоспособности.
- 49) Личная карточка сотрудника.
- 50) Медицинская карта.
- 51) Медицинский рецепт.
- 52) Накладная на получение материальных ценностей.
- 53) Налоговая декларация по транспортному налогу.
- 54) Направление на анализы пациента.

- 55) Наряд на выполнение работ.
- 56) Опись вложения.
- 57) Опись дел.
- 58) Отчет о движении денежных средств.
- 59) Отчет о кассовых поступлениях и выбытиях.
- 60) Отчет о целевом использовании полученных средств.
- 61) Отчет об изменениях капитала.
- 62) Отчет об использовании акцизных марок.
- 63) Отчеты о прибылях и убытках.
- 64) Паспорт больного аллергическим заболеванием.
- 65) Платежное требование.
- 66) Приемный акт на материальные ценности.
- 67) Приказ о наложении дисциплинарного взыскания.
- 68) Приказ о переводе работника на другую работу.
- 69) Приказ о поощрении работника.
- 70) Приказ о приеме на работу.
- 71) Приказ об отпуске сотрудника.
- 72) Приказ об увольнении.
- 73) Протокол осмотра и исследования вещественных доказательств.
- 74) Распоряжение об отмене доверенности.
- 75) Расчет платы за негативное воздействие на окружающую среду.
- 76) Расчетный листок.
- 77) Регистрационные карточки внутренних приказов и распоряжений предприятия (организации) документов.
- 78) Регистрационные карточки исходящих документов предприятия (организации).
- 79) Реестр сведений о доходах физических лиц.
- 80) Свидетельство о рождении.
- 81) Сводная ведомость успеваемости группы студентов.
- 82) Служебное задание для направления в командировку.
- 83) Согласие на выезд детей.

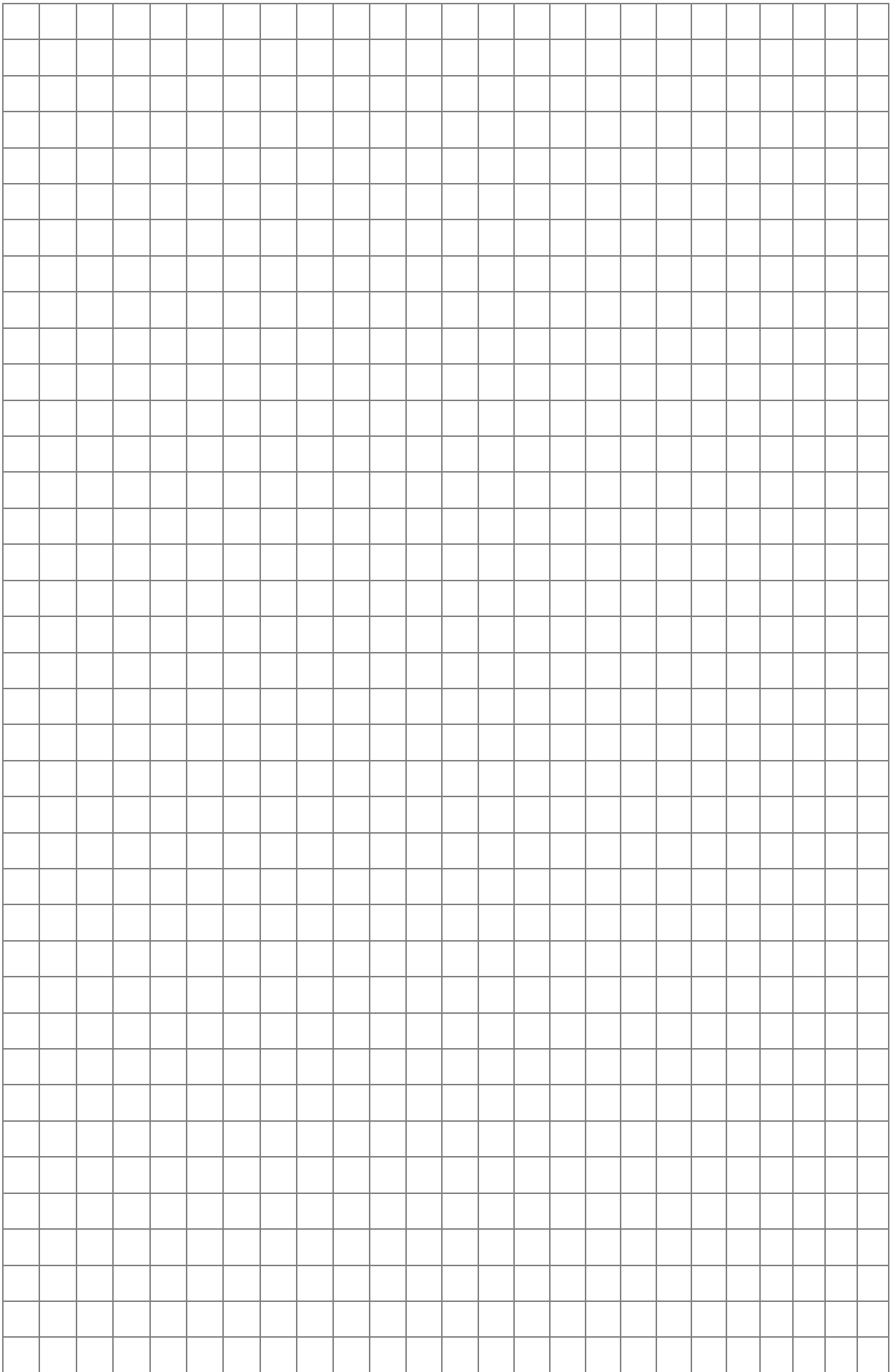


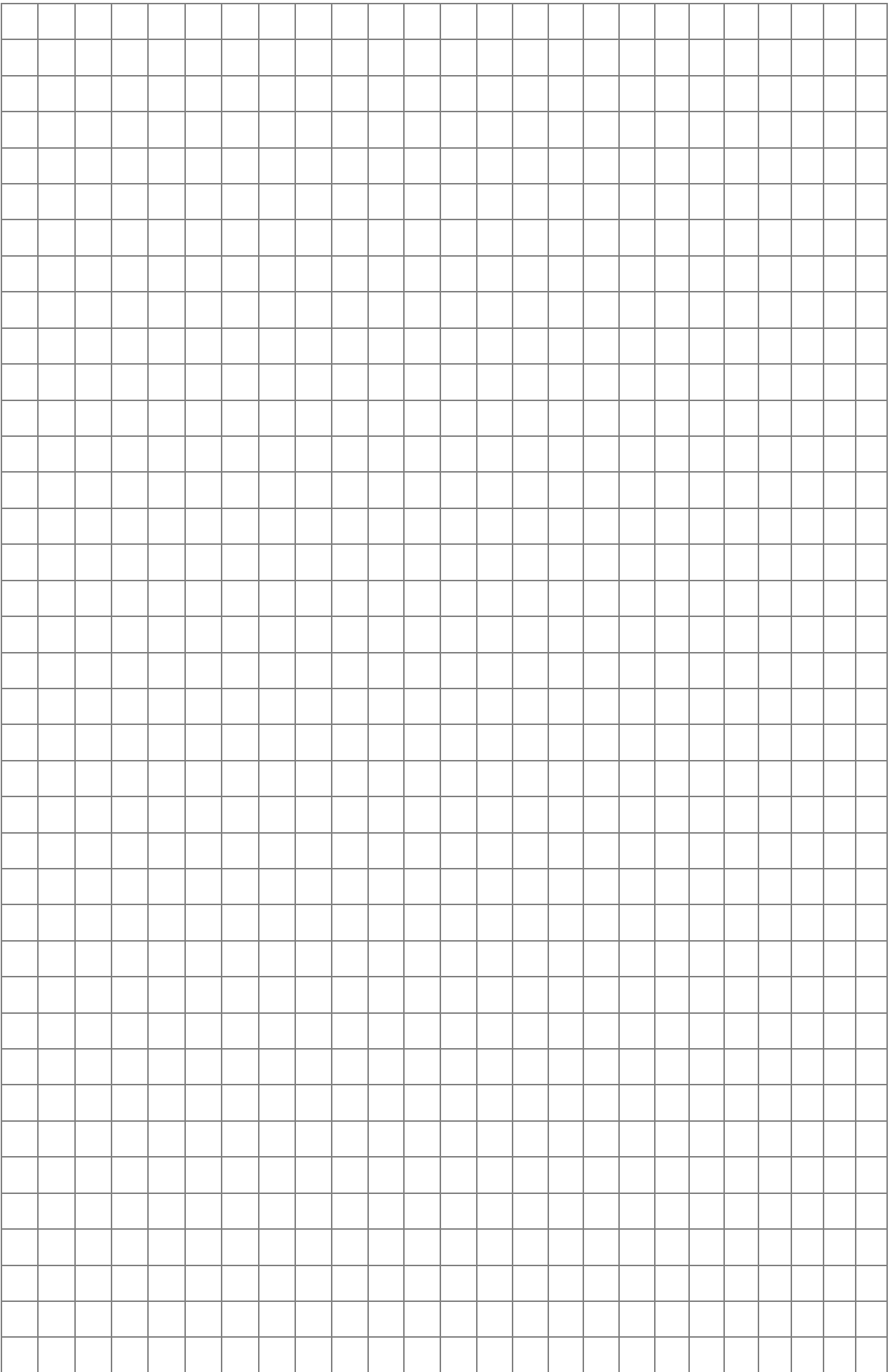
- 84) Соглашение о месте жительства ребенка при раздельном проживании родителей.
- 85) Соглашение об уплате алиментов.
- 86) Справка о заработной плате.
- 87) Справка о составе семьи.
- 88) Справка об обучении на факультете.
- 89) Справка с места работы.
- 90) Срочный трудовой договор.
- 91) Счета на оплату предоставляемых товаров (услуг).
- 92) Таможенная декларация.
- 93) Температурный лист.
- 94) Товарный ценник.
- 95) Товарный чек.
- 96) Трудовая книжка.
- 97) Туристическая путевка.
- 98) Уведомление о необходимости получения трудовой книжки.
- 99) Уведомление работника об истечении срока трудового договора.
- 100) Штатное расписание.

**3. Ответить на вопросы:**

- а)** Что такое форма, фрейм?
- б)** Какими свойствами и событиями обладает форма?
- в)** Какова разница между формами и фреймами?
- г)** Какое свойство элементов управления отвечает за порядок обхода?
- д)** Какие компоненты Вы использовали при выполнении своего варианта задания?







# Тема 3. Проектирования фактографических баз данных

**Цель:** Изучение этапов проектирования фактографических БД на примере создания локальной реляционной базы данных.

## Теоретические сведения

### 1. Концептуальное моделирование структуры данных. Модель «сущность-связь»

Одной из наиболее популярных концептуальных моделей данных является модель «сущность-связь» (часто называемая также ER-моделью - по первым буквам английских слов Entity (сущность) и Relation (связь)). Моделирование предметной области базируется на использовании графических диаграмм, включающих небольшое число разнородных компонентов. Основными понятиями ER-диаграммы являются *сущность*, *связь* и *атрибут*.

**Сущность** - это реальный или виртуальный объект, имеющий существенное значение для рассматриваемой предметной области, информация о котором подлежит хранению [7]. Можно считать, что сущности соответствуют таблицам реляционной модели. Каждая сущность должна обладать следующими свойствами:

- иметь уникальный идентификатор;
- содержать один или несколько атрибутов, которые либо принадлежат сущности, либо наследуются через связь с другими сущностями;
- содержать совокупность атрибутов, однозначно идентифицирующих каждый экземпляр сущности [7].

#### 1.1. Изображение простого объекта

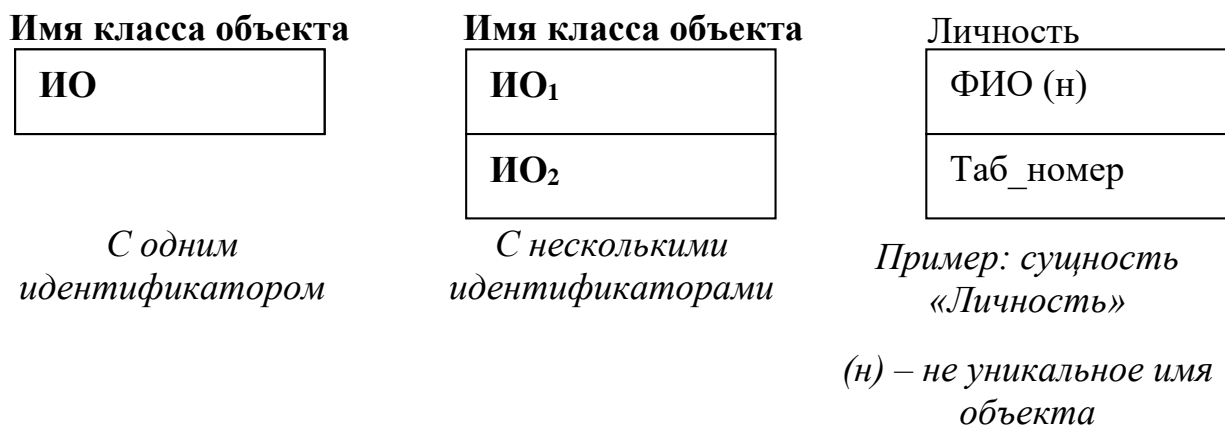


Рис. 3.1. Изображение простых объектов

## Описание свойств объекта



Рис. 3.2. Описание сущности

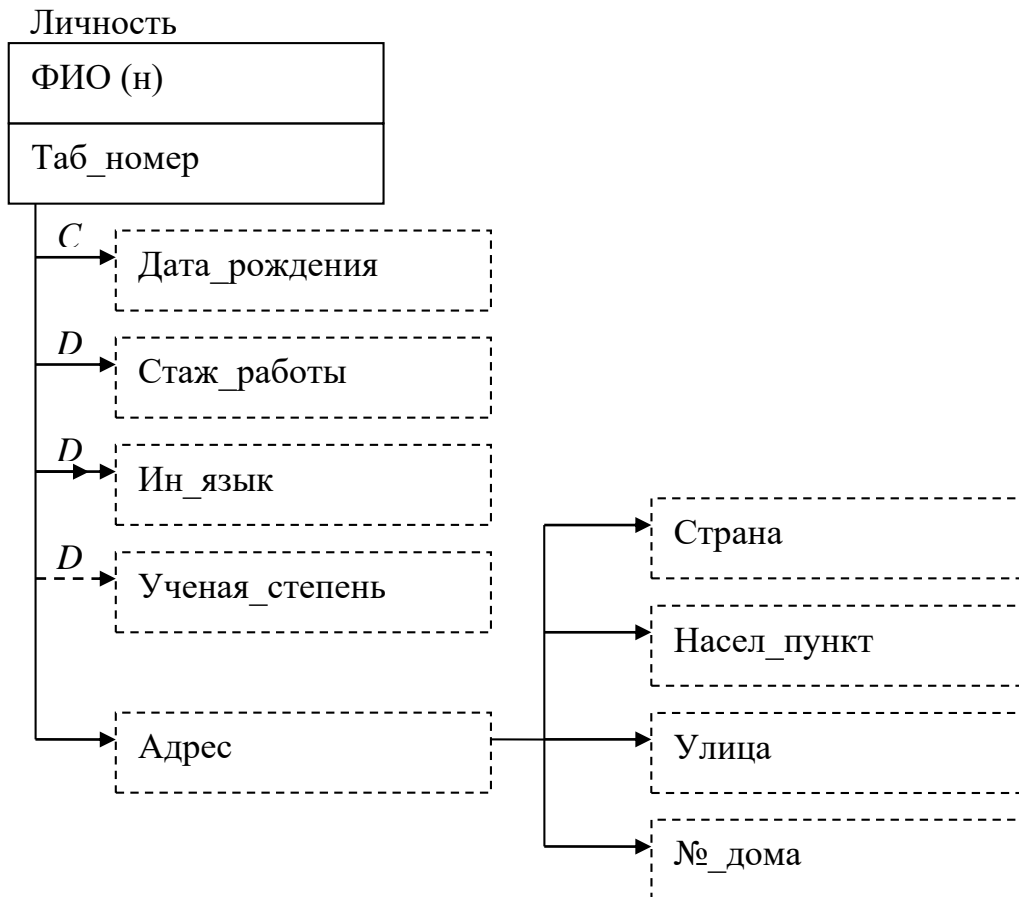


Рис. 3.3. Пример описания сущности «Личность»

Любая сущность может иметь произвольное количество связей с другими сущностями.



## 2. Дatalogическое проектирование БД

### 2.1. *Отображение простых объектов*

*Определение состава полей основной таблицы.* Для каждого простого объекта и его единичных свойств строится отношение, атрибутами которого являются идентификаторы объекта и реквизиты, соответствующие каждому из единичных свойств.

Любой из уникальных идентификаторов объекта является вероятным ключом полученного отношения.

Например, сущности **Личность** изображенной на Рис. 3.3 соответствует отношение

Личность (Таб\_номер, ФИО, Дата\_рождения, Стаж\_работы, Ученая\_степень, Адрес)

*Отображение множественных свойств.* Если у объекта имеются множественные свойства, то каждому из них ставится в соответствие отдельное отношение, полями которого будут идентификатор объекта и поле, отображающее множественное свойство. Ключ этого отношения будет составным, включающим оба эти атрибута. Так, для множественного свойства Ин\_язык будет создано отношение

Ин\_язык (Таб\_номер, ин\_язык)

*Отображение условных свойств объекта.* Если объект обладает условными свойствами, то при отображении их в реляционную модель возможны варианты.

1. Если многие объекты обладают рассматриваемым свойством, то его можно хранить в базе данных так же, как и обычное свойство, т.е. в той же таблице, в которой бы атрибут хранился, если бы свойство было определенным для всех экземпляров рассматриваемой сущности.

2. Если только незначительное число объектов обладает указанным свойством, то для многих записей в файле базы данных при использовании предыдущего решения значение соответствующего поля будут пустым. Поэтому выделяют отдельное отношение, которое будет включать идентификатор объекта и атрибут, соответствующий рассматриваемому свойству.

Ученая\_степень (Таб\_номер, ученая\_степень)



*Отображение составных свойств объекта.* Если объект имеет составное свойство, то возможны два варианта его отображения в БД [7]:

1. Всему составному свойству ставится в соответствие одно поле.

2. Каждому из составляющих элементов составного свойства ставится в соответствие отдельное поле:

Личность (Таб\_номер, ФИО, Дата\_рождения, Стаж\_работы, Ученая\_степень, Страна, Насел\_пункт, Улица)

## 2.2. Отображение связей между объектами

Связи между объектами также должны отражаться в структуре БД.

Универсальный способ отображения связей между объектами – введение вспомогательного связующего файла, содержащего идентификаторы связанных объектов. Ключ этого отношения будет составным.

### Отображение связи типа M:M

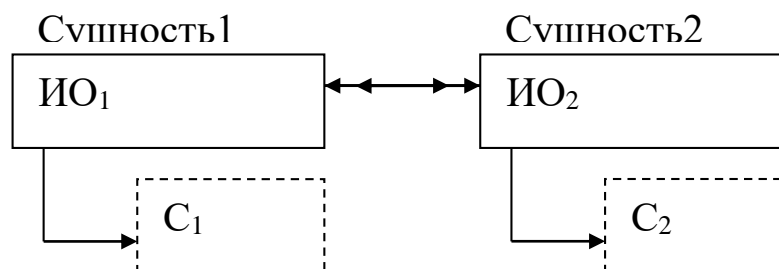


Рис. 3.7.

Для хранения такой информации потребуется три отношения: по одному для каждой сущности и одно дополнительное – для отображения связи между ними.

$R_1(\underline{IO_1}, C_1)$ ,

$R_2(\underline{IO_2}, C_2)$ ,

$R_3(\underline{IO_1}, \underline{IO_2})$ .

Например, модели (см. Рис. 3.8) соответствует схема данных:

Преподаватель (ФИО, Должность),

Предмет (ID\_предмета, Название),

Нагрузка (ФИО, ID\_предмета).

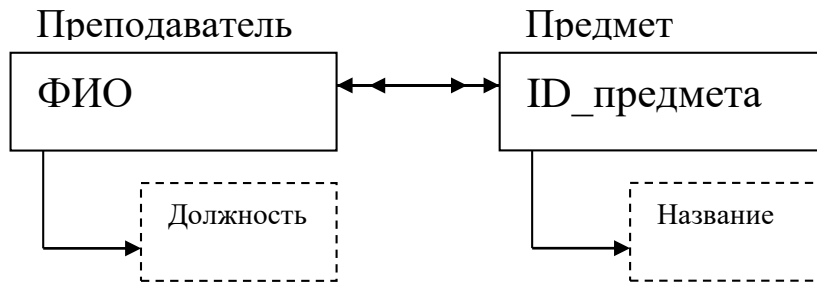


Рис. 3.8.

### Отображение связи типа 1:M

Если между объектами предметной области имеется связь 1:M, то можно, использовать отдельную связывающую таблицу.

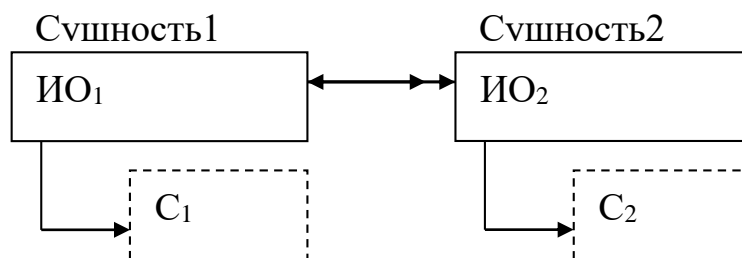


Рис. 3.9

#### Вариант 1

$R_1(\underline{ИО}_1, C_1),$   
 $R_2(\underline{ИО}_2, C_2),$   
 $R_3(\underline{ИО}_1, \underline{ИО}_2).$

#### Вариант 2

$R_1(\underline{ИО}_1, C_1),$   
 $R_2(\underline{ИО}_2, C_2, \underline{ИО}_1).$

В этом случае ключом связующей таблицы будет только идентификатор объекта, к которому направлен единичный конец связи.

Например,

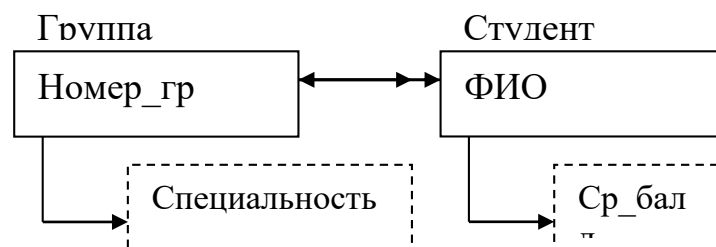


Рис. 3.10

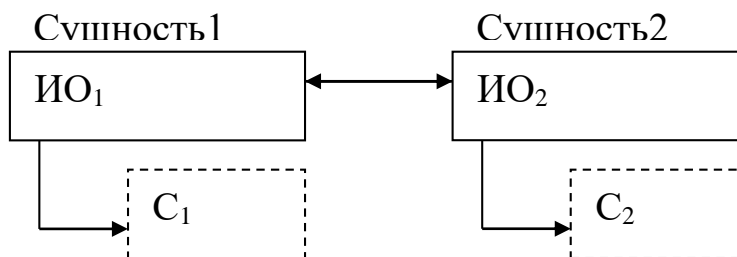
#### Вариант 1

Группа (Номер\_гр,  
 Специальность),  
 Студент (ФИО, Ср\_балл),  
 Ст-Гр(Номер\_гр, ФИО).

#### Вариант 2

Группа (Номер\_гр, Специальность),  
 Студент (ФИО, Ср\_балл, Номер\_гр).

### **Отображение связи типа 1:1**



*Рис. 3.11*

**Вариант 1**

$R_1(\underline{ИО_1}, \underline{ИО_2}, C_1, C_2)$

**Вариант 2**

$R_1(\underline{ИО_1}, C_1, \underline{ИО_2}),$   
 $R_2(\underline{ИО_2}, C_2).$

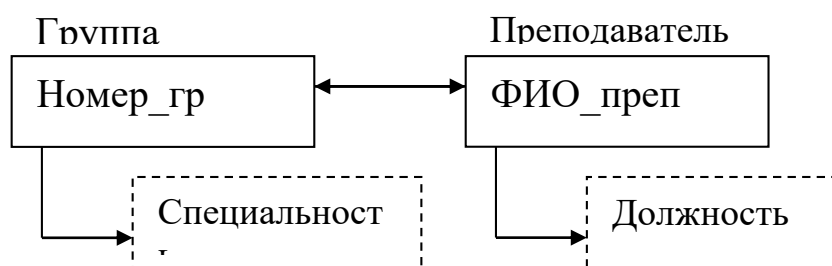
**Вариант 3**

$R_1(\underline{ИО_1}, C_1),$   
 $R_2(\underline{ИО_2}, C_2, \underline{ИО_1}).$

**Вариант 4**

$R_1(\underline{ИО_1}, C_1),$   
 $R_2(\underline{ИО_2}, C_2),$   
 $R_3(\underline{ИО_1}, \underline{ИО_2}).$

Например,



*Рис. 3.12*

**Вариант 1**

Кураторство (Номер\_гр, ФИО\_преп, Специальность, Должность)

**Вариант 2**

Группа (Номер\_гр, Специальность, ФИО\_преп),  
Преподаватель (ФИО\_преп, Должность).

**Вариант 3**

Группа (Номер\_гр, Специальность),  
Преподаватель (ФИО\_преп, Должность, Номер\_гр).

**Вариант 4**

Группа (Номер\_гр, Специальность),  
Преподаватель (ФИО\_преп, Должность),  
Кураторство (Номер\_гр, ФИО\_преп).

### Пример: Справочник коммерческих банков

Рассмотрим проектирования БД на примере «Справочника коммерческих банков».

Справочник коммерческих банков должен содержать наименование банка, адрес, статус (форма собственности), условия хранения средств на лицевом счете (годовые проценты на различных видах вкладов) и позволять выбирать банк с наибольшим процентом для заданного типа вклада.

**Инфологическая модель** данной предметной области показана на Рис. 3.13.

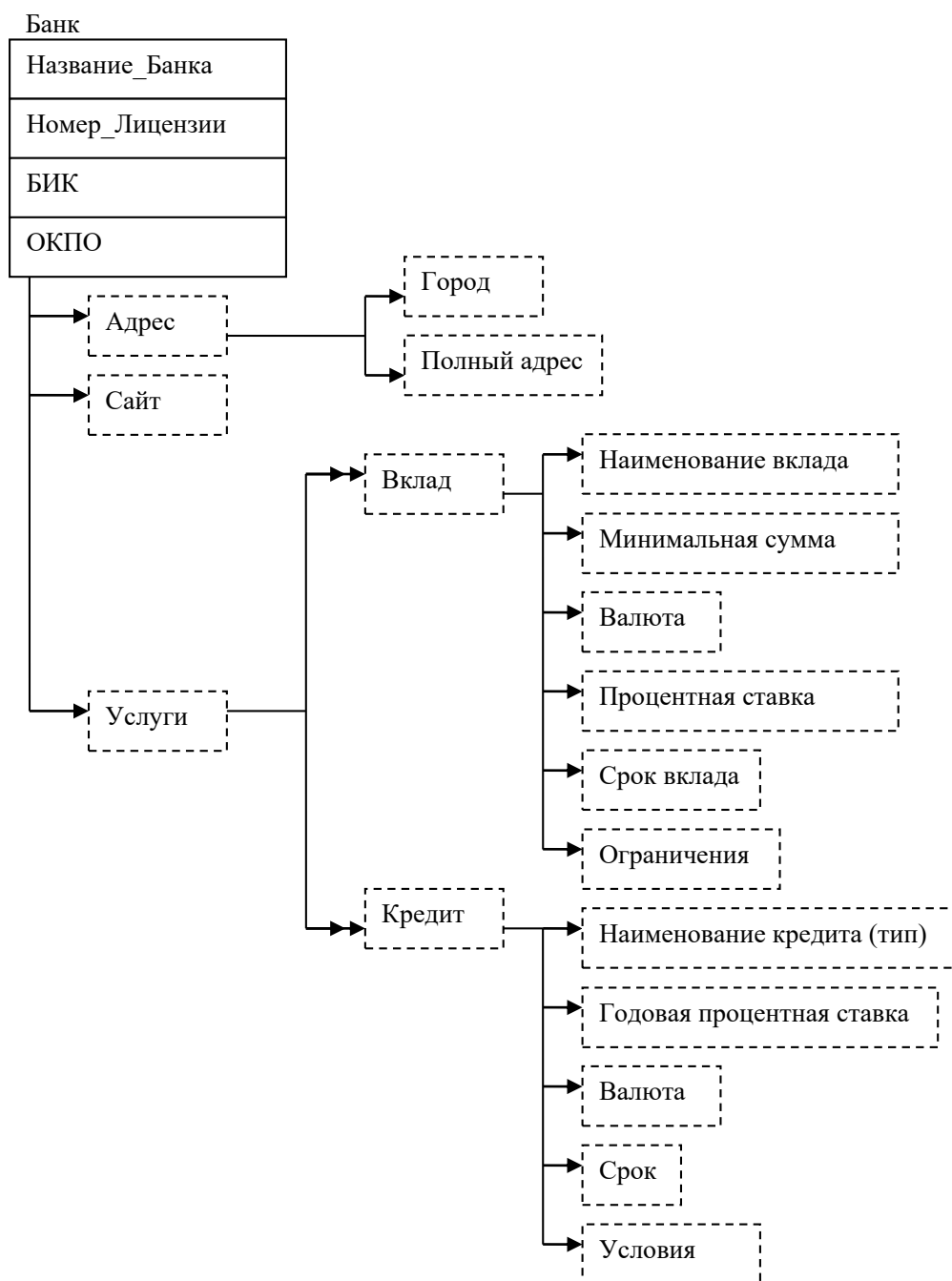


Рис. 3.13. Описание сущности «Банк»

*Даталогическая модель* содержит 3 отношения:

**Банк** (Название\_Банка, Номер\_Лицензии, БИК, ОКПО, Сайт, Город, полный адрес)

**Кредит** (Наименование кредита, Название Банка, Годовая процентная ставка, Валюта, Срок, Условия)

**Вклад** (Наименование вклада, Название Банка, Минимальная сумма, Валюта, Процентная ставка, Срок вклада, Ограничения)

*Схема данных*, построенная на основе этой даталогической модели приведена на Рис. 3.14.

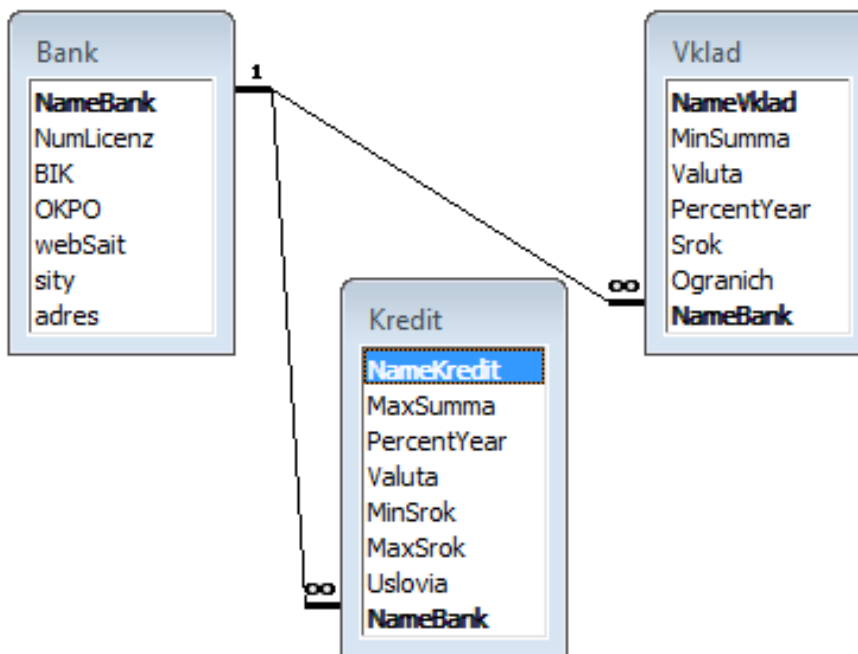


Рис. 3.14. Схема данных

Таким образом, база данных «Справочника коммерческих банков» должна содержать три таблицы: «Банк», «Кредит», «Вклад».

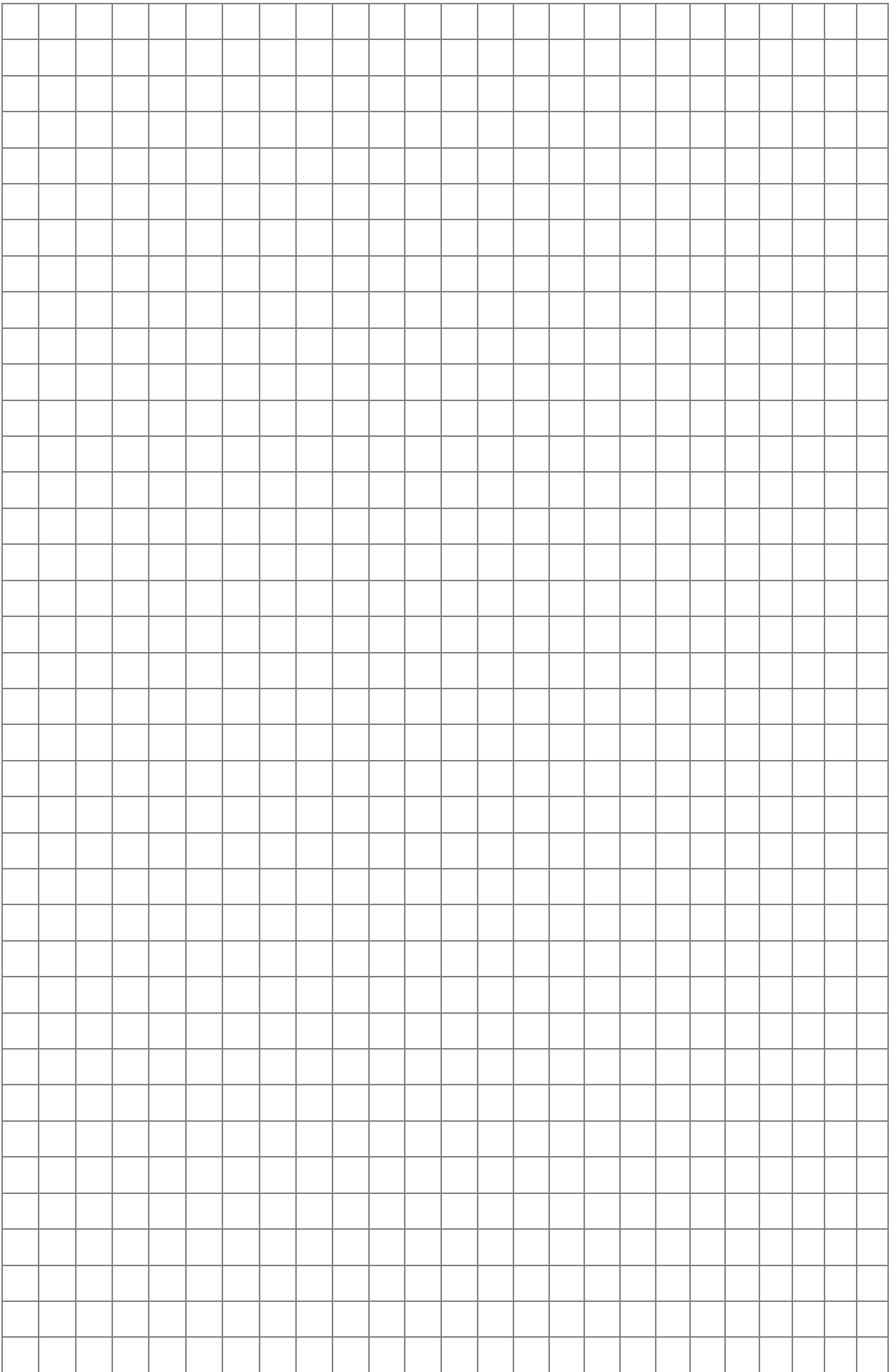
### Задания для самостоятельного выполнения

1. Постройте инфологическую модель предметной области согласно своему варианту (см. ниже).
2. Представьте результат анализа в виде даталогической схемы БД.
3. Создайте таблицы БД в среде Access и заполните их данными (не менее 10 записей в каждой таблице).
4. Проанализируйте данные в таблицах на предмет их согласованности.
5. Результаты работы по всем этапам отразите в отчете.

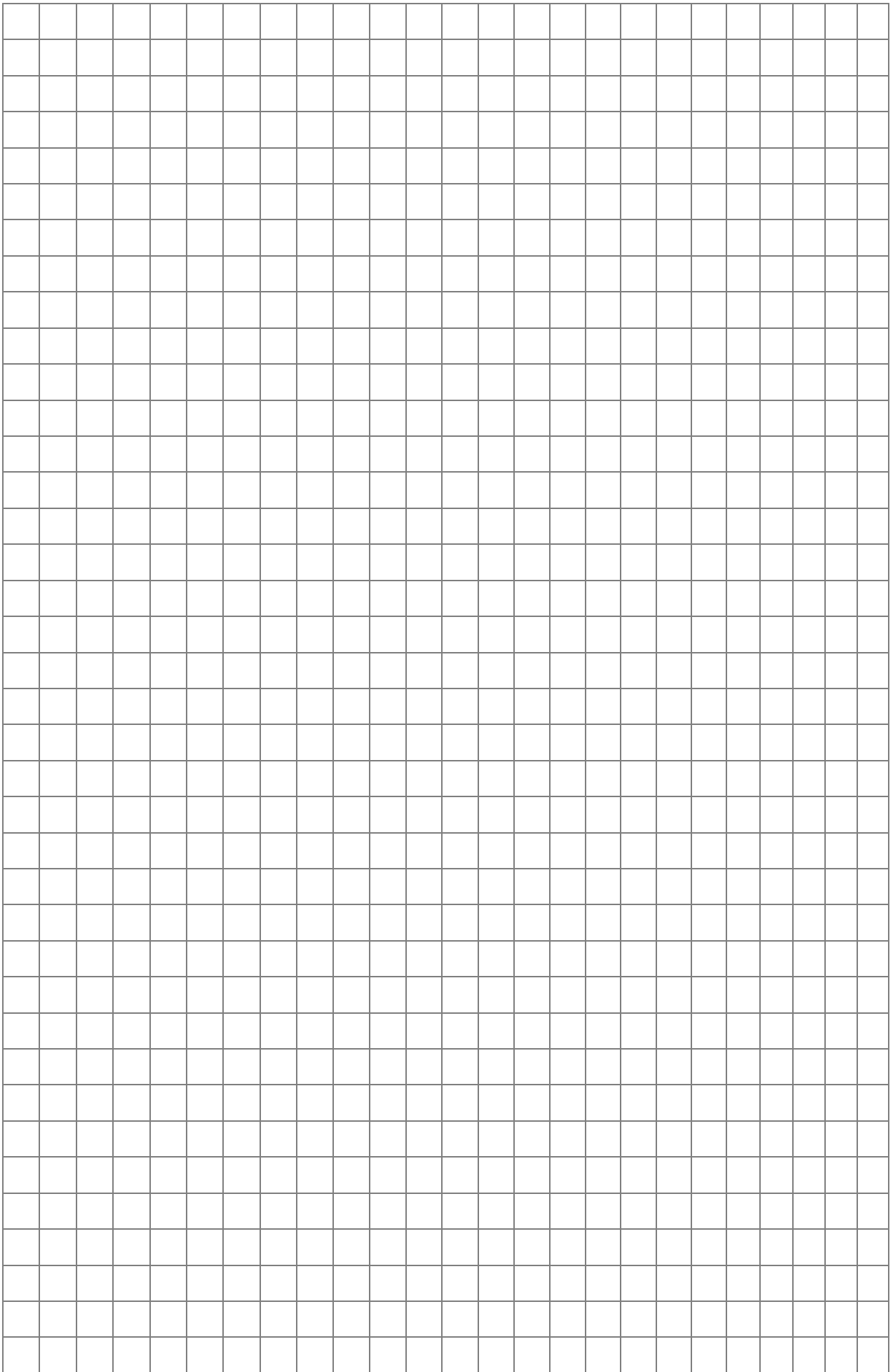
## **Варианты:**

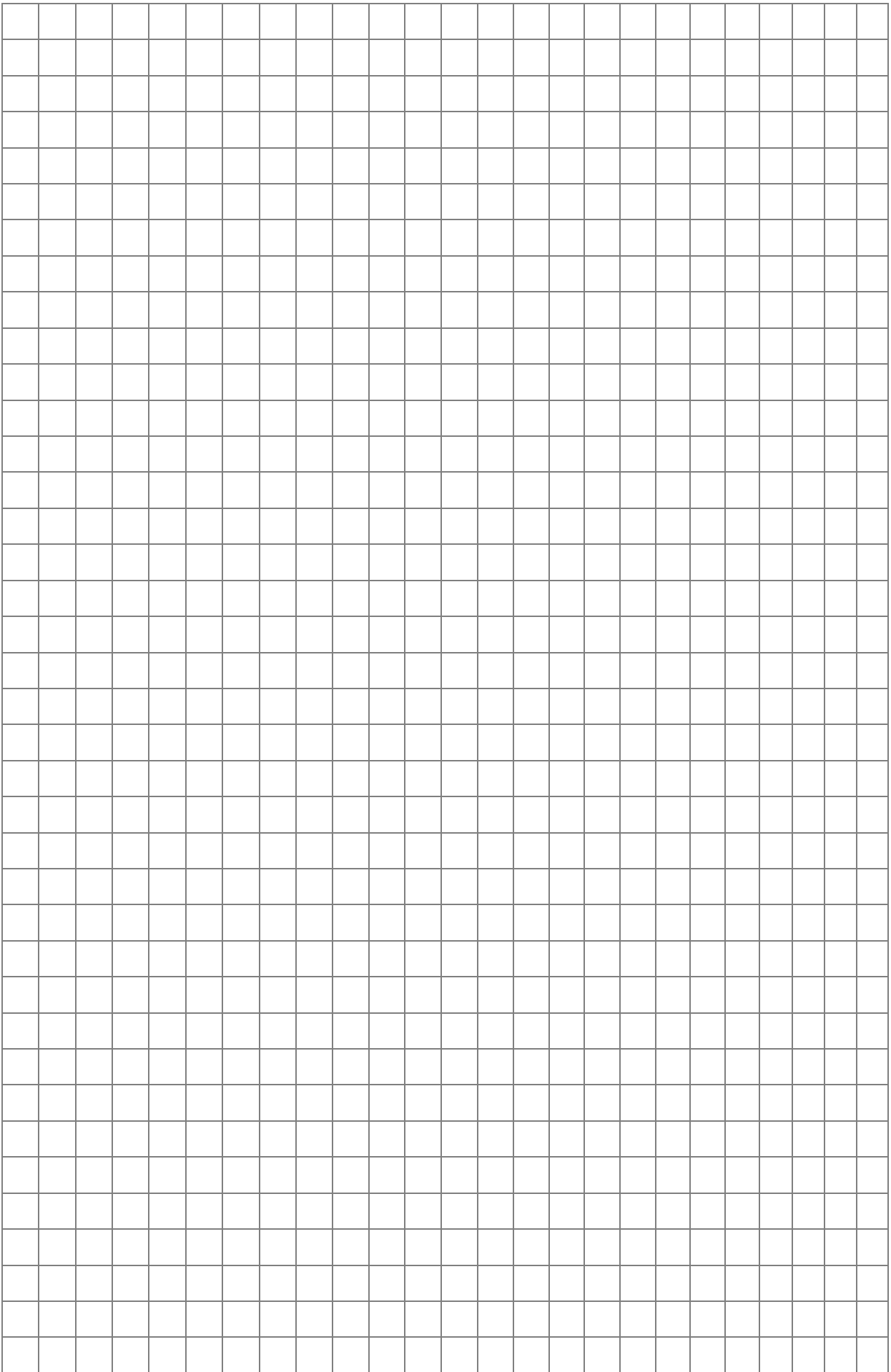
1. Расписание автобусов: номер рейса, конечный и промежуточный пункты, время отправления.
2. База абитуриентов: анкетные данные, совокупность оценок на вступительных экзаменах, готовность учиться на договорной основе.
3. База предложений по обмену: район, площадь, планировка и т.д.; требования к вариантам обмена.
4. Справочник почтовой индексации. Республика, область (край), район, населенный пункт, почтовый индекс.
5. База продавцов: наименование товара, объём партии при оптовой продаже, цена, условия продажи-отгрузки, форма оплаты, контактный адрес или телефон, примечание (например, «посредников прошу не беспокоить»). База покупателей: наименование товара, объём покупки, приемлемая цена и форма оплаты, контактный адрес или телефон, примечание.
6. Справочник любителя живописи. Художники с анкетными данными и стилями. Картины со ссылкой на художников, датой создания, жанром. Коллекционеры и музеи: наличие оригиналов картин и копий. Аукционы и комиссионки: дата проведения, список выставленных шедевров и цены на них. Собственная коллекция.
7. Справочник географа. Города (географические координаты, численность населения), регионы (области, провинции, штаты и т.д. – принадлежность стране, столица, численность населения), страны (площадь, численность населения, форма государственного правления, столица), материка.
8. Реки мира: протяженность, куда впадает, годовой сток, площадь бассейна.
9. Видимые звезды: название, созвездие, видимая звездная величина, расстояние от Земли, координаты на небосклоне: прямое восхождение (ч, мин) и склонение (град., мин).
10. Марки: страна, нарицательная стоимость, год выпуска, тираж, особенности. Филателисты: страна, имя, контактные координаты, наличие редких марок в коллекции.
11. База безработных: анкетные данные, профессия, образование, место и должность последней работы, причина увольнения, семейное положение, жилищные условия, контактные координаты, требования к будущей работе. База вакансий: фирма,

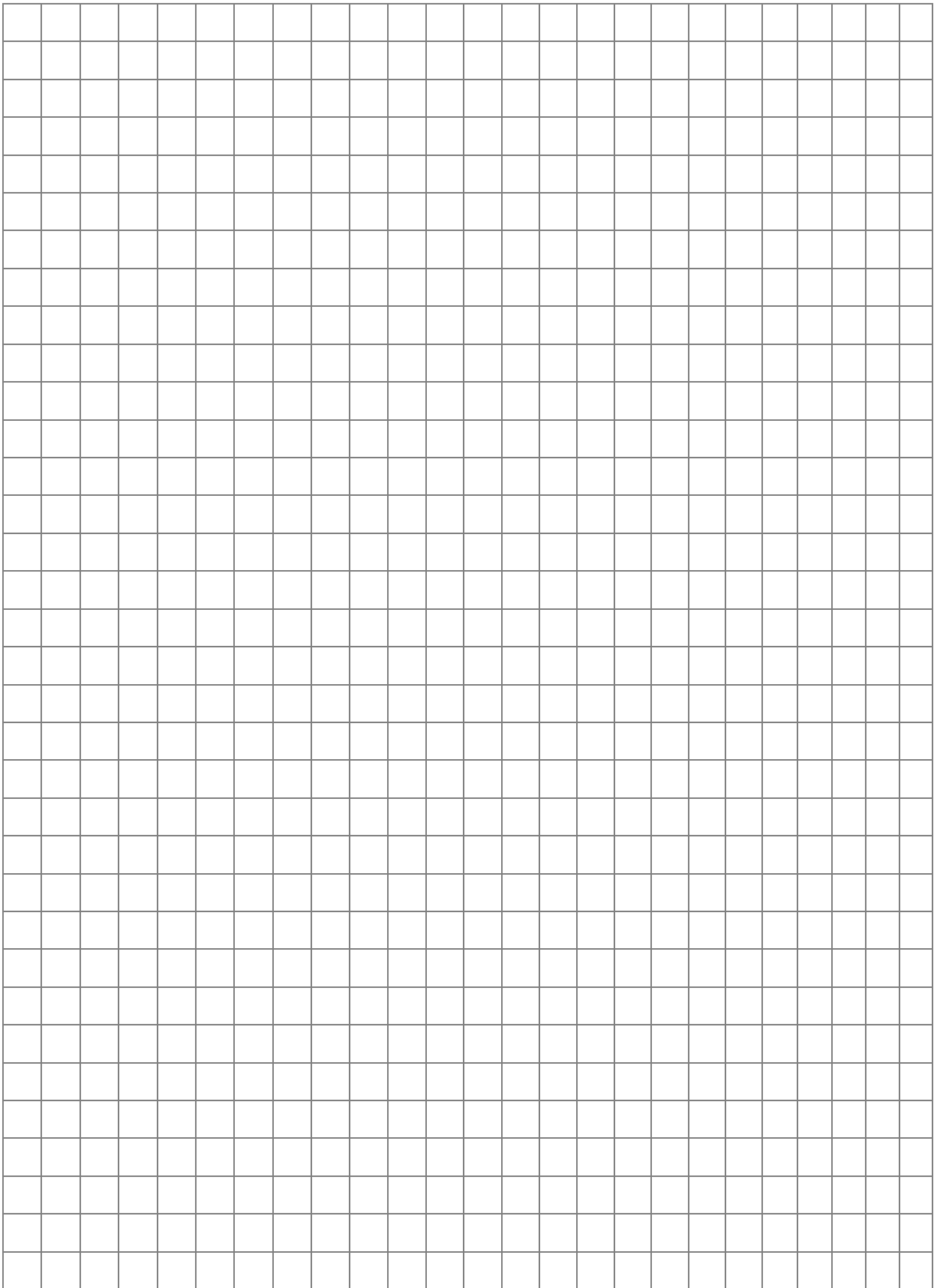












## Тема 4. Функционально-ориентированное проектирование ИС

*Цель:* изучить основы функционально-ориентированного проектирования ИС.

### Теоретические сведения

#### 1. Технология функционально-ориентированного проектирования ИС

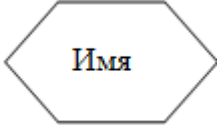
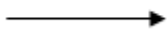
Основными идеями функционально-ориентированной CASE-технологии являются идеи структурного анализа и проектирования ИС. Сущность структурного подхода к разработке ИС заключается в её декомпозиции на автоматизируемые функции: система разбивается на функциональные подсистемы, которые, в свою очередь, делятся на подфункции, подразделяемые на задачи и т.д. Процесс разбиения продолжается вплоть до конкретных процедур. При этом автоматизируемая система сохраняет целостная представление, в котором все составляющие компоненты взаимосвязаны.

В качестве инструментальных средств структурного анализа и проектирования выступают следующие диаграммы:

- диаграмма бизнес-функций (BFD – Business Function Diagram);
- диаграмма потоков данных (DFD – Data Flow Diagram);
- диаграмма переходов состояний (STD – State Transition Diagram);
- модель «сущность-связь» (ERD – Entity Relationship Diagram);
- диаграмма структуры программного приложения (SSD – System Structure Diagram) [18].

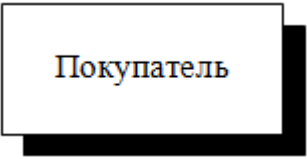
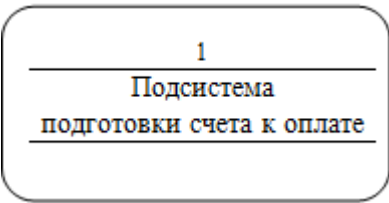
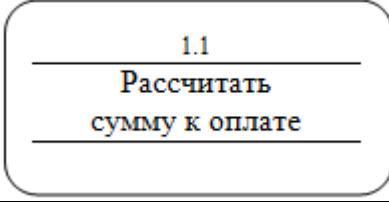
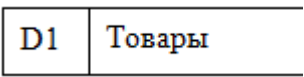

*Диаграммы функциональных спецификаций* (диаграммы бизнес-функций) отражают взаимосвязь различных задач процессе получения требуемых результатов. В таблице 4.1 приведены основные объекты BFD в нотации SAG.

Таблица 4.1

Объект	Описание	Обозначение
Функция	некоторое действие ИС, необходимое для решения экономической задачи	
Декомпозиция функции	разбиение функции на множество подфункций	

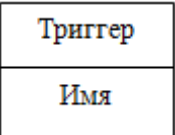
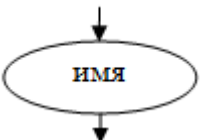
**Диаграммы потоков данных** отражают передачу информации от одной функции к другой в рамках заданной технологии обработки. В таблице 4.2 приведены основные объекты DFD.

Таблица 4.2

Объект	Описание	Обозначение
Внешняя сущность	материальный предмет или физическое лицо, представляющее собой источник или приемник информации	
Система/подсистема	номер подсистемы служит для её идентификации, в поле имени вводится наименование подсистемы	
Процесс	преобразование входных потоков данных в выходные в соответствии с определенным алгоритмом	
Накопитель данных	абстрактное устройство для хранения информации, которую можно в любой момент поместить в накопитель и через некоторое время извлечь	
Потоки данных	информация, передаваемая через некоторое соединение от источника к приемнику	

**Диаграммы переходов состояний** модулируют поведение системы во времени в зависимости от происходящих событий (нажатая клавиша, дата отчетного периода и т.д.). В таблице 4.3 приведены основные объекты STD.

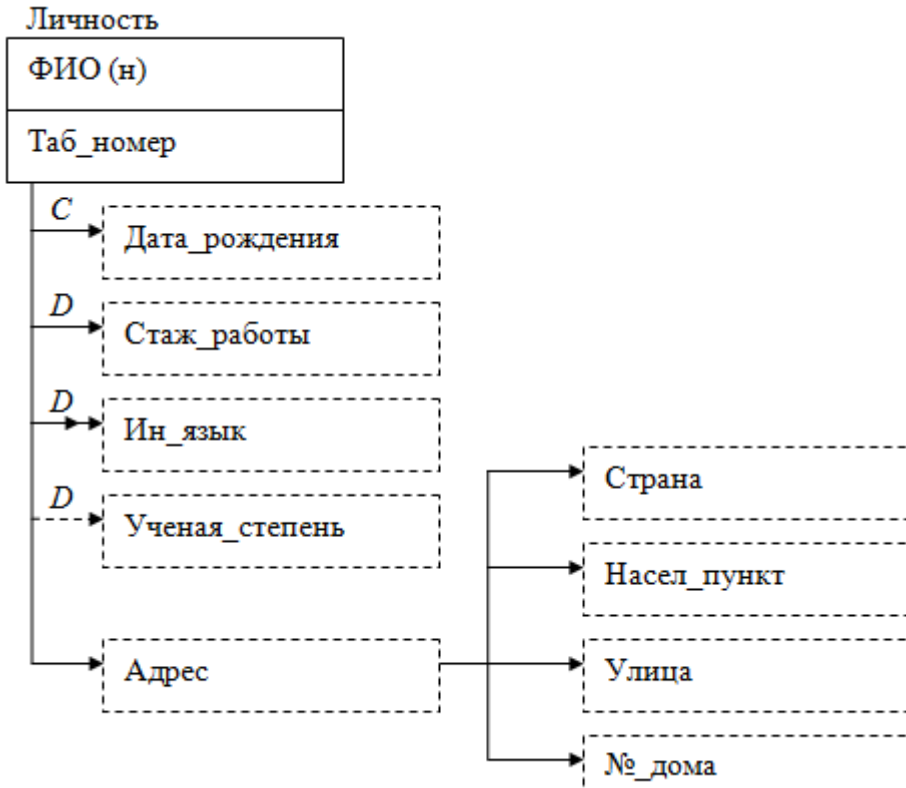



Таблица 4.3

Объект	Описание	Обозначение
Состояние	устойчивое значение некоторого свойства в течение определенного времени	
Условие перехода	условие по данным	

	условие по времени	
--	--------------------	---

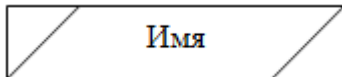
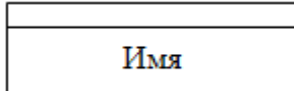
**Диаграмма «сущность-связь»** отражает множества объектов и их характеристик, а также взаимосвязей между ними, которые в дальнейшем используются функциями проектируемой системы. В таблице 2.4 приведены основные объекты ERD.

Таблица 4.4

Объект	Обозначение
Сущность	
Связь 1:1	
Связь 1:M	
Связь M:M	

**Диаграмма структуры программного приложения** задает взаимосвязь функций и программных модулей, которые их реализуют (меню, формы, отчеты и т.д.). В таблице 4.5 приведены основные объекты ERD.

Таблица 4.5

Объект	Обозначение
Модуль	
Библиотечный модуль	

## 2. Этапы функционально-ориентированного проектирования ИС

Процесс функционально-ориентированного проектирования содержит следующие этапы [18]:

1. Инициализация проекта. На основании материалов обследования создается новый репозиторий.

2. Задание начальных параметров проекта. Выбирается методология проектирования, и в её рамках определяется нотация.

Операции 3 – 6 выполняются последовательно-параллельно и взаимно уточняются в ходе выполнения.

3. Построение диаграммы иерархии функций.

4. Построение диаграммы потоков данных.

5. Построение диаграммы переходов состояний.

6. Построение диаграммы «сущность-связь».

7. Построение системной структурной диаграммы.

8. Генерация описания схемы БД.

9. Генерация описания системы БД.

10. Генерация приложения.

11. Интеграция модулей приложения.

### *Пример: Счет-фактура*

Проведем анализ предметной области для проектирования системы подготовки счета-фактуры (см. Рис. 4.1) на основе функционально-ориентированной технологии. Для этого необходимо построить 5 диаграмм: иерархии функций, потоков данных, переходов состояний, «сущность-связь», системной структурной диаграммы.

Приложение №1  
к Правилам ведения журналов учета полученных и выставленных счетов-фактур,  
книг покупок и книг продаж при расчётах по налогу на добавленную стоимость,  
утвержденным постановлением Правительства Российской Федерации от 2 декабря 2000 г. N 814  
(в редакции постановлений Правительства Российской Федерации  
от 15 марта 2001 г. N 189, от 27 июля 2002 г. N 575, от 16 февраля 2004 г. N 84, от 11 мая 2006 г. N 283)

### СЧЕТ-ФАКТУРА № 000289 от 12 июня 2007 г.

Продавец: ЗАО "Милана"  
Адрес: 129366, Москва, Ракетный бульвар, 17  
ИНН/КПП продавца: 7717027908/671010011  
Грузоотправитель и его адрес: ЗАО "Милана", Адрес: 129366, Москва, Ракетный бульвар, 17  
Грузополучатель и его адрес: ООО "Чемпион", Адрес: 129366, Москва, Ракетный бульвар, 1  
К платежно-расчетному документу \_\_\_\_\_ от \_\_\_\_\_  
Попуатель: ООО "Чемпион"  
Адрес: 129366, Москва, Ракетный бульвар, 1  
ИНН/КПП покупателя: 7717000408/671010011

Валюта: руб.

Наименование товара (описание выполненных работ, оказанных услуг), имущественного права	Единица измерения	Количество	Цена (тариф) за единицу измерения	Стоимость товаров (работ, услуг), имущественных прав, всего без налога	В том числе акциз	Налоговая ставка	Сумма налога	Стоимость товаров (работ, услуг), имущественных прав, всего с учетом налога	Страна происхождения	Номер таможенной декларации
1	2	3	4	5	6	7	8	9	10	11
Монитор 17" Samsung 710N (SKN)	шт.	5.000	4'631.56	23'157.80	---	18%	4'168.40	27'326.20	Китай	10210130211206/0017348/1
Принтер HP LaserJet 1020 Q5911A A4, 600x600dpi, 14ppm, USB	шт.	2.000	3'461.97	6'923.93	---	18%	1'246.31	8'170.24		
Сканер Bear Paw 2448 TA Pro (A4, 1200x2400, 48bit, USB, Slide)	шт.	1.000	1'711.64	1'711.64	---	18%	308.10	2'019.74		
<b>Всего к оплате</b>				<b>5722.81</b>			<b>5722.81</b>	<b>37'516.18</b>		

Руководитель организации \_\_\_\_\_ (подпись) \_\_\_\_\_ (ф.и.о) Иванова В.П. \_\_\_\_\_ (ф.и.о)  
Иванов В.П.

Индивидуальный предприниматель \_\_\_\_\_ (подпись) \_\_\_\_\_ (ф.и.о) Петрова Г.С. \_\_\_\_\_ (ф.и.о)  
Петрова Г.С.

ПРИМЕЧАНИЕ. Первый экземпляр - покупателю, второй экземпляр - продавцу

Рис. 2.1



## 1. Диаграмма иерархии функции.

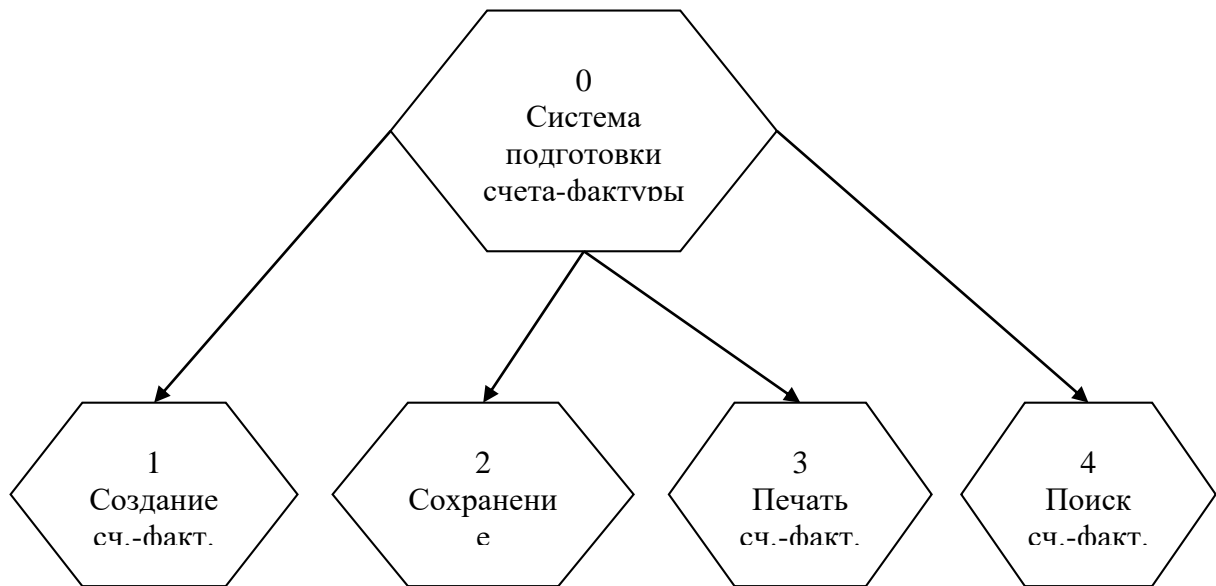


Рис. 4.2. Диаграмма иерархии функции

## 2. Диаграмма потоков данных.

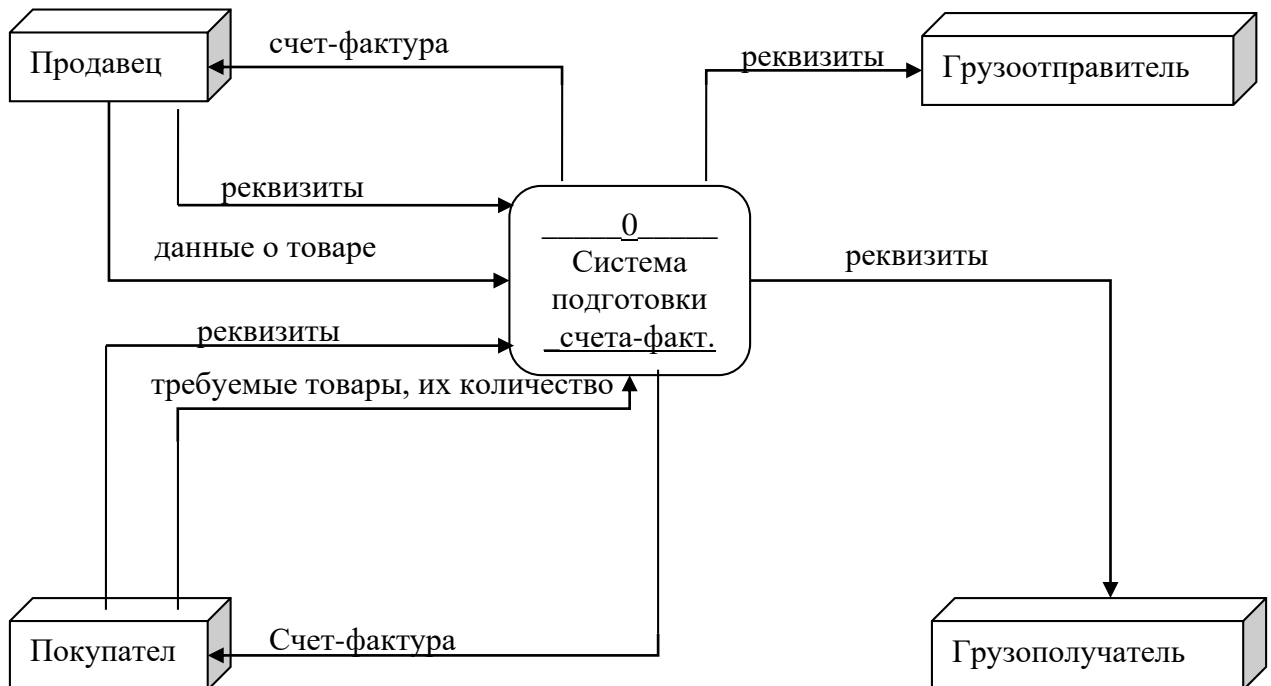


Рис. 4.3. Диаграмма потоков данных

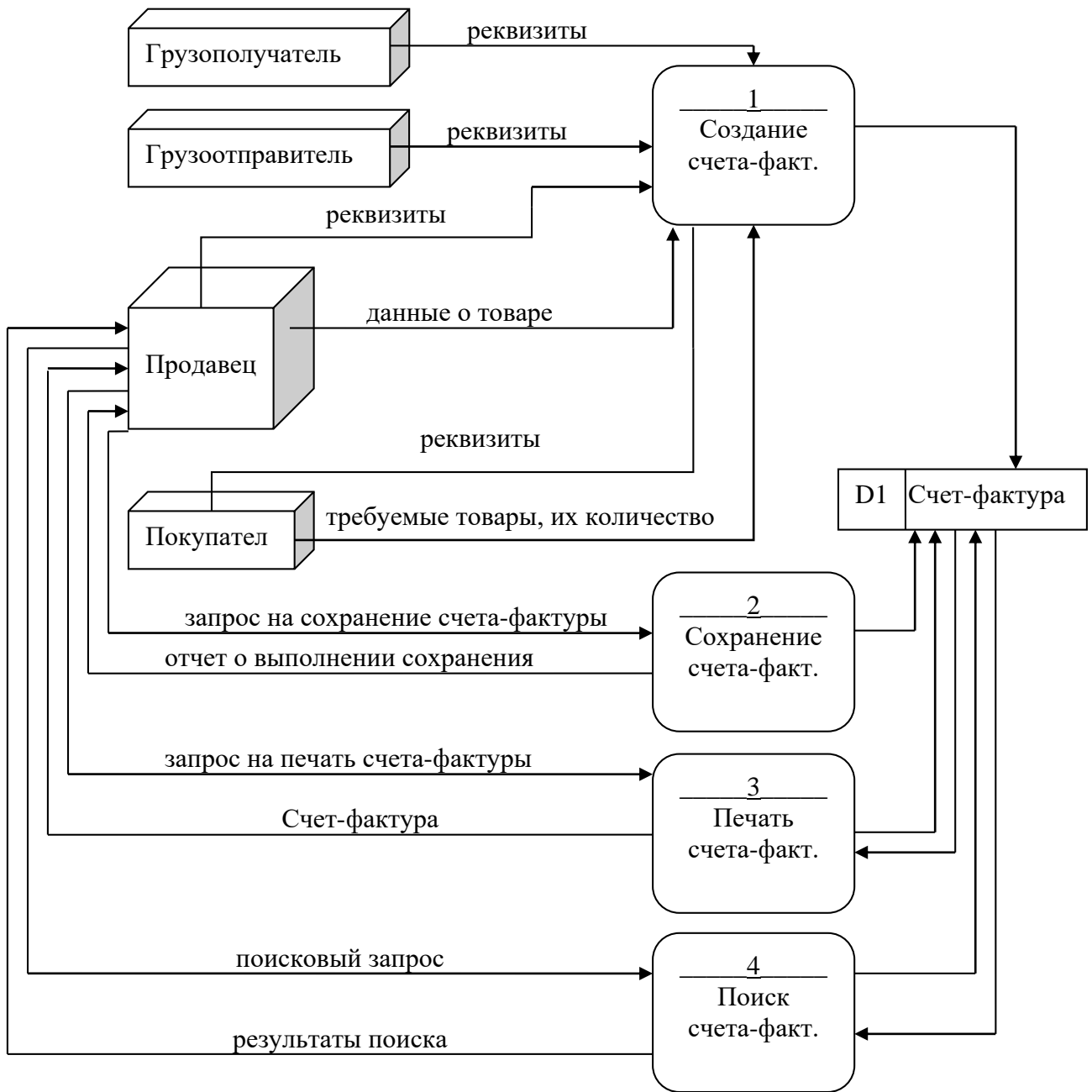


Рис. 4.4. Диаграмма потоков данных

### 3. Диаграмма переходов состояний.

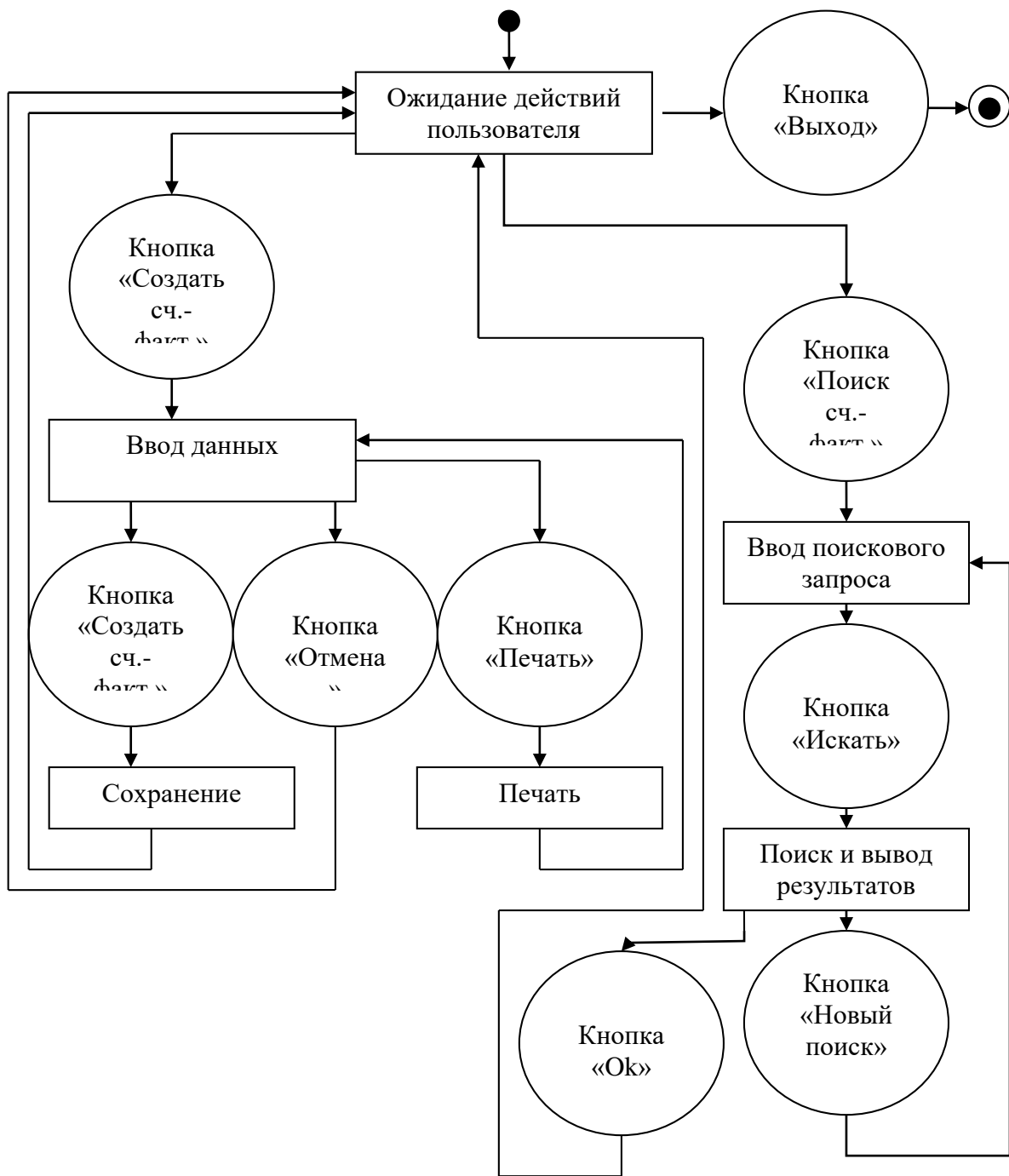


Рис. 4.5. Диаграмма переходов состояний

#### 4. Диаграмма «сущность-связь»

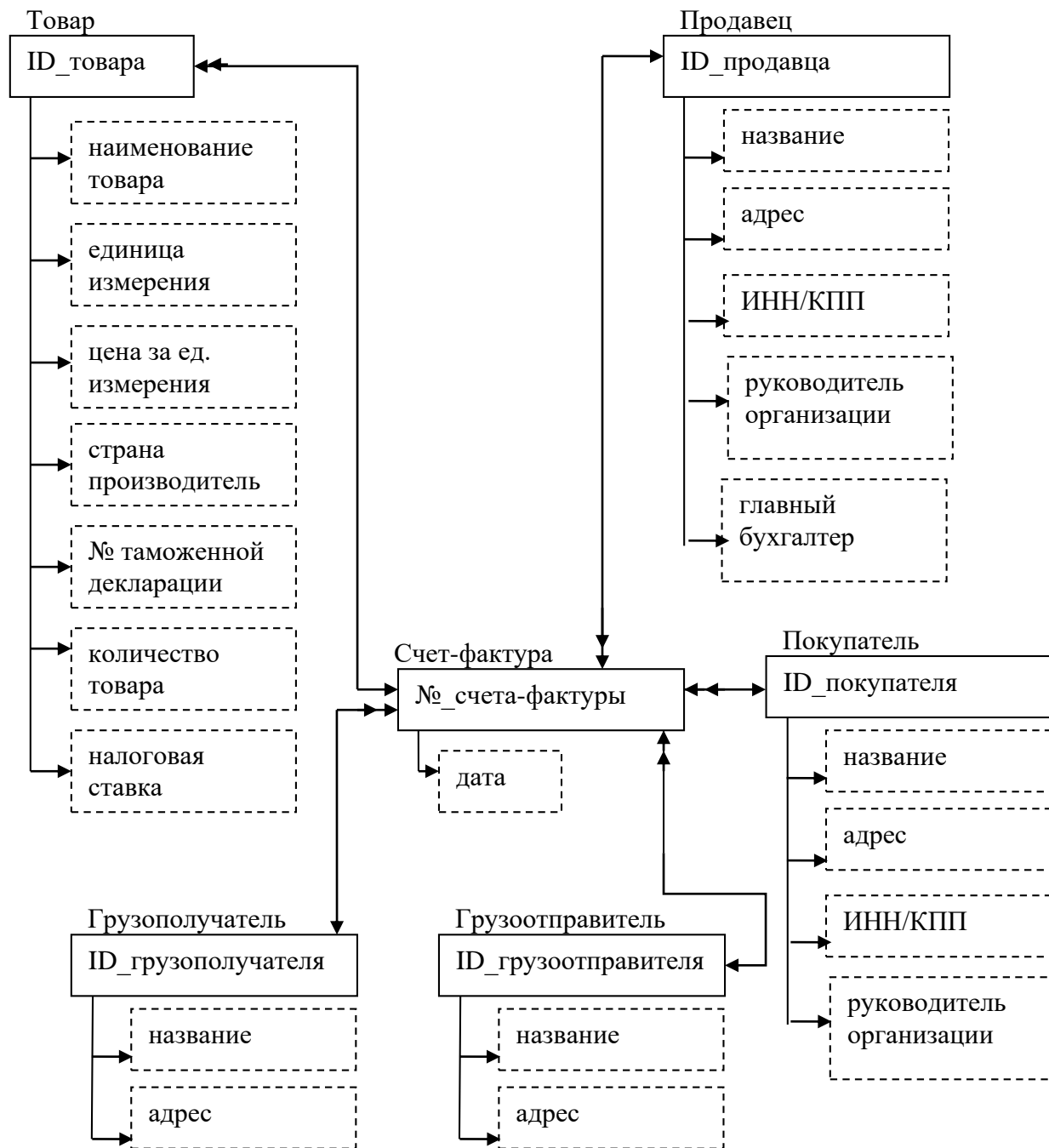


Рис. 4.6.

#### 5. Диаграмма структуры программного приложения.

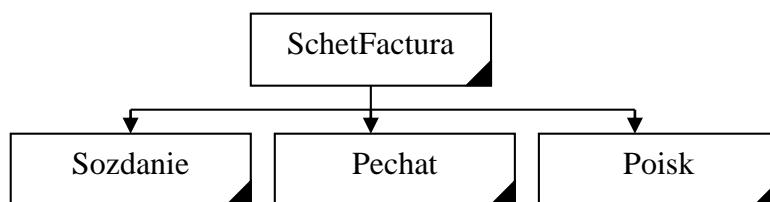


Рис. 4.7

## Задания для самостоятельного выполнения

1. Проведите функционально-ориентированный анализ предметной области своего варианта (см. варианты ниже).  
Приведите 5 построенных диаграмм в письменном отчете.

### *Информационные системы подготовки следующих документов:*

- 101) Авансовый отчет.
- 102) Академическая справка.
- 103) Акт о неисполнении трудовых обязанностей.
- 104) Акт о приемке выполненных работ.
- 105) Акт приема-передачи автомобиля.
- 106) Акт сдачи-приемки товара.
- 107) Анкета поступающего на работу.
- 108) Балльно-рейтинговый лист студентов.
- 109) Банковская книжка.
- 110) Брачный договор.
- 111) Бухгалтерский баланс.
- 112) Генеральная доверенность.
- 113) График отпусков.
- 114) Доверенность на ведение дел в суде.
- 115) Доверенность на получение материальных ценностей.
- 116) Доверенность на получение пенсии.
- 117) Договор банковского вклада.
- 118) Договор дарения квартиры.
- 119) Договор купли-продажи квартиры.
- 120) Договор купли-продажи транспортного средства.
- 121) Договор мены квартир.
- 122) Договор на реализацию продукции покупателям.
- 123) Договор о займе денег.
- 124) Договор о полной индивидуальной материальной ответственности.
- 125) Журнал регистрации амбулаторных больных.
- 126) Журнал регистрации приходных и расходных кассовых документов.
- 127) Журнал учета пропущенных и замещенных уроков.

- 128) Журнал факультативных занятий.
- 129) Записка-расчет о предоставлении отпуска.
- 130) Записка-расчет при прекращении трудового договора с работником.
- 131) Заявление о выдаче водительского удостоверения.
- 132) Заявление о выдаче паспорта.
- 133) Заявление о государственной регистрации транспортного средства.
- 134) Заявление о расторжении договора банковского счета.
- 135) Инвентаризационная опись основных средств.
- 136) Исковое заявление о взыскании алиментов на ребенка.
- 137) Исковое заявление о взыскании заработной платы.
- 138) Исковое заявление о возврате вклада и защите прав потребителя.
- 139) Исковое заявление о возмещении ущерба, причиненного заливом квартиры.
- 140) Исковое заявление о расторжении брака.
- 141) Исковое заявление о снятии дисциплинарного взыскания.
- 142) Исполнительская надпись.
- 143) Карта учета диспансеризации.
- 144) Квитанция о принятии денежных средств в депозит.
- 145) Книга покупок.
- 146) Книга продаж.
- 147) Командировочное удостоверение.
- 148) Лист нетрудоспособности.
- 149) Личная карточка сотрудника.
- 150) Медицинская карта.
- 151) Медицинский рецепт.
- 152) Накладная на получение материальных ценностей.
- 153) Налоговая декларация по транспортному налогу.
- 154) Направление на анализы пациента.
- 155) Наряд на выполнение работ.
- 156) Опись вложения.
- 157) Опись дел.

- 158) Отчет о движении денежных средств.
- 159) Отчет о кассовых поступлениях и выбытиях.
- 160) Отчет о целевом использовании полученных средств.
- 161) Отчет об изменениях капитала.
- 162) Отчет об использовании акцизных марок.
- 163) Отчета о прибылях и убытках.
- 164) Паспорт больного аллергическим заболеванием.
- 165) Платежное требование.
- 166) Приемный акт на материальные ценности.
- 167) Приказ о наложении дисциплинарного взыскания.
- 168) Приказ о переводе работника на другую работу.
- 169) Приказ о поощрении работника.
- 170) Приказ о приеме на работу.
- 171) Приказ об отпуске сотрудника.
- 172) Приказ об увольнении.
- 173) Протокол осмотра и исследования вещественных доказательств.
- 174) Распоряжение об отмене доверенности.
- 175) Расчет платы за негативное воздействие на окружающую среду.
- 176) Расчетный листок.
- 177) Регистрационные карточки внутренних приказов и распоряжений предприятия (организации) документов.
- 178) Регистрационные карточки исходящих документов предприятия (организации).
- 179) Реестр сведений о доходах физических лиц.
- 180) Свидетельство о рождении.
- 181) Сводная ведомость успеваемости группы студентов.
- 182) Служебное задание для направления в командировку.
- 183) Согласие на выезд детей.
- 184) Соглашение о месте жительства ребенка при раздельном проживании родителей.
- 185) Соглашение об уплате алиментов.
- 186) Справка о заработной плате.

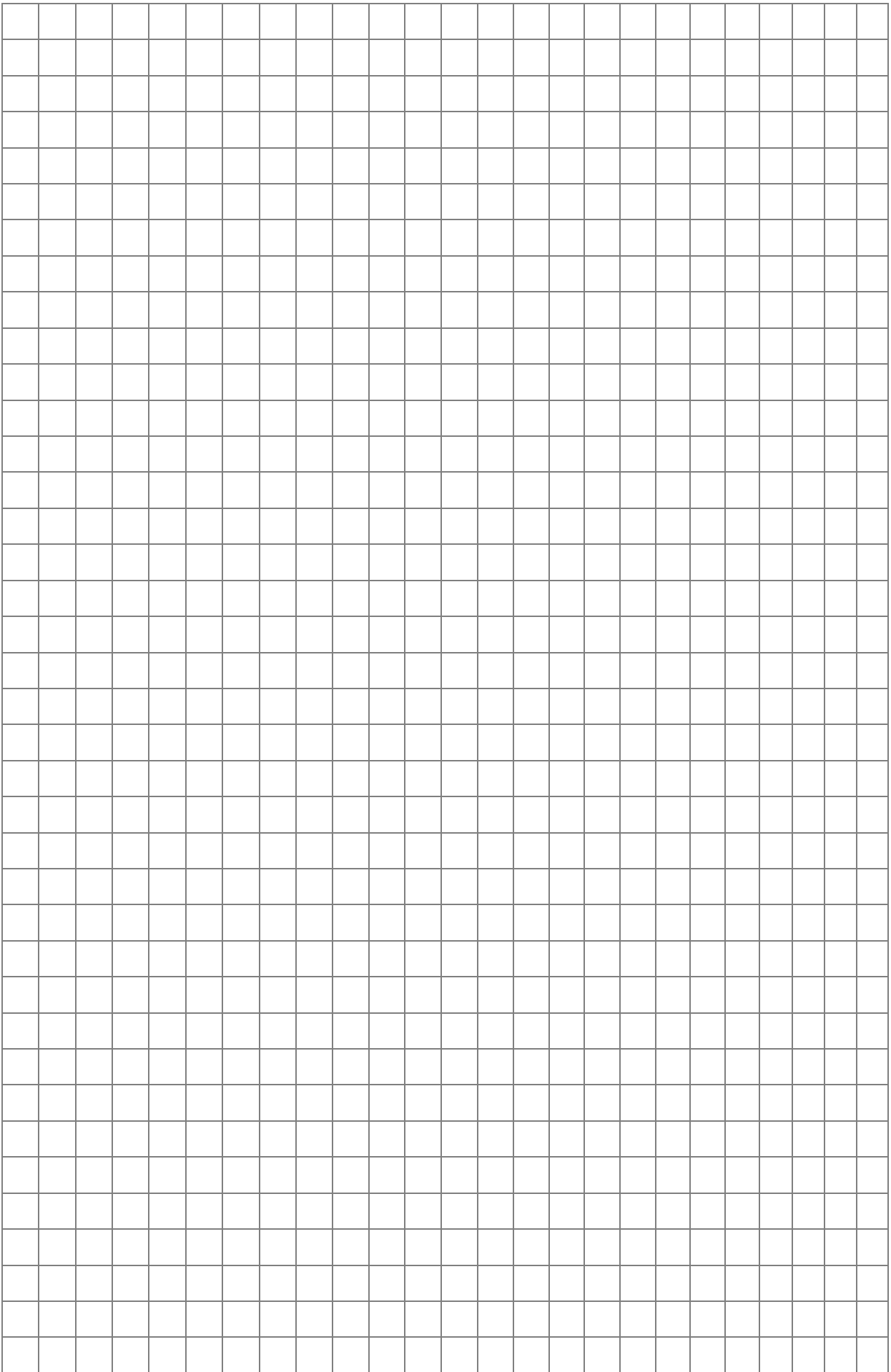
- 187) Справка о составе семьи.
- 188) Справка об обучении на факультете.
- 189) Справка с места работы.
- 190) Срочный трудовой договор.
- 191) Счета на оплату предоставляемых товаров (услуг).
- 192) Таможенная декларация.
- 193) Температурный лист.
- 194) Товарный ценник.
- 195) Товарный чек.
- 196) Трудовая книжка.
- 197) Туристическая путевка.
- 198) Уведомление о необходимости получения трудовой книжки.
- 199) Уведомление работника об истечении срока трудового договора.
- 200) Штатное расписание.

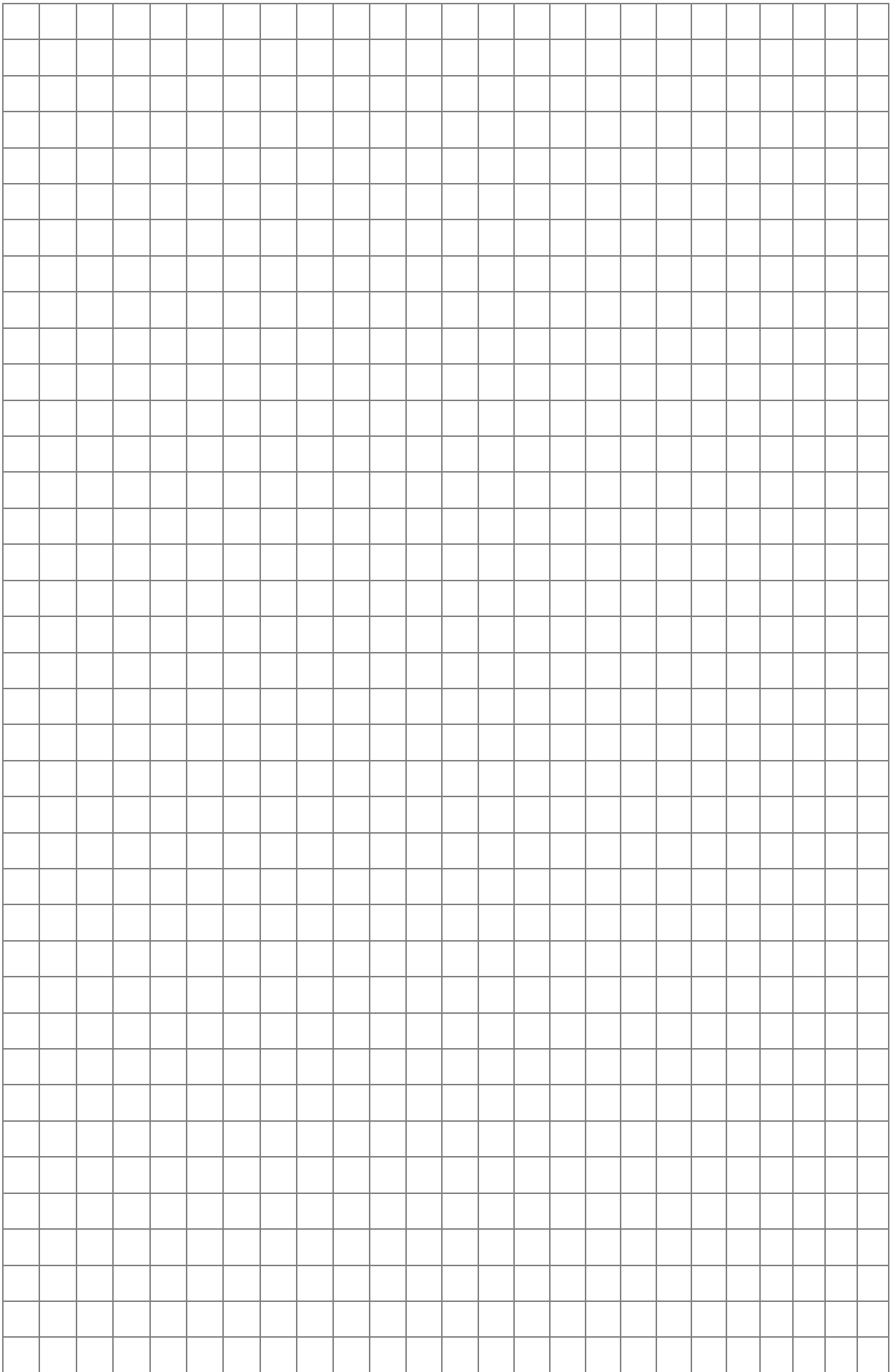
**2. Ответьте на вопросы:**

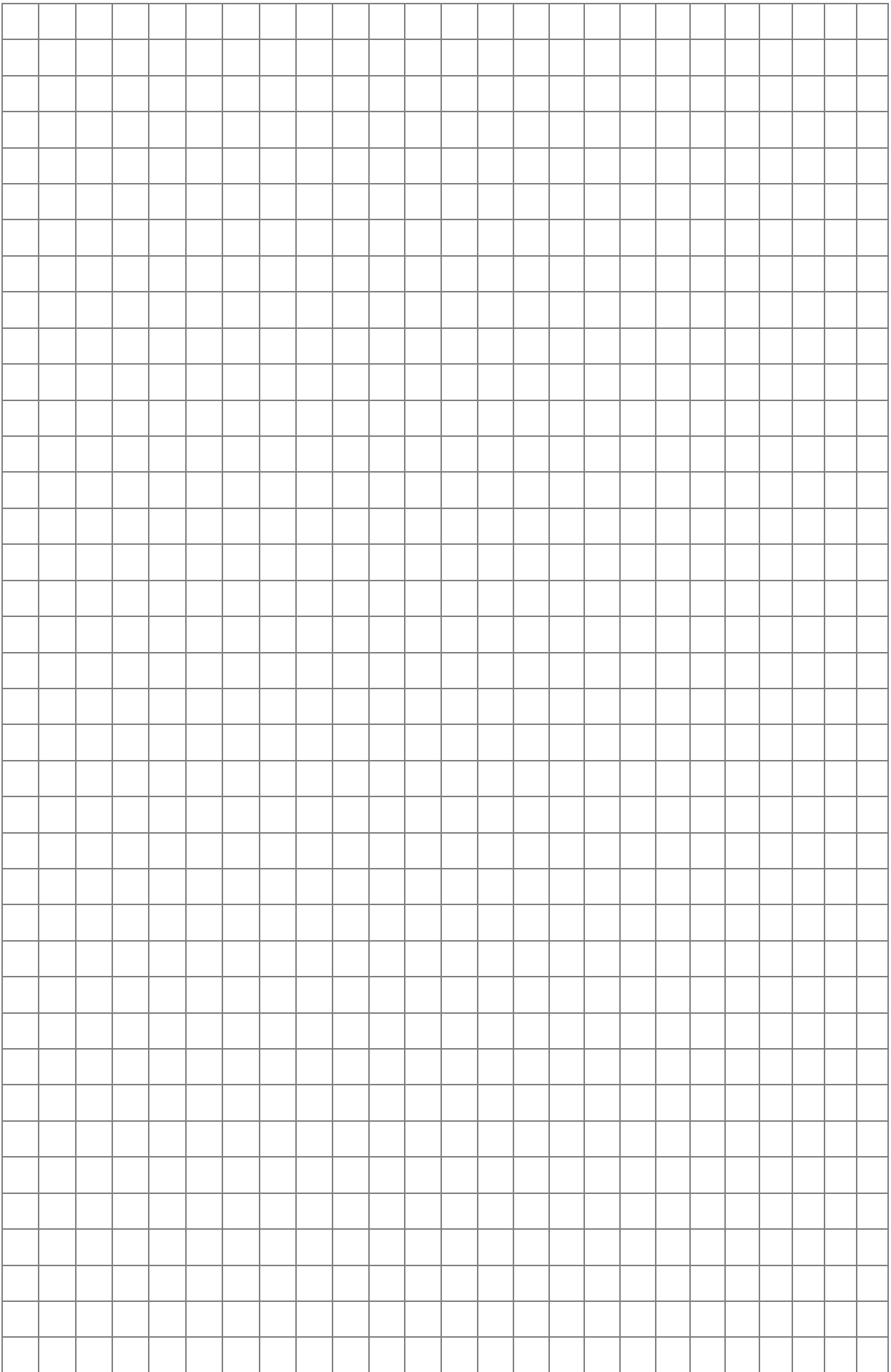
- а)** Дайте определение CASE-технологии проектирования ИС? Какова структура CASE-средства?
- б)** Как можно определить функционально-ориентированную CASE-технологию? Какие диаграммы выступают в качестве инструментальных средств функционально-ориентированного анализа и проектирования?
- в)** Определите основные понятия и конструктивные элементы диаграммы функциональных спецификаций?
- г)** Зачем создаются диаграммы потоков данных? Определите основные понятия и конструктивные элементы диаграммы потоков данных.
- д)** Зачем создаются диаграммы переходов состояний? Определите основные понятия и конструктивные элементы диаграммы переходов состояний.
- е)** Зачем создаются диаграммы «сущность-связь»? Определите основные понятия и конструктивные элементы диаграммы «сущность-связь»?
- ж)** Зачем создаются системные структурные диаграммы? Определите основные понятия и конструктивные элементы диаграммы системной структуры.











## Тема 5. Организация доступа к базам данных из

**Цель:** сформировать навыки организации доступа к базам данных из Delphi.

### Теоретические сведения

В Delphi изначально доступ к базам данных обеспечивался процессором баз данных *BDE* (Borland Database Engine). Начиная с пятой версии Delphi, добавлена новая возможность доступа к базам данных с использованием технологии *ADO* (ActiveX Data Objects), разработанной и поддерживаемой фирмой Microsoft [6, 7, 14].

Для отображения и редактирования данных, хранящихся в базах данных, в визуальной библиотеке компонентов VCL реализован ряд компонентов, специально ориентированных на работу с базами данных.

Любое приложение, работающее с базами данных, должно обеспечивать ряд типовых функциональных возможностей, включающих: подключение к базе данных, считывание информации из таблиц этой базы данных, редактирование данных и навигацию по набору данных. В Delphi указанные функции обеспечиваются с помощью любой из двух упомянутых выше технологий: BDE либо ADO. Рассмотрим механизмы доступа к данным с помощью технологии ADO.

### 1. Доступ к данным с использованием ADO

Все использующие ADO компоненты доступа к данным размещены на странице dbGo (в ранних версиях ADO) палитры компонентов.

Основные компоненты – это TADOTable  и TADOQuery .

Компонент **TADOTable** обеспечивает использование в приложениях Delphi таблиц БД, подключенных через провайдеры OLE DB.

Имя таблицы БД задается свойством: *TableName: WideString*;

Компонент **TADOQuery** обеспечивает применение запросов SQL при работе с данными через ADO.

Текст запроса задается свойством:

property SQL: TStrings;

Если запрос должен возвращать набор данных, для его открытия используется свойство:

property Active: Boolean;

...или метод:

procedure Open;

В противном случае достаточно использовать метод:

function ExecSQL: Integer;

Число обработанных запросом записей возвращает свойство:

property RowsAffected: Integer;


База данных, с которой связываются объекты TADOTable и TADOQuery, указывается с помощью свойства *ConnectionString*.

## 2. Компоненты Delphi для отображения и редактирования данных


Для полноценной работы с БД недостаточно только обеспечить доступ к информации, хранящейся в базе. Необходимы также возможности визуализации и редактирования этой информации. В VCL Delphi имеется целый ряд компонентов, предназначенных для визуализации и редактирования данных, содержащихся в полях набора данных. Все эти компоненты размещены на вкладке **Data Controls** палитры компонентов.

Для обеспечения взаимодействия между набором данных и элементами отображения и редактирования данных используется специальный компонент TDataSource, расположенный на вкладке **Data Access** палитры компонентов.

### 2.1. Класс TDataSource

Класс TDataSource  используется в качестве интерфейса для соединения набора данных с компонентами отображения данных. Он обеспечивает передачу в компоненты отображения данных значений полей из набора данных и внесение в набор данных изменений при редактировании этих значений. Компонент TDataSource передает в элементы отображения данных значения полей текущей записи. При перемещении курсора набора данных на другую запись значения полей в компонентах отображения данных автоматически обновляются [6, 7, 14].

## 2.2. Класс *TDBGrid*

Компонент **TDBGrid**  используется для представления набора данных в виде таблицы. Структура этой таблицы соответствует структуре набора данных: строки являются записями, а столбцы – полями.


### 2.3. Компоненты для доступа к отдельным полям

Для доступа к отдельным полям базы данных в VCL Delphi имеется ряд элементов, аналогичных обычным элементам управления, рассмотренным нами ранее. Отличие заключается только в том, что элементы, работающие с базой данных, получают значения напрямую из набора данных и изменения этих значений заносятся в базу данных.


Все элементы отображения данных, отвечающие за доступ к отдельным полям набора данных, имеют два общих свойства:

- *DataSource: TDataSource* – указывает на источник данных, с которым связан компонент отображения данных;
- *DataField: String* – имя поля набора данных, из которого элемент отображения данных получает информацию.


#### *TDBText*

Компонент **TDBText**  отображает текущее значение поля набора данных. Редактировать значение поля с помощью этого элемента нельзя (он аналогичен компоненту TLabel).


#### *TDBEdit*

Компонент **TDBEdit**  представляет собой обычную строку ввода, аналогичную TEdit. В отличие от TDBText с помощью данного компонента можно не только просматривать базу данных, но и редактировать ее.

#### *TDBMemo*


**TDBMemo**  предназначен для отображения и редактирования полей, содержащих несколько строк текста (аналогичен компоненту TMemo).

#### *TDBCheckBox*

Компонент **TDBCheckBox**  (аналогичен компоненту TCheckBox) предназначен для просмотра и редактирования данных, которые могут принимать только два значения. Состояние флажка определяется свойствами *ValueChecked* и *ValueUnChecked*, а также


значением, содержащемся в поле, с которым связан компонент. По умолчанию *ValueChecked = true* и *ValueUnChecked = false*.

### ***TDBRadioGroup***


Компонент **TDBRadioGroup**  представляет собой группу переключателей, состояние которых зависит от значения связанного с ним поля. Если текущее значение поля соответствует значению какого-либо переключателя, то он включается. При изменении состояния переключателей пользователем в поле заносится значение включенного переключателя.

Свойство *Items* содержит список переключателей. Значения, на которые реагируют переключатели, определяются свойством *Values*. Каждому значению, заданному в списке *Values*, соответствует один переключатель. Текущее значение поля содержится в свойстве *Value*.


### ***TDBListBox***

Компонент **TDBListBox**  служит для отображения текущего значения поля данных и замены его на любое значение из списка. При этом значение поля должно совпадать с одним из элементов списка. В остальном компонент TDBListBox ничем не отличается от TListBox.

### ***TDBComboBox***

**TDBCombobox**  отображает значение поля, связанного с данным компонентом, в строке редактирования. Текущее значение поля можно изменять, выбирая новое значение из выпадающего списка либо редактируя текст в поле ввода. Аналогичен компоненту TComboBox.

### ***TDBImage***

**TDBImage**  Используется для отображения графической информации, хранящейся в базе данных.

Этот компонент похож на TImage, но содержит некоторые дополнительные свойства и методы:


- *AutoDisplay: Boolean* – если для данного свойства установлено значение true, то изображение из связанного поля отображается автоматически, если значение false, то для загрузки изображения необходимо вызывать метод *LoadPicture*;



- *procedure LoadPicture* – загружает изображение из связанного поля;
- *procedure CopyToClipboard* – копирует изображение в буфер обмена;
- *procedure CutToClipboard* – копирует изображение из буфера обмена и обнуляет текущее значение поля;
- *procedure PasteFromClipboard* – загружает изображение из буфера обмена [6, 7, 14].

### ***Навигация по набору данных***

Компоненты, работающие с отдельными полями, не имеют встроенных средств для изменения положения курсора набора данных, добавления новых записей и удаления существующих записей. Поэтому при их использовании требуются дополнительные элементы управления, обеспечивающие навигацию по набору данных.

В VCL Delphi содержится компонент **TDBNavigator** , позволяющий легко решить задачу такой навигации. Этот компонент представляет собой набор кнопок (см. Рис. 5.1), выполняющих следующие функции:

- перемещение курсора на первую запись;
- перемещение курсора набора данных на следующую запись;
- перемещение курсора на предыдущую запись;
- перемещение курсора на последнюю запись;
- вставка новой пустой записи в текущую позицию курсора набора данных;
- удаление текущей записи;
- перевод набора данных в режим редактирования;
- запись изменений в набор данных;
- отмена изменений, внесенных в текущую запись;
- восстановление исходного значения записи.



*Рис. 5.1. Компонент DBNavigator*

Набор кнопок, содержащихся в компоненте TDBNavigator, определяется пользователем с помощью свойства *VisibleButtons*, имеющего тип TButtonSet.

Связь компонента TDBNavigator с набором данных устанавливается через свойство *DataSource*, указывающее на

источник данных, связанный с требуемым набором данных. Свойство *ConfirmDelete*, имеющее тип *Boolean*, позволяет включить запрос подтверждения при удалении записи.

### ***Пример 1: Анкета сотрудника***

В качестве примера разработаем форму для просмотра и редактирования информации, содержащейся в таблице «Физические лица». Данная таблица содержит следующие поля:

- «Код» – используется в качестве первичного ключа, тип целый числовой;
- «Фамилия», «Имя», «Отчество», «Телефон», «Индекс», «Страна», «Город», «Адрес», «Телефон» – текстовые поля;
- «Дата рождения» – поле типа Дата/время;
- «Пол» – поле логического типа.

Для отображения текстовых полей и поля «Дата рождения» будем использовать компоненты *TDBEdit*. Логические поля удобнее отображать с помощью флажков – компонентов *TDBCheckBox*. Кроме того, на форму необходимо поместить элемент *TDBNavigator* для обеспечения навигации по набору данных, а также несколько обычных элементов *TLabel*, с помощью которых будем пояснять назначение полей ввода.

Последовательность действий при создании простых форм будет примерно следующей:

1. Создайте папку и дайте ей имя.
2. По приведенному выше описанию создайте базу данных и сохраните под именем «ФизЛица.mdb» в папку.
3. Создайте приложение Delphi (для этого запустите Delphi, выполнить команды *File → New → VCL Forms Application*).
4. Сохраните приложение в созданную папку (*File → Save project as ...* в появившемся окне выбрать свою папку и ввести имя приложения).
5. Используя вкладку *dbGo* палитры компонентов, разместите на форме компонент *TADOTable*. Затем перейдите в палитре компонентов на вкладку *DataAccess* и установите на форму компонент *TDataSource*. Последний необходим для связи набора данных ADO с компонентами визуализации данных.
6. Теперь необходимо подключить к компоненту *TADOTable* таблицу «Физические лица» базы данных *ФизЛица.mdb*. Выделите на форме компонент *TADOTable* и щелкните на кнопке

с многоточием в поле ввода свойства `ConnectionString` в инспекторе объектов. В открывшемся окне диалога `ConnectionString` (Рис. 5.2) выберите переключатель `Use Connection String` и щелкните на кнопке `Build`.

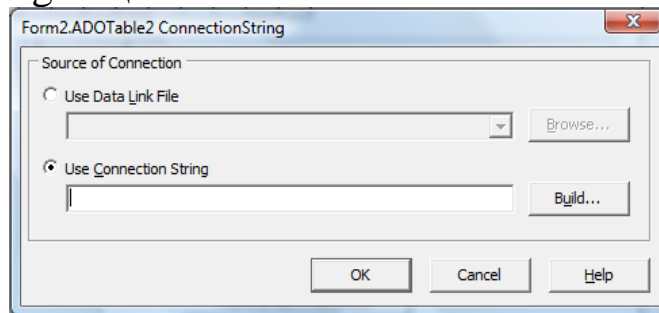


Рис. 5.2. Подключение к файлу БД

5. На вкладке `Provider` открывшегося окна диалога `Data Link Properties` задайте вид соединения с базой данных – `Microsoft Jet 4.0 OLE DB Provider` (Рис. 5.3).

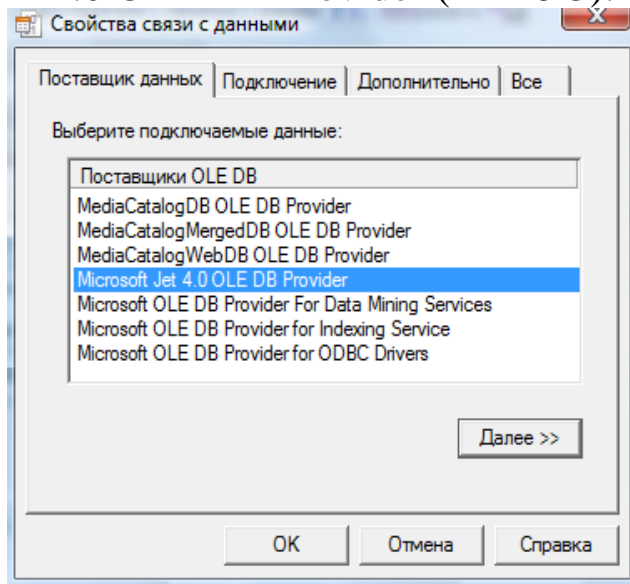


Рис. 5.3

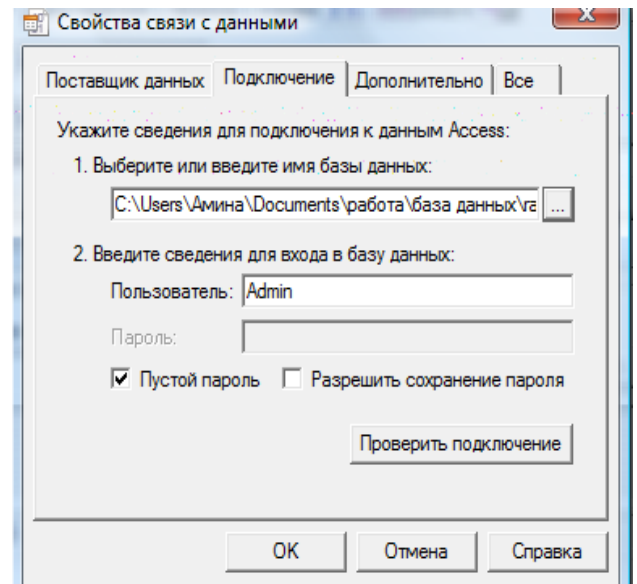


Рис. 5.4

6. Укажите имя подключаемой базы данных в поле ввода *Выберите или введите имя базы данных* (Select or enter a database name) на вкладке *Подключение* (Connection) окна диалога *Свойства связи с данными* (Data Link Properties) (Рис. 5.4) и щелкните на кнопке `Ok` (предварительно можно щелкнуть на кнопке *Проверить подключение* (Test Connection), чтобы убедиться, что база данных подключена корректно).
7. Теперь, после подключения базы данных, необходимо указать используемую таблицу. Выделите на форме компонент `TADOTable` и затем в поле ввода свойства `TableName` в

инспекторе объектов укажите имя используемой таблицы – Физические лица

8. Следующий этап – настройка источника данных TDataSource. Чтобы связать источник данных с набором данных, используйте свойство DataSet. С помощью инспектора объектов укажите в свойстве DataSet имя объекта TADOTable (по умолчанию – ADOTable1).
9. Разместите на форме необходимые элементы управления и выполните их настройку. Примерный вариант размещения компонентов показан на Рис. 5.5.

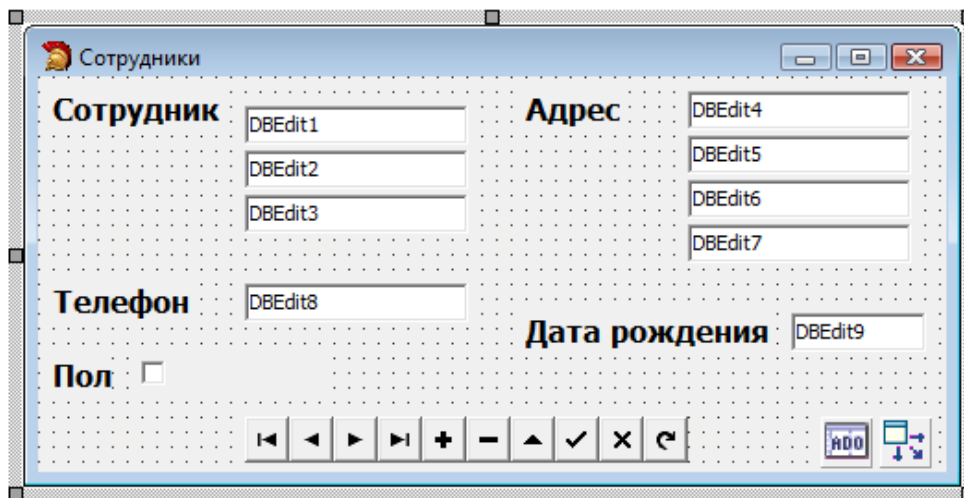


Рис. 5.5. Макет формы

10. Для настройки элементов визуализации полей базы данных (десять полей ввода TDBEdit и флажок TDBCheckBox) и элемента навигации по набору данных (TDBNavigator) отредактируйте в инспекторе объектов их свойство DataSource. Затем укажите имя источника данных (по умолчанию – DataSource1) и имя поля набора данных, с которым связывается элемент отображения и редактирования данных.
11. Осталось реализовать процедуры открытия и закрытия набора данных. Набор данных должен открываться при запуске приложения и закрываться при его завершении. Для открытия набора данных используется метод Open класса TADOTable, для закрытия – метод Close того же класса. Вызовите метод Open в обработчике события OnShow главной формы, а метод Close – в обработчике OnClose.

Текст модуля разработанной формы приведен в листинге 1.

## ЛИСТИНГ 1

```
unit Unit2;
  interface
  uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls, Mask, DBCtrls, DB, ADODB, ExtCtrls;
  type
    TForm2 = class(TForm)
      ADOTable1: TADOTable;
      DataSource1: TDataSource;
      Label1: TLabel;
      Label2: TLabel;
      DBEdit1: TDBEdit;
      DBEdit2: TDBEdit;
      DBEdit3: TDBEdit;
      DBEdit4: TDBEdit;
      DBEdit5: TDBEdit;
      DBEdit6: TDBEdit;
      DBEdit7: TDBEdit;
      Label3: TLabel;
      Label4: TLabel;
      Label5: TLabel;
      DBCheckBox1: TDBCheckBox;
      DBEdit8: TDBEdit;
      DBEdit9: TDBEdit;
      DBNavigator1: TDBNavigator;
      Label6: TLabel;
      DBEdit10: TDBEdit;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormShow(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form2: TForm2;
implementation
```

```
{$R *.dfm}
```

```
procedure TForm2.FormShow(Sender: TObject);
```

```
begin
```

```
ADOTable1.Open;
```

```
end;
```

```
procedure TForm2.FormClose(Sender: TObject; var Action: TCloseAction);
```

```
begin
```

```
ADOTable1.Close;
```

```
end;
```

```
end.
```

12. Откомпилируйте и запустите программу. Внешний вид окна программы приведен на Рис. 5.6.

<b>Сотрудник</b>	Сидоров	<b>Адрес</b>	369200
	Сидор		Россия
	Сидорович		Карачаевск
			ул. Мира 3, кв. 6
<b>Телефон</b>	88787923333	<b>Дата рождения</b>	07.09.1978
<b>Пол</b> <input checked="" type="checkbox"/>		<b>Код</b>	1

Рис. 5.6. Тестирование приложения

### Пример 2: Анкета сотрудника (табличная форма)

Часто наиболее естественным способом представления информации при просмотре и редактировании является таблица. Для представления данных в табличной форме используется компонент **TDBGrid**.

Рассмотрим пример создания табличной формы (см. Рис. 5.7). Добавим на предыдущую форму компонент **TDBGrid** и установим свойство *DataSource* – *DataSource1*.

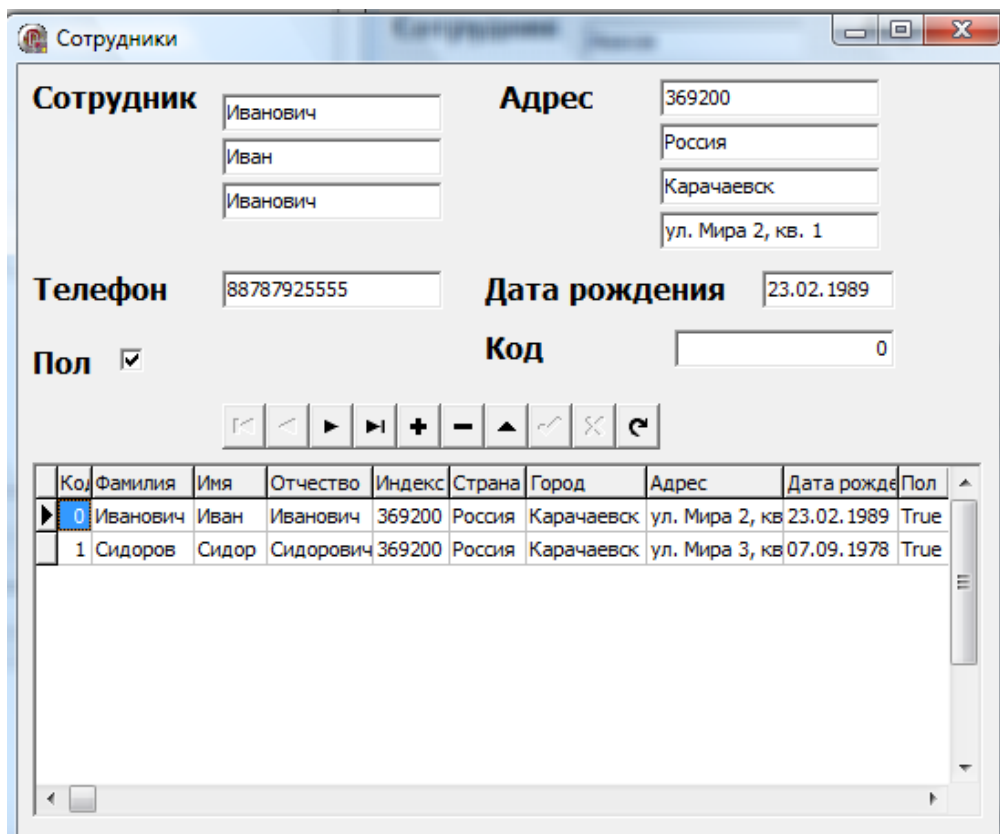


Рис. 5.7.

### Пример 3: Справочник компаний и их сотрудников (связанные таблицы)

Рассмотрим работу со связанными таблицами в Delphi на примере справочников компаний и их сотрудников. В базе данных содержится две таблицы: Компания и Сотрудник, связанные отношением 1:М (см. Рис. 5.8). Создадим приложение позволяющие редактировать перечень компании и их сотрудников.

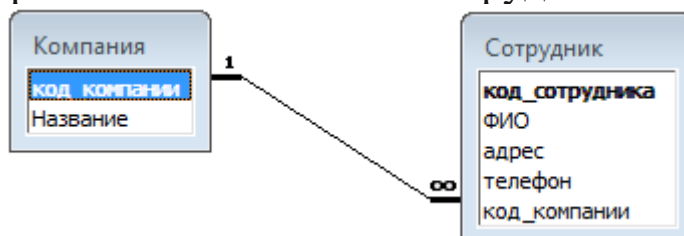


Рис. 5.8. Схема данных

1. Создайте папку и дайте ей имя.
2. По Рис. 5.8 создайте базу данных и сохраните в Вашу папку.
3. Создайте приложение Delphi (для этого запустите Delphi, выполнить команды File → New → VCL Forms Application).
4. Сохраните приложение в созданную папку (File → Save project as ... в появившемся окне выбрать свою папку и ввести имя приложения).
5. Разместите на форме следующие компоненты (см. Рис. 5.9): TADOTable (2 шт.), TDBGrid (2 шт.), TDataSource (2 шт.), TDBNavigator (2 шт.).

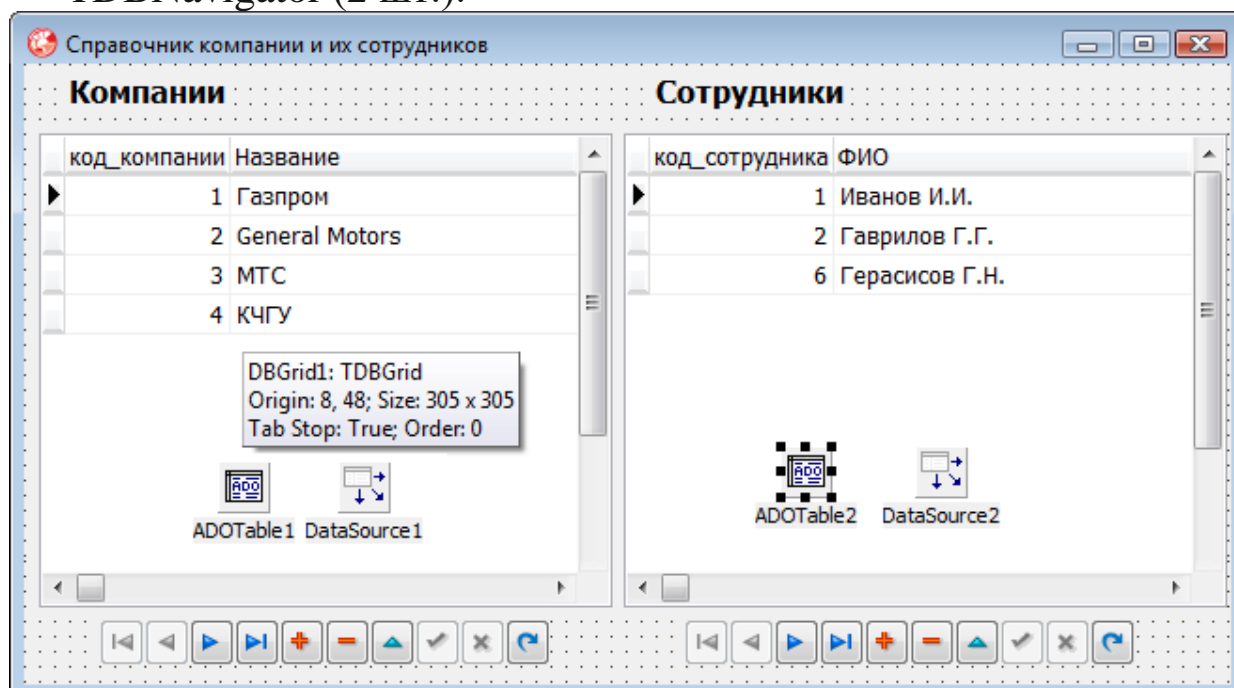


Рис. 5.9. Форма приложения



6. Свяжите TADOTable с базой данных, заполняя свойство `ConnectionString`. Далее выделите DBGrid1 и свяжите его с `DataSource1`, в свойстве `DataSource`. У компонента `DataSource1` в свойстве `DataSet` укажите `ADOTable1`. Это будет главная таблица, а точнее список компаний. Далее главную таблицу необходимо активировать. Выделите `ADOTable1`, в свойстве `TableName` выберите «Компания» и свойство `Active` установите в «True».
7. Теперь настройте подчиненную таблицу (Сотрудники). Выделите компонент DBGrid2 и в свойстве `DataSource` свяжите его с компонентом `DataSource2`, а компонент `DataSource2`, в свойстве `DataSet`, свяжите с компонентом `AdoTable2`. Выделите компонент `AdoTable2` и в свойстве `TableName` выберите `Сотрудник`, в свойстве `MasterSource` выберите `DataSource1` (т.е. необходимо выбрать `TDataSource` главной таблицы, то есть главную таблицу). Далее, в свойстве `MasterFields`, компонента `AdoTable2` (подчиненной таблицы), укажите связующие поля (см. Рис. 5.10).

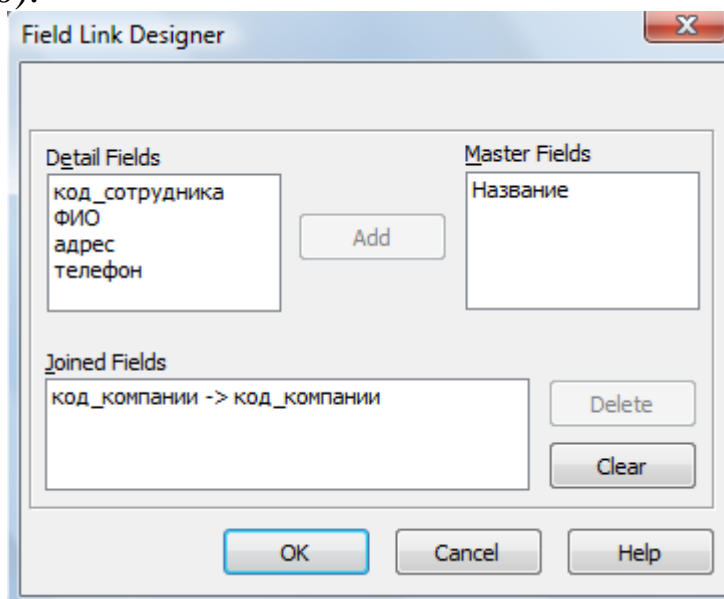


Рис. 5.10. Определение связующих полей

Также активируйте данную таблицу - свойство `Active` установите в «True».

8. Запустите проект, выделите запись из главной таблицы, тогда автоматически в DBGrid2 для данной записи из главной таблицы выводятся записи из подчиненной таблицы. Добавлять записи необходимо также, выделите запись из главной таблицы и только после этого, добавьте запись в подчиненную таблицу.

**9. Автоматическая установка пути файла БД.** Для этого сначала свяжите компонент ADOTable с БД, заполняя свойство `ConnectionString`, скопируйте эту строку в буфер памяти и заполните его теперь программно в обработчике события `FormCreate`. Ту часть строки, которая содержит путь к файлу, нужно заменить вызовом процедуры `ExtractFileDir(Application.ExeName)`, которая будет определять путь, из которого запущено приложения.

```
procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
```

```
    adotable1.Active:=false;
```

```
    adotable1.ConnectionString:='Provider=Microsoft.Jet.OLEDB.4.0;Data  
Source='+ExtractFileDir(Application.Exename)+'\Сотрудники.mdb;Persist  
Security Info=False';
```

```
    adotable1.Active:=true;
```

```
    adotable2.Active:=false;
```

```
    adotable2.ConnectionString:='Provider=Microsoft.Jet.OLEDB.4.0;Data  
Source='+ExtractFileDir(Application.Exename)+'\Сотрудники.mdb;Persist  
Security Info=False';
```

```
    adotable2.Active:=true;
```

```
end;
```

Существует еще один метод связывания таблиц, при котором программисту необходимо самому отслеживать для какой записи из основной таблицы добавляется запись подчиненной таблицы, самому записывать идентификатор основной таблицы в подчиненную, а затем уже, по выделенной записи (получать ее идентификатор), делать выборку записей из подчиненной таблицы.

### **Задания для самостоятельного выполнения**

**1.** Реализуйте и проверьте приложения, приведенные в примерах 1, 2 и 3.

**2.** Разработайте приложения доступа к базе данных, разработанной в теме 6. Макет окна и листинг разработанного приложения отразите в отчете

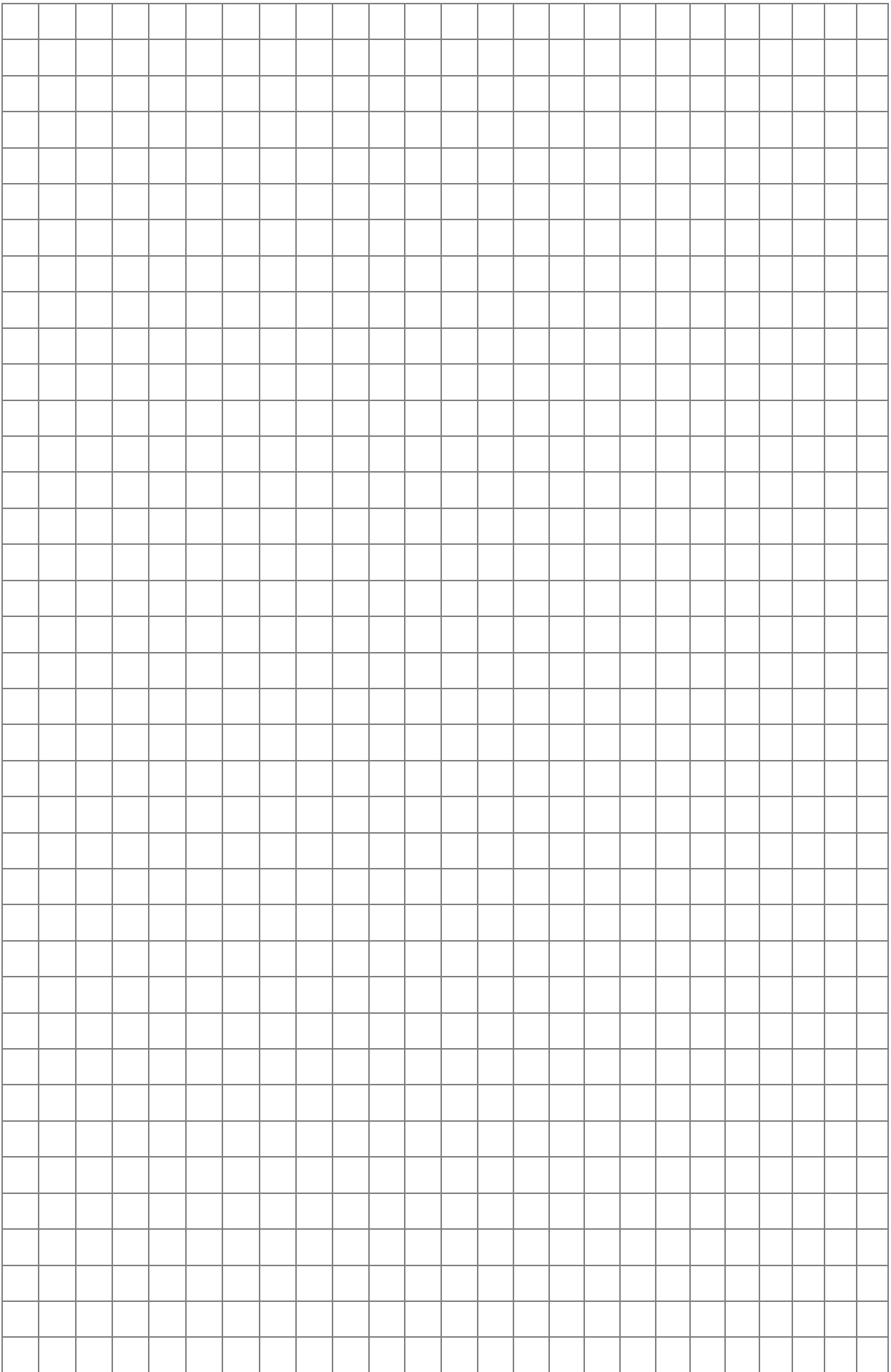
**3.** Ответьте на следующие вопросы:

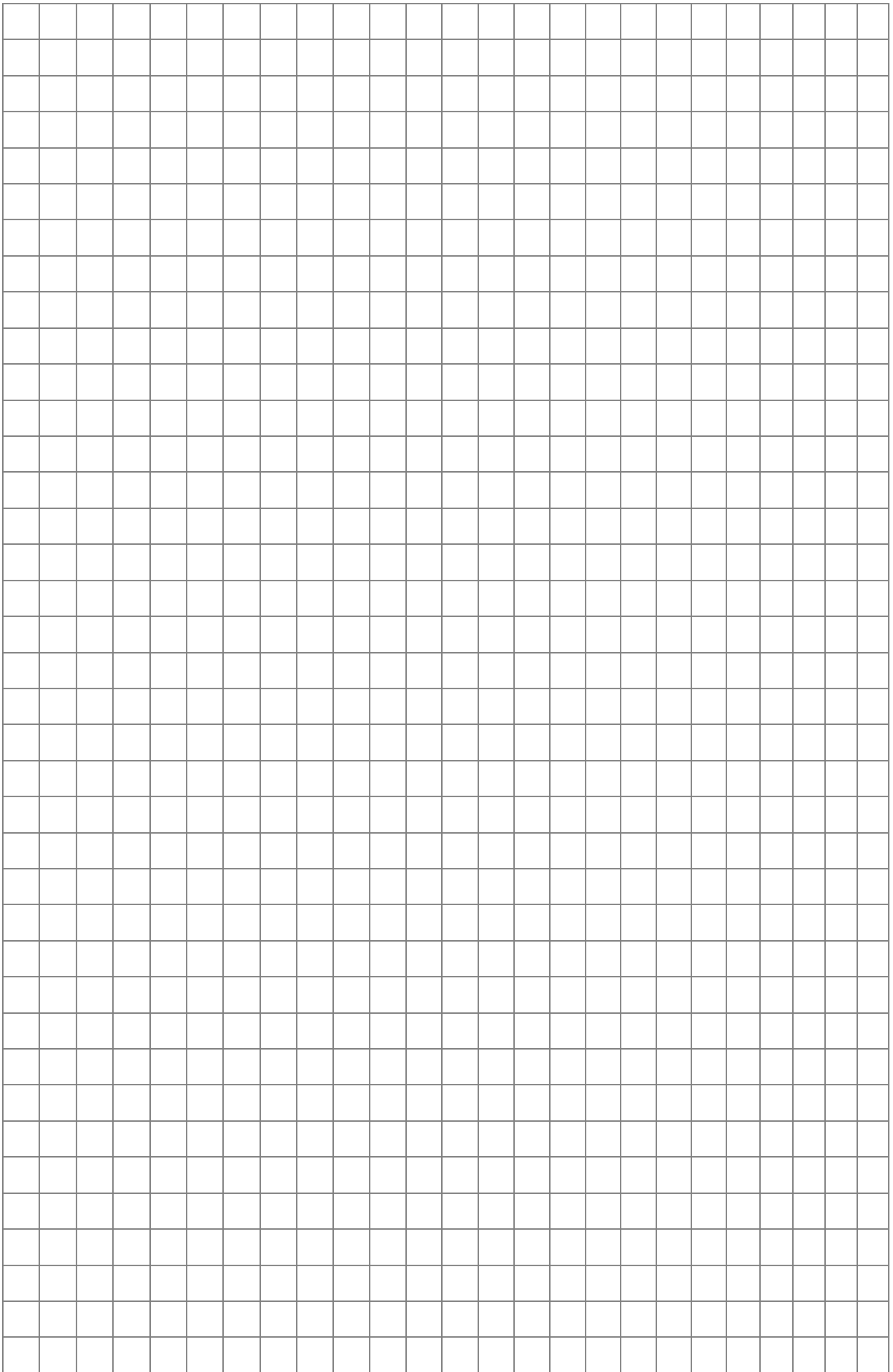
**a)** Что такое технология ADO?

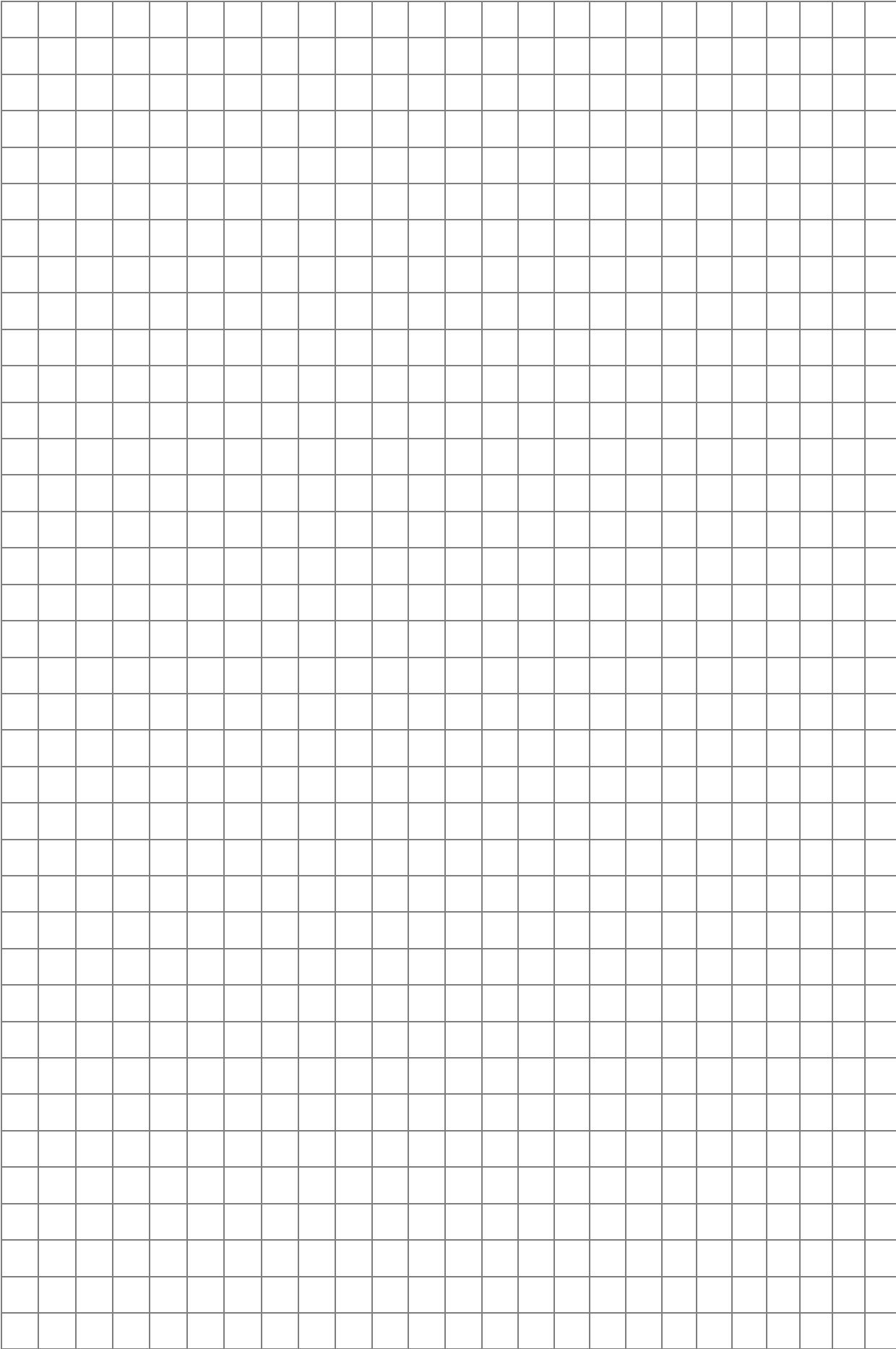
**b)** Какие средства позволяют организовать доступ к базам данных в Delphi?

**c)** Какие компоненты Вы использовали при выполнении задания?









## Тема 6. Выборка данных

**Цель:** сформировать навыки разработки приложений, осуществляющих выборку из базы данных. По завершению выполнения заданий данной темы студент должен уметь создавать приложения, реализующих некоторые функции на основе запроса к базе данных.

### Теоретические сведения

#### 1. Использование SQL для выборки данных из таблицы

Одним из наиболее эффективных и универсальных способов выборки данных из таблиц базы данных является использование запросов языка SQL.

SQL стандартизованы (стандарт ANSI SQL 92). Однако следует учитывать, что производители СУБД обычно предлагают свои реализации SQL, которые могут включать расширения команд стандарта и даже отклонения от него. Тем не менее, большинство команд SQL имеют одинаковый или очень похожий синтаксис в различных реализациях.

В Delphi для работы с таблицами локальных баз данных с использованием BDE и ADO применяется собственная реализация языка SQL, называемая локальным SQL (Local SQL). Данная реализация является подмножеством языка SQL 92 [6, 7, 14]. Для работы с базами данным через SQL-запросы в VCL Delphi используются, как правило, два вида компонентов:

- **TQuery** - взаимодействует с БД посредством драйверов BDE;
- **TADOQuery** - работает с БД с использованием технологии ADO.

Компоненты TQuery и TADOQuery получают данные как результат выполнения SQL-запроса. Некоторые свойства этих классов приведены в таблице 6.1.

Таблица 6.1.

Свойство	Тип	Описание
DataSource	TDataSource	Задаёт источник данных, связанный с набором данных, поля которого используются в качестве параметров SQL-запроса
SQL	TStrings	Содержит текст SQL-запроса
Text	PChar	Указывает на текст SQL-запроса, передаваемый BDE
RecordCount	Integer	Количество записей, соответствующих запросу

Также имеется ряд методов, которые довольно часто используются при работе с базами данных через SQL-запросы:

- *procedure ExecSQL* – выполняет SQL-запрос, заданный в свойстве SQL. Обычно используется в тех случаях, когда в результате выполнения запроса не возвращаются данные (например, при выполнении команд INSERT, UPDATE, DELETE и CREATE TABLE);
- *function ParamByName (const Value: String): TParam* – обеспечивает доступ к заданным параметрам по имени;
- *procedure Prepare* – посылает запрос для подготовки к выполнению. Вызов данного метода перед выполнением запроса повышает скорость выполнения;
- *procedure UnPrepare* – освобождает ресурсы, занятые при подготовке запроса к выполнению.

### **Пример: Поисковый запрос**

1. Создайте новый проект: File → New → VCL Forms Application.
2. Сохраните проект: File → Save project as ...
3. Разместите на форму компонент ADOQuery1 с вкладки dbGo. Свяжите этот компонент с базой данных с помощью свойства *ConnectionString*. Запрос запишите с помощью свойства *SQL* (см. Рис. 6.1).

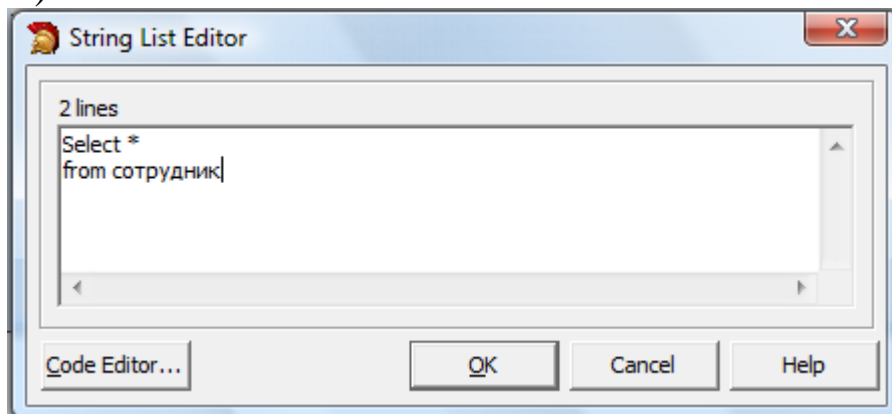


Рис. 6.1. Редактор SQL свойства компонента ADOQuery

В свойство *Active* установите значение true.

4. Разместите на форму компонент DataSource1 с вкладки Data Access. В свойстве Data Set компонента DataSource1 выберите ADOQuery1.
  5. Далее разместим на форме компонент DBGrid1 с вкладки Data Controls. В свойстве Data Source указываем DataSource1.
- Результат этих действий показан на Рис. 6.2.



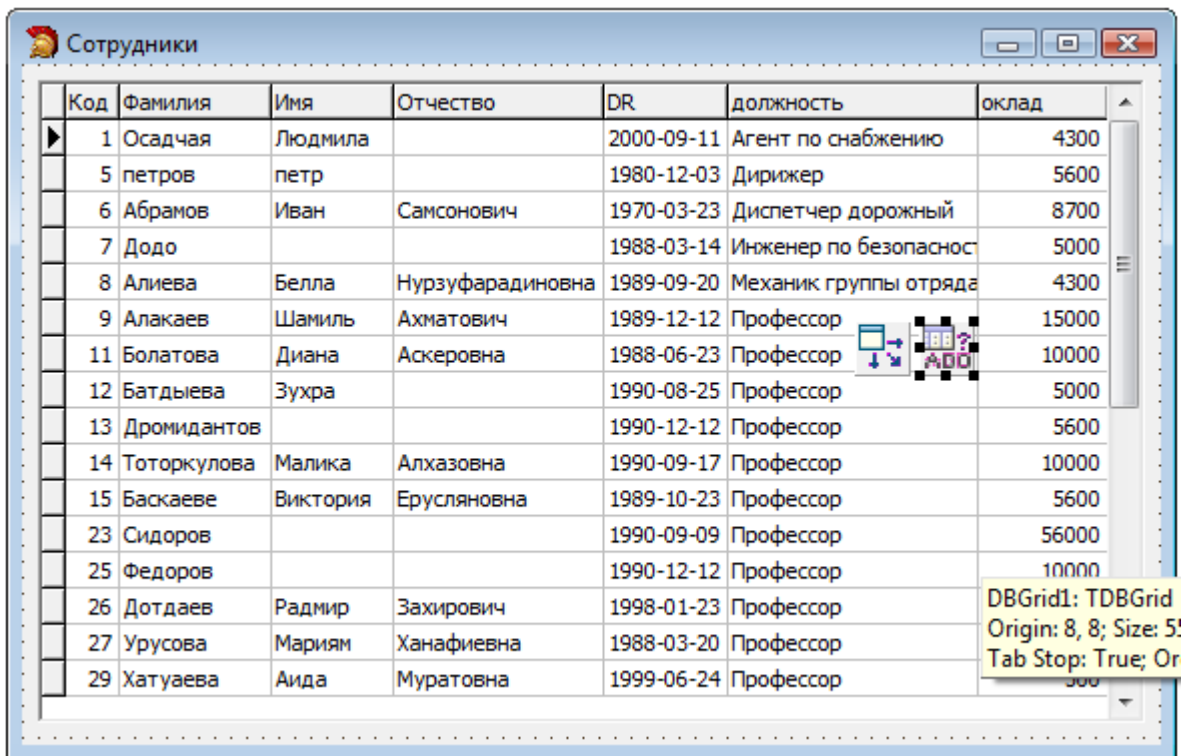


Рис. 6.2. Макет окна приложения

6. Добавьте возможность поиска сотрудника по фамилии, имени, отчеству (см. Рис. 6.3).

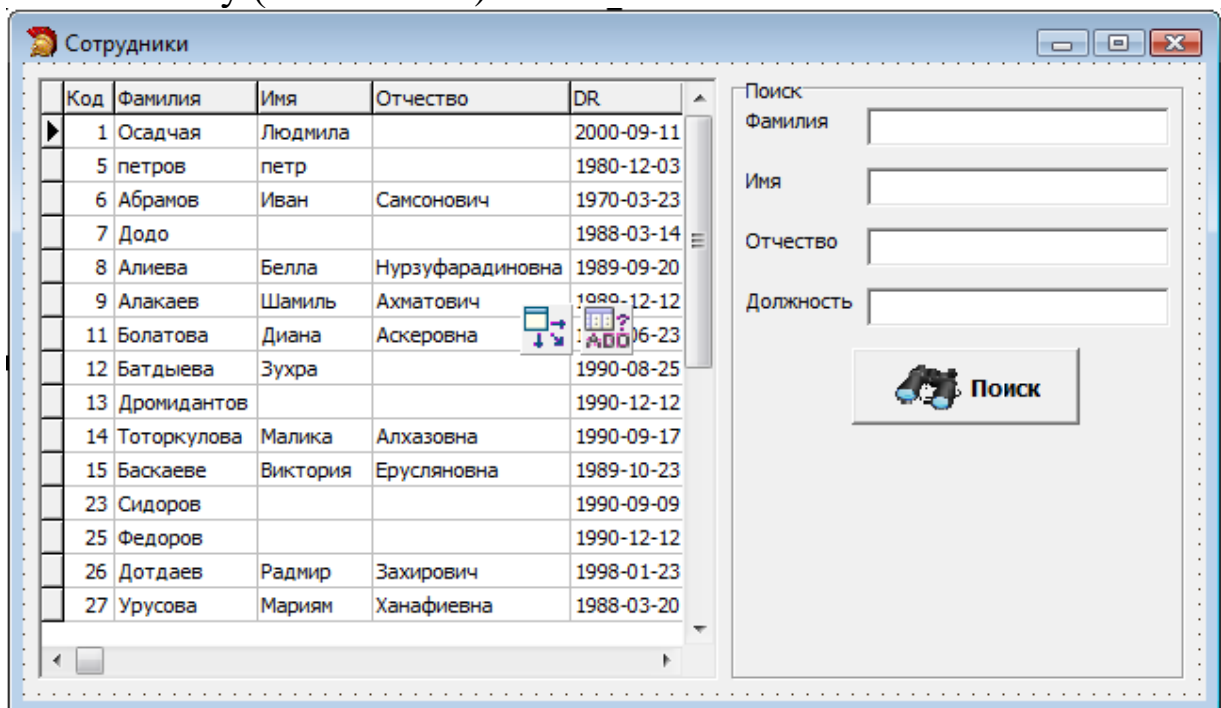


Рис. 6.3. Макет окна приложения

Обработчик события OnClick (щелчок) кнопки **Поиск**:  
**procedure TForm6.BitBtn1Click(Sender: TObject);**

var s,s1,s2,s3, s4:string;

**begin**

if edit1.Text<>' then s1:=' and (Фамилия ='+edit1.text+' ';

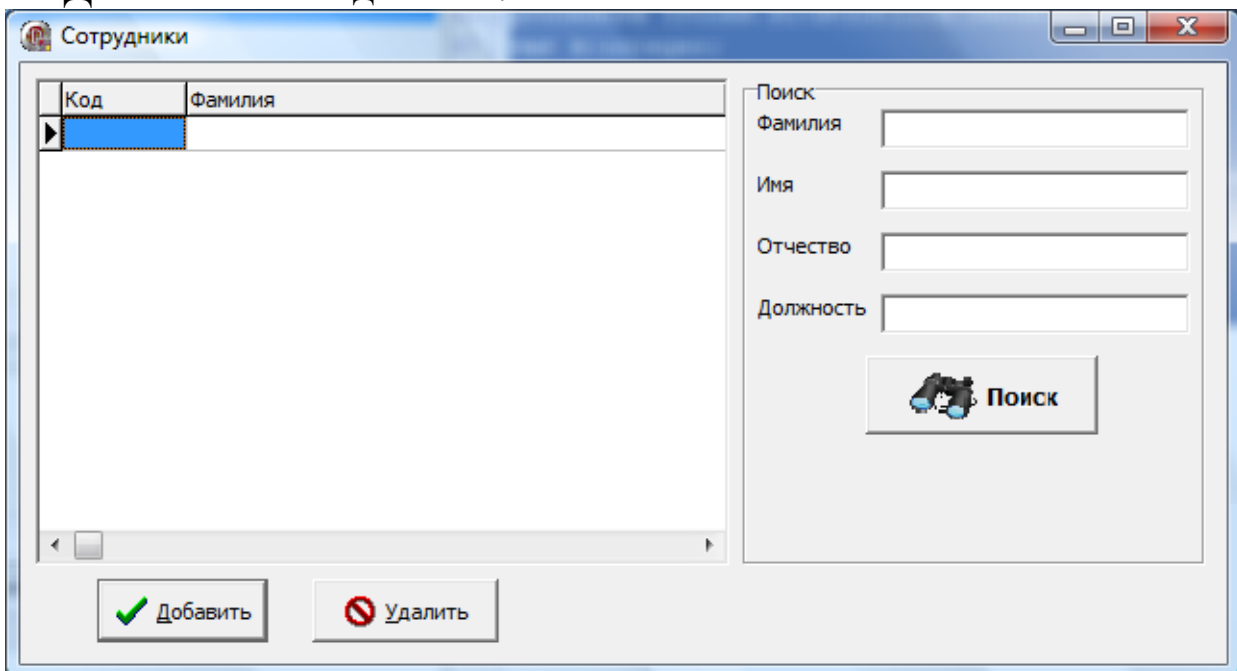
```

if edit2.Text<>" then s2:='and (Имя = '"+edit2.text+"' )';
if edit3.Text<>" then s3:='and (Отчество = '"+edit3.text+"' )';
if edit4.Text<>" then s4:='and (Должность = '"+edit4.text+"' )';
s:=s1+s2+s3+s4;
ADOQuery1.Close;
ADOQuery1.SQL.Clear;
ADOQuery1.SQL.Add('SELECT * ');
ADOQuery1.SQL.Add('FROM сотрудник');
ADOQuery1.SQL.Add('WHERE (true)'+s);
ADOQuery1.Open;

```

**end;**

7. Добавьте средства вставки и удаления записей - кнопки «Добавить» и «Удалить».



*Рис. 6.4. Макет главного окна приложения*

Новую запись будем добавлять с помощью отдельного окна. Чтобы добавить окно в приложение щелкните меню File → New → Form. Разместите компоненты ввода данных на новую форму так, как показано на Рис. 6.5.

Чтобы это окно появлялось при щелчке по кнопке «Добавить», в обработчике события нужно ввести команду отображения окна:

```

procedure TForm6.BitBtn2Click(Sender: TObject);
  begin
    form1.show;
  end;

```

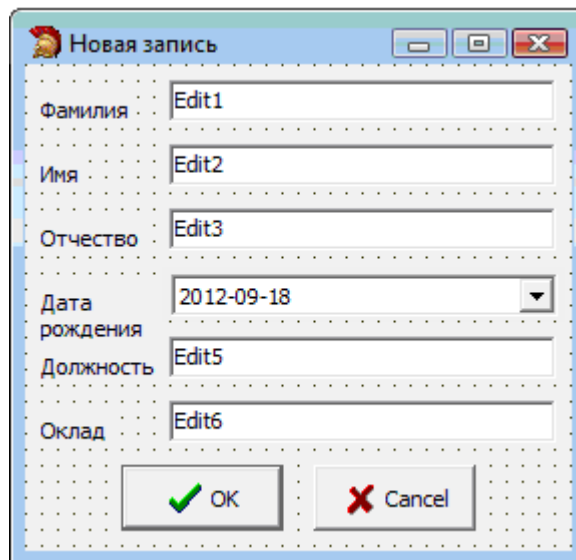


Рис. 6.5. Форма добавления новой записи в БД

Кнопка подтверждения добавления **Ok:**

```
procedure TForm1.BitBtn1Click(Sender: TObject);
```

```
begin
```

```
  // проверить заполнены ли все поля
```

```
  if (edit1.Text<>") and (edit2.Text<>") and (edit3.Text<>") and  
  (edit4.Text<>") and (edit5.Text<>") and (edit6.Text<>") then
```

```
    begin
```

```
      FORM6.adoquery1.active:=false;
```

```
      form6.ADOQuery1.SQL.Text:='INSERT INTO сотрудник  
(Фамилия, Имя, Отчество,DR, должность, оклад) VALUES  
( '"+edit1.text+"', '"+edit2.text+"', '"+edit3.text+"',  
' +datetostr(datetempicker1.Date)+'', '"+edit5.text+"', '+  
edit6.text+')';
```

```
      form6.ADOQuery1.ExecSQL;
```

```
      form6.ADOQuery1.close;
```

```
      form6.ADOQuery1.SQL.Text:='select * from сотрудник';
```

```
      form6.ADOQuery1.Open;
```

```
      form1.Close; // закрыть форму ввода новой записи
```

```
    end
```

```
  else
```

```
    showmessage('Заполнены не все поля! Запись не может быть  
добавлена в базу данных!');
```

```
end;
```

Кнопка отмены **Cancel:**

```
procedure TForm1.BitBtn2Click(Sender: TObject);
```

```
begin
```

```

        form1.Close;
end;
8. Удаление записи.
procedure TForm6.BitBtn3Click(Sender: TObject);
    var k:integer;
begin
k:=adoquery1.FieldName('код').AsInteger;
adoquery1.Active:=false;
adoquery1.SQL.Text:='Delete From сотрудник where Код='+inttostr(k);
adoquery1.ExecSQL;
ADOQuery1.close;
ADOQuery1.SQL.Text:='select * from сотрудник';
ADOQuery1.Open;
end;

```

## 2. Язык запросов SQL

Язык запросов, являющийся одной из категорий языка SQL, состоит всего из одной команды SELECT. Эта команда вместе с множеством опций и предложений используется для формирования запросов к базе данных.

Оператор SELECT не используется автономно, вместе с ним обязательно должны задаваться уточняющие предложения. Предложения, используемые совместно с командой SELECT, могут быть *обязательными* и *дополнительными*. *Обязательным* является только одно предложение - FROM, без которого оператор SELECT не может использоваться.

### 2.1. Простейшая форма оператора SELECT

Оператор SELECT вместе с предложением FROM используется для получения информации из базы данных. Синтаксис простейшей формы оператора SELECT приведен ниже [6, 7, 14]:

```

SELECT {* | ALL | DISTINCT field1, field2, ..., fieldN}
FROM table1 {, table2, ..., tableN}

```

Здесь за ключевым словом SELECT следует список полей, которые возвращаются в результате выполнения запроса:

- имена полей в списке разделяются через запятую;
- для выборки всех полей таблицы (таблиц) используется символ подстановки «\*»;

- опция ALL (задана по умолчанию) означает, что результат выборки будет содержать все записи, включая дублирующие друг друга;

- при использовании опции DISTINCT результат запроса не будет содержать дублирующихся строк.

Совместно с командой SELECT всегда используется предложение FROM, с помощью которого указывается имя таблицы (таблиц), из которой производится выборка. Если в предложении FROM указывается несколько таблиц, то их имена разделяются запятыми.

Чтобы выбрать не все поля, а лишь некоторые, необходимо после слова SELECT указать имена полей, которые будут включены в результат выборки. В качестве примера ниже приведен запрос, возвращающий значения только трех полей: «Код товара», «Наименование» и «Цена»:

```
SELECT [Код товара], Наименование, Цена
FROM Товары
```

Обратите внимание, что при указании в списке оператора SELECT имен полей, содержащих пробел, их необходимо заключать в квадратные скобки. Это правило необходимо выполнять и для имен таблиц с пробелами, указываемых, например, в предложении FROM.

## 2.2. Задание условий при выборке данных

Для ограничения отбираемой из базы данных информации оператор SELECT позволяет использовать условие, которое задается с помощью предложения WHERE. В случае реализации условной выборки оператор SELECT имеет следующий вид [6, 7, 14]:

```
SELECT {* | ALL | DISTINCT field1, field2, ..., fieldN}
FROM table1 {, table2, ..., tableN}
WHERE условие
```

Специальные операторы языка SQL, применяемые для задания условия, можно разделить на следующие группы:

- операторы сравнения;
- логические операторы;
- операторы объединения;
- операторы отрицания.

Результатом выполнения каждого из этих операторов является логическое значение (true или false). Если для некоторой записи оператор возвращает значение true, то запись включается в результат выборки, если false – не включается.

**Операторы сравнения** используются в запросах SQL для наложения ограничений на информацию, возвращаемую в результате выполнения запроса. Это типичные операторы, существующие во всех алгоритмических языках:

- оператор равенства «=» используется для отбора записей, в которых значение определенного поля точно соответствует заданному;
- оператор неравенства «<>» возвращает значение true, если значение поля не совпадает с заданным значением;
- операторы «меньше» и «больше» (соответственно, «<» и «>») позволяют отбирать записи, в которых значение определенного поля меньше или больше некоторой заданной величины;
- операторы «меньше или равно» и «больше или равно» (соответственно, «<=» и «>=») представляют собой объединение операторов «меньше» и «равно», «больше» и «равно». В отличие от операторов «<» и «>» операторы «<=» и «>=» возвращают значение true, если значение поля совпадает с заданным значением.

В качестве примера рассмотрим запрос, выбирающий из таблицы «Товары» только те записи, категория товаров в которых равна 2:

```
SELECT *  
FROM Товары  
WHERE Категория=2
```

**Логические операторы.** К логическим относятся операторы, в которых для задания ограничений на отбор данных используются специальные ключевые слова. В SQL определены следующие логические операторы: Is null, BETWEEN...AND, IN, LIKE, EXISTS, UNIQUE, ALL, ANY.

*Оператор IS NULL* предназначен для сравнения текущего значения поля со значением NULL. Он используется для отбора записей, в некоторое поле которых не занесено никакое значение.

*Оператор BETWEEN.AND* применяется для отбора записей, в которых значения поля находятся внутри заданного диапазона. Границы диапазона включаются в условие отбора.

Чтобы продемонстрировать работу этого оператора, вернемся к таблице «Товары» и выберем в ней товары, цена которых находится в диапазоне от 200 до 2000. Для этого сформируем следующий запрос:

```
SELECT *  
FROM Товары  
WHERE Цена BETWEEN 200 AND 2000
```

*Оператор IN* используется для выборки записей, в которых значение некоторого поля соответствует хотя бы одному из значений заданного списка.

Например, из таблицы «Клиенты» список клиентов, которые живут в Беларуси, Украине или Казахстане:

```
SELECT Фамилия, Имя, Отчество, Страна  
FROM Клиенты  
WHERE Страна IN ('Беларусь', 'Украина', 'Казахстан')
```

*Оператор LIKE* применяется для сравнения значения поля со значением, заданным при помощи шаблонов. Для задания шаблонов используются два символа:

- знак процента «%» - заменяет последовательность символов любой (в том числе и нулевой) длины;
- символ подчеркивания «\_» - заменяет любой единичный символ.

Найдем в таблице «Клиенты» записи, в которых фамилия клиента начинается с буквы «М»:

```
SELECT Фамилия, Имя, Отчество, Телефон  
FROM Клиенты  
WHERE Фамилия LIKE 'М%'
```

*Оператор EXISTS* используется для отбора записей, соответствующих заданному критерию.

*Оператор UNIQUE* используется для проверки записи таблицы на уникальность. По своему действию он аналогичен оператору EXISTS. Единственное отличие заключается в том, что подзапрос, задаваемый после ключевого слова UNIQUE, не должен возвращать более одной записи.

*Оператор ALL* используется для сравнения исходного значения со всеми другими значениями, входящими в некоторый набор данных.

Например, для того чтобы выбрать из таблицы «Товары» те товары, которые имеют цену большую, чем цена всех товаров, проданных в количестве более 10, используется следующий запрос:

```
SELECT *  
FROM Товары  
WHERE Цена>ALL (SELECT Продажи.Цена  
FROM Продажи  
WHERE Продажи.Продано>10)
```

*Оператор ANY* применяется для сравнения заданного значения с каждым из значений некоторого набора данных. Если в предыдущем примере заменить оператор ALL на ANY, то будет возвращен список товаров, цена которых больше, чем хотя бы у одного из товаров, проданных в количестве больше 10.

***Операторы объединения.*** Часто при написании запроса на выборку данных требуется задать сложное условие, для которого недостаточно использовать только один оператор. В этом случае используется объединение нескольких условий с помощью специальных операторов.

В SQL определены два таких оператора:

- Оператор *AND* используется в тех случаях, когда необходимо отобрать записи, соответствующие нескольким условиям. Причем для каждой записи, включаемой в результат выборки, должны выполняться все заданные ограничения. Оператор AND объединяет несколько условий путем выполнения операции логического умножения результатов всех заданных ограничений. Результат true, соответственно, будет получен только в том случае, если все объединяемые условия принимают значение true.
- Оператор *OR* выполняет операцию логического сложения результатов всех заданных условий. При использовании данного оператора запись включается в результирующую выборку в случае выполнения хотя бы одного из заданных ограничений [6, 7, 14].



При использовании операторов объединения каждое логическое выражение следует заключать в круглые скобки. Для примера произведем выборку данных о товарах, цена которых больше 50, но меньше 1000:

```
SELECT *  
FROM Товары  
WHERE (Цена>50) AND (Цена<1000)
```

Синтаксические правила использования оператора OR такие же, как и для оператора AND. Следующий запрос:

```
SELECT *  
FROM Товары  
WHERE (Цена<50) OR (Цена>1000)
```

возвратит список товаров, цена которых меньше 50 или больше 1000.

**Оператор отрицания.** Для каждого из рассматриваемых операторов может быть выполнена операция отрицания, меняющая результат выполнения оператора на противоположный. Для реализации этой используется оператор *NOT*. Ниже приведены примеры использования этого оператора с логическими операторами [6, 7, 14]:

```
IS NOT NULL  
NOT BETWEEN  
NOT IN  
NOT LIKE  
NOT EXISTS  
NOT UNIQUE
```

### 2.3. Упорядочение данных

Для упорядочения данных в выборке, полученной в результате выполнения запроса, используется предложение *ORDER BY*. Синтаксис оператора SELECT в этом случае будет следующим:

```
SELECT {* | ALL | DISTINCT field1, field2, ..., fieldN}  
FROM table1 {, table2, ..., tableN}  
WHERE условие  
ORDER BY field {ASC | DESC}
```

После ключевых слов **ORDER BY** указывается имя поля (полей), по которому производится сортировка, а затем указывается режим сортировки:

- *ASC* – режим, используемый по умолчанию. При этом информация располагается в порядке возрастания значения указанного поля (для текстовых полей - в алфавитном порядке).
- *DESC* – используется для вывода информации в порядке убывания значений указанного поля (для текстовых полей – в порядке, обратном алфавитному).

Например, чтобы отсортировать список товаров по алфавиту, следует использовать следующий запрос:

```
SELECT Категория. Наименование. Цена  
FROM Товары  
ORDER BY Наименование
```

Вместо имени поля в предложении **ORDER BY** можно использовать целое число, определяющее порядковый номер поля в списке после ключевого слова **SELECT** (если производится выборка всех полей таблицы с помощью символа «\*»), то число указывает порядковый номер поля в таблице базы данных). Например, для вывода списка товаров в порядке убывания цены можно использовать следующий запрос:

```
SELECT Категория, Наименование, Цена  
FROM Товары  
ORDER BY 3 DESC
```

#### **2.4. Вставка данных INSERT INTO**

**INSERT INTO** - используется для добавления новых строк в таблицу. Синтаксис SQL **INSERT INTO**, используя перечисление значений, с указанием столбцов:

```
INSERT INTO ([, ... ]) VALUES (...)
```

Используя перечисление значений, без указания столбцов:

```
INSERT INTO VALUES (...)
```

В последнем случае, в таблицу можно вставить более одной записи. Если в таблице есть другие поля, требующие заполнения, но не указанные в операторе **INSERT**, то для них будет установлено значение по умолчанию, либо **NULL**, если значение по умолчанию не установлено.

## 2.4. Удаление данных DELETE FROM

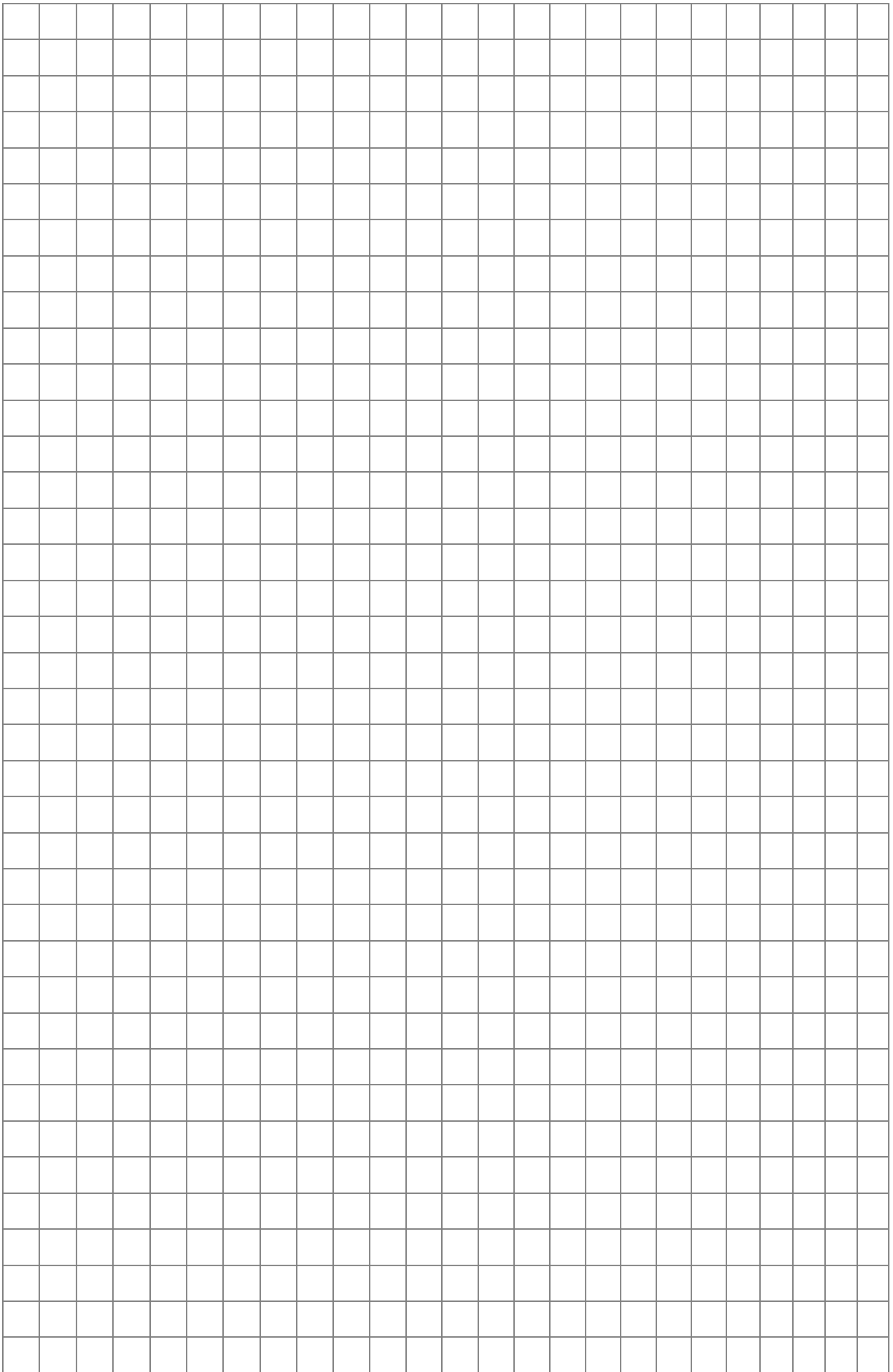
DELETE используется для удаления записей в таблице. Синтаксис  
DELETE FROM table\_name  
WHERE some\_column=some\_value

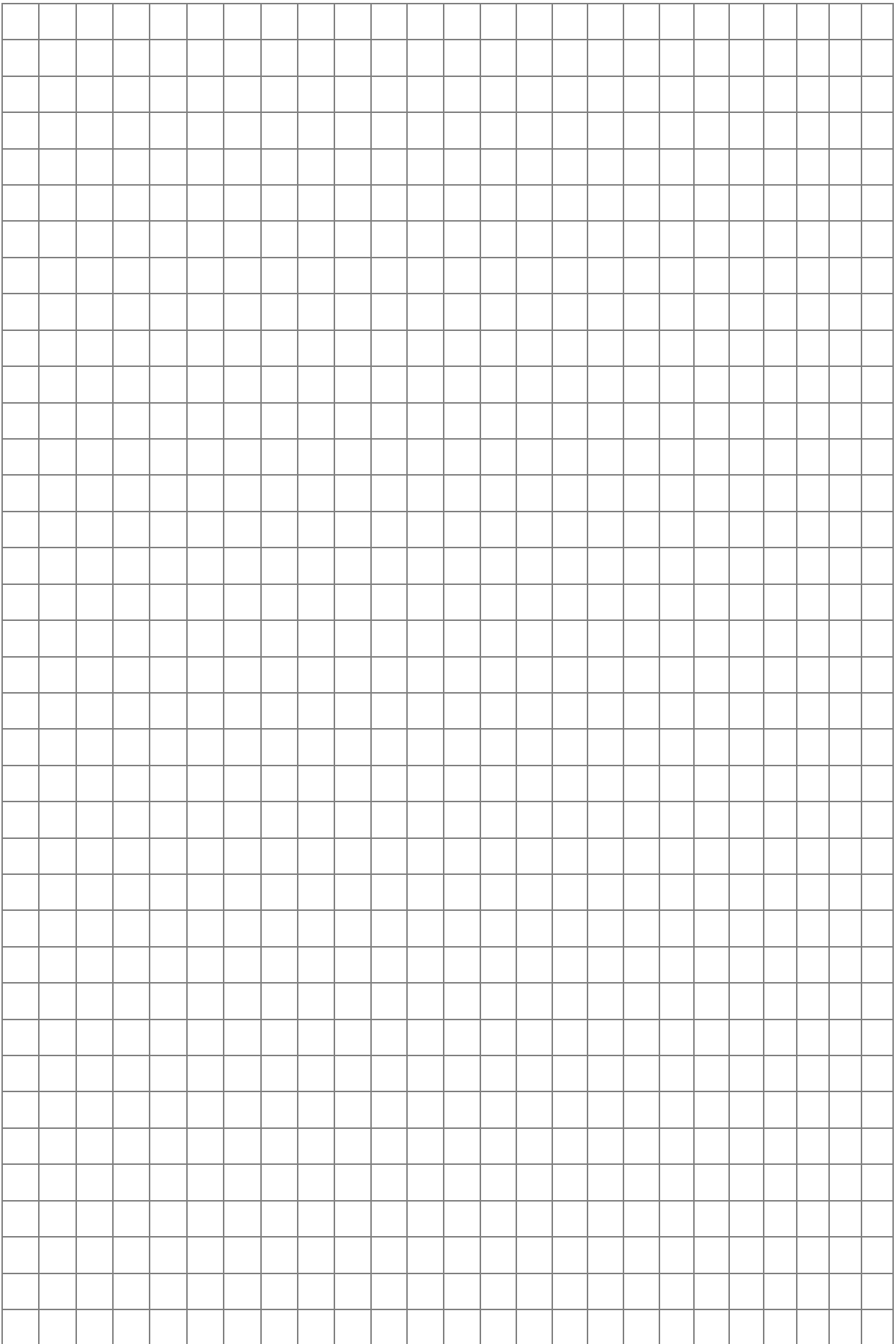
При составлении запроса на удаление, используйте условие WHERE, иначе все записи могут быть удалены [6, 7, 14].

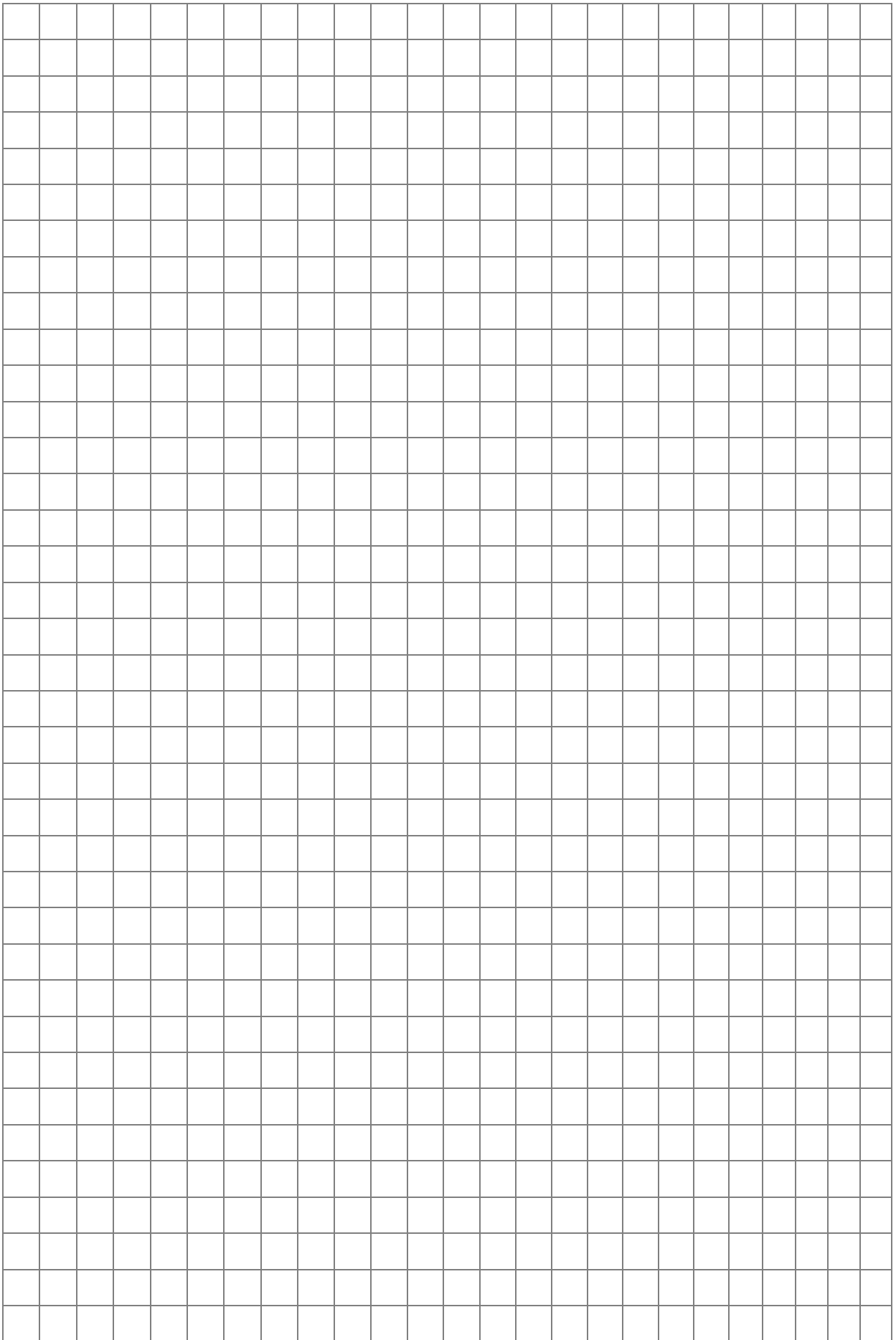
### Задания для самостоятельного выполнения

1. Выполните действия, приведенные в примере.
2. В разработанном приложении добавьте возможность поиска по размеру оклада и дате рождения.
3. Измените программу так, чтобы таблица сотрудников была отсортирована в алфавитном порядке по фамилии.
4. Добавьте в программу проверку существования вводимой должности в связанной таблице должностей при добавлении новой записи в базу данных.
5. Добавьте возможность редактирования списка должностей.
6. В приложении, разработанном в теме 7, добавьте функцию поиска данных по любому полю.
7. Ответьте на вопросы:
  - a) Перечислите свойства и методы компонентов TQuery, TADOQuery.
  - b) Запишите синтаксис оператора SELECT.
  - c) Запишите синтаксис оператора INSERT.
  - d) Запишите синтаксис оператора DELETE.









# Тема 7. Моделирование классов

**Цель:** сформировать навыки моделирования классов.

## Теоретические сведения

### 1. Концепции объекта и класса

Модель классов описывает статическую структуру системы: объекты и отношения между ними, атрибуты и операции для каждого класса объектов.

**Объект** – это концепция, абстракция или сущность, обладающая индивидуальностью и имеющая смысл в рамках приложения.

**Класс** описывает группу объектов с одинаковыми свойствами (атрибутами), одинаковым поведением (операциями), типами отношений и семантикой.

**Диаграммы классов** позволяют описать модель классов и их отношений при помощи графической системы обозначений. На диаграммах объектов изображаются отдельные объекты и отношения между ними. Диаграмма классов описывает бесконечное множество диаграмм объектов.

В языке UML используются следующие обозначения классов и объектов [12, 15, 16, 18].

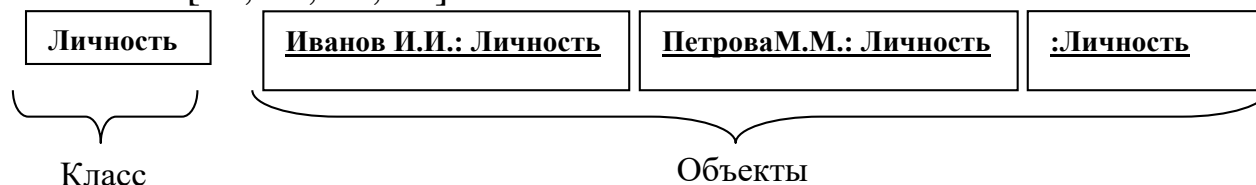


Рис. 7.1. Класс и объекты

**Значение** – это элемент данных.

**Атрибут** – это именованное свойство класса, описывающее значение, которое может иметь каждый объект класса. Атрибуты получают абстрагированием типичных знаний.

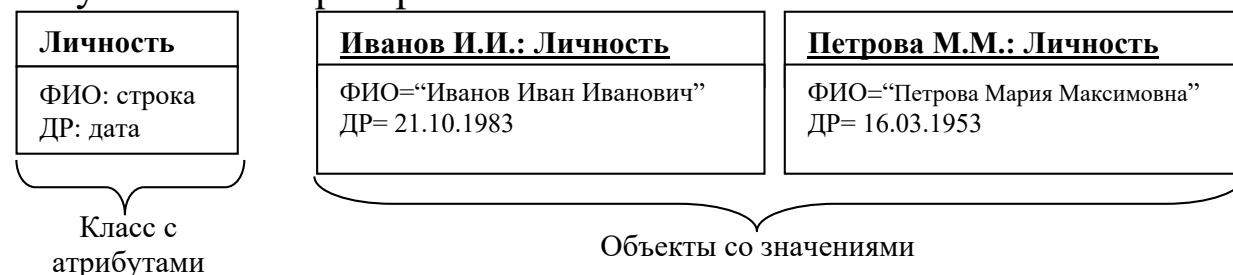


Рис. 7.2. Атрибуты обеспечивают детализацию классов



**Операция** – это функция или процедура, которая может быть применена к объектам класса.

**Методом** называется реализация операции в конкретном классе.

<b>Личность</b>	<b>Файл</b>	<b>ГеометрическаяФигура</b>
ФИО: строка ДР: дата Адрес: строка	имяФайла размерВБайтах датаПослОбновления	цвет положение
изменитьФИО изменитьАдрес	печатать переименовать открыть	изменитьЦвет переместить

Рис. 7.3. Классы с методами

<b>ИмяКласса</b>
имяАтрибута1:тип1=значениеПоУмолчанию1 имяАтрибута2:тип2=значениеПоУмолчанию2 ...
метод1 метод2 ...

Рис. 7.4. Полная система обозначения классов

## 2. Концепции связи и ассоциации

Связи и ассоциации позволяют устанавливать отношения между объектами. **Связь** – это физическое или концептуальное соединение между объектами. Например, *Иванов И.И. владеет акциями компании Газпром*. Связь – это экземпляр ассоциации.

**Ассоциация** – это описание группы связей, обладающих общей структурой и общей семантикой. Например, *Личность может владеть акциями какой-либо компании*. Связи, являющиеся экземплярами некоторой ассоциации, соединяют объекты тех классов, которые соединены между собой этой ассоциацией. Ассоциация описывает множество потенциальных связей точно так же, как класс описывает множество возможных объектов [12, 15, 16, 18].

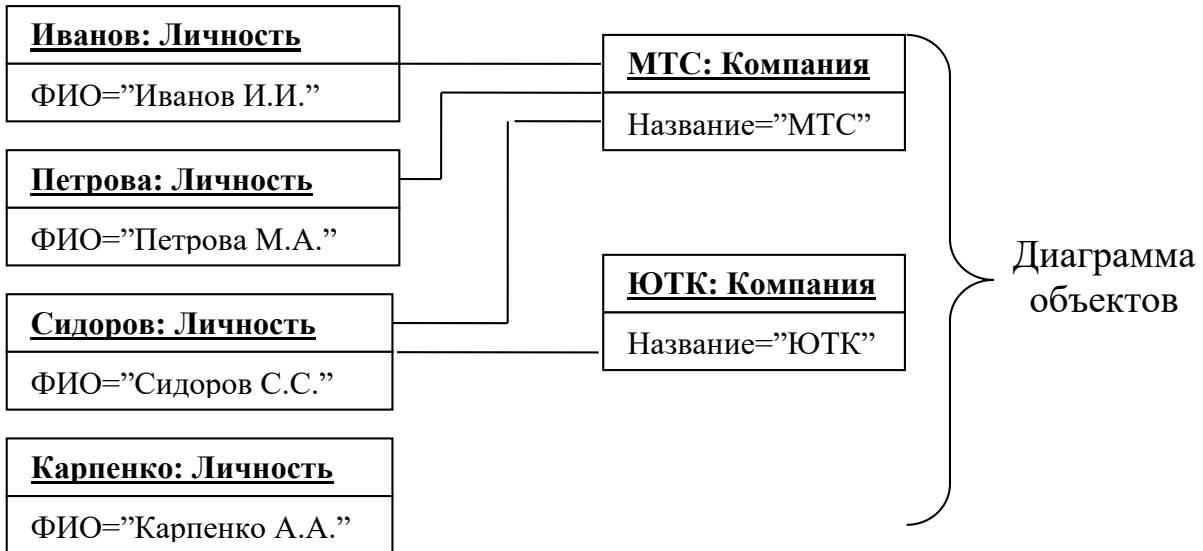
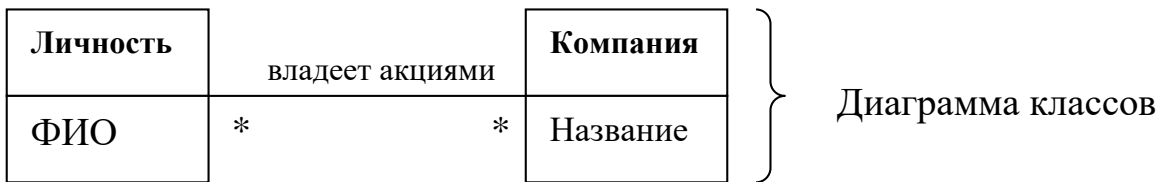


Рис. 7.5. Пример ассоциации и связей

Полюс ассоциации может иметь имя и кратность.

**Кратность** – это количество экземпляров одного класса, которые могут быть связаны с одним экземпляром другого класса через одну ассоциацию. Кратность ограничивает количество связанных между собой объектов. На диаграммах UML кратность указывается явно около конца линии, которой обозначается ассоциация. Значение кратности указывается в виде диапазона, например «1» (ровно один), «1..\*» (один и более) или «3..5» (от трех до пяти включительно). Символ «\*» обозначает слово «много» – нуль и более [12, 15, 16, 18].

Имена полюсов ассоциаций часто присутствуют в описаниях задач в виде существительных. Имя полюса указывается около конца ассоциации. Имена полюсов обязательны для установления ассоциаций между двумя объектами одного и того же класса.

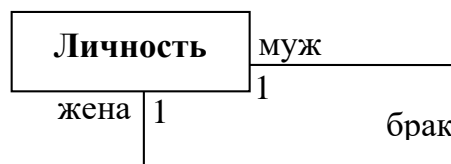


Рис. 7.6. Моделирование ссылок на один и тот же класс

### 3. Обобщение и наследование

**Обобщение** – это отношение между классом (суперклассом) и одной или несколькими его вариациями (подклассами). Обобщение объединяет классы по их общим свойствам, благодаря чему обеспечивается структурирование описания объектов. Суперкласс характеризуется общими атрибутами, операциями и ассоциациями. Подклассы добавляют к ним свои собственные атрибуты, операции и ассоциации. Подкласс наследует составляющие суперкласса [12, 15, 16, 18].

Пример обобщений приведен на Рис. 7.7. Обобщение обозначается не закрашенной стрелкой.

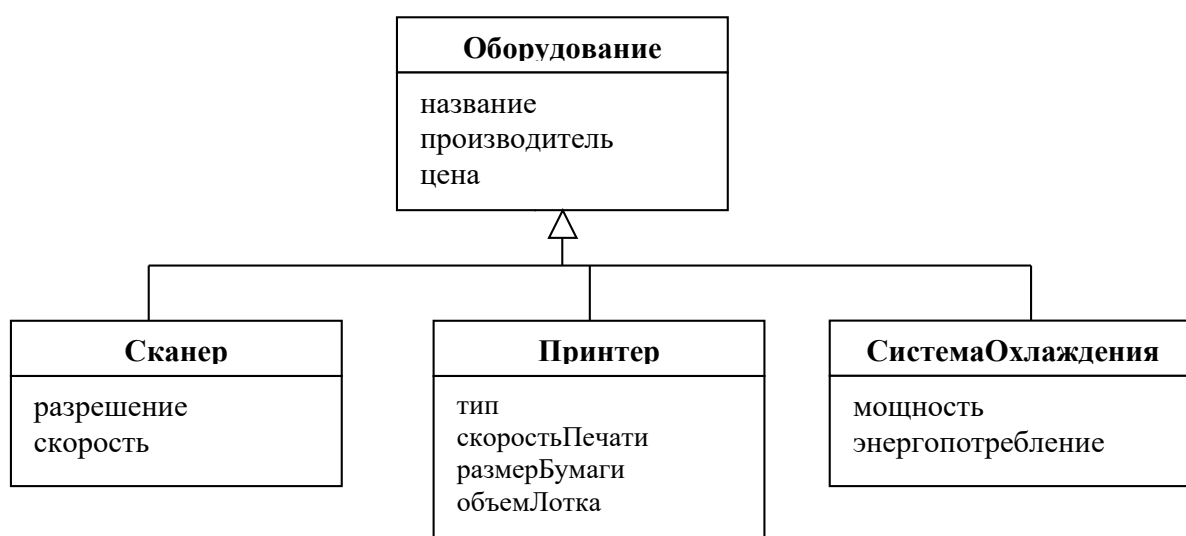


Рис. 7.7. Иерархия наследования

Термины обобщение, конкретизация и наследование описывают одну и ту же концепцию. Термины обобщение и конкретизация – это отношения между классами, противоположные друг другу по смыслу. Обобщение подчеркивает, что суперкласс обобщает подклассы. Конкретизация подчеркивает, что подклассы конкретизируют (уточняют) суперкласс. **Наследование** – это механизм совместного использования атрибутов, операций и ассоциаций объектами, классы которых находятся в отношениях обобщения (конкретизации) [12, 15, 16, 18].

### 4. Агрегация и композиция

В UML определено две формы отношения «часть-целое»: агрегация и композиция.

**Агрегация** – это частный случай ассоциации, описывающий объекты, состоящие из частей. Агрегат с семантической точки

зрения представляет собой расширенный объект, обрабатываемый многими операциями как единое целое, хотя физически он состоит из нескольких объектов [12, 15, 16, 18].



Рис. 7.8. Агрегация

**Композиция** – это частный случай агрегации, характеризующийся двумя дополнительными ограничениями. Составляющая часть может принадлежать не более чем одному агрегату. Более того, составляющая часть, приписанная к некоторому агрегату, автоматически получает срок жизни, совпадающий со сроком жизни агрегата. Таким образом, композиция подразумевает, что части принадлежат целому [12, 15, 16, 18].



Рис. 7.9. Композиция

## 5. Моделирование классов с помощью StarUML

StarUML – программный инструмент моделирования на UML со свободной лицензией. StarUML поддерживает одиннадцать различных типов диаграмм, принятых в нотации UML 2.0:

- 1) диаграмма классов (Class diagram);
- 2) диаграмма прецедентов (Use case diagram);
- 3) диаграмма сообщений (Sequence Diagram);
- 4) диаграмма сообщений роли (Sequence Role Diagram);
- 5) диаграмма коллаборации (Collaboration Diagram);
- 6) диаграмма коллаборации ролей;
- 7) диаграмма состояний (Statechart Diagram);
- 8) диаграмма действий (Activity Diagram);
- 9) диаграмма компонентов (Component Diagram);

- 10) диаграмма развертывания (Deployment Diagram);
- 11) композиционная структурная диаграмма (Composite Structure Diagram).

Главное окно StarUML (см. Рис. 7.10) содержит следующие элементы управления:

- главное меню и кнопки быстрого доступа в верхней части окна;
- панель элементов (Toolbox) с изображениями элементов диаграмм слева;
- рабочее поле диаграммы в центре;
- инспектор модели находится справа и содержит вкладки навигатора модели Model Explorer, навигатора диаграмм Diagram Explorer, окно редактора свойств Properties, окно документирования элементов модели Documentation и редактор вложений Attachments.

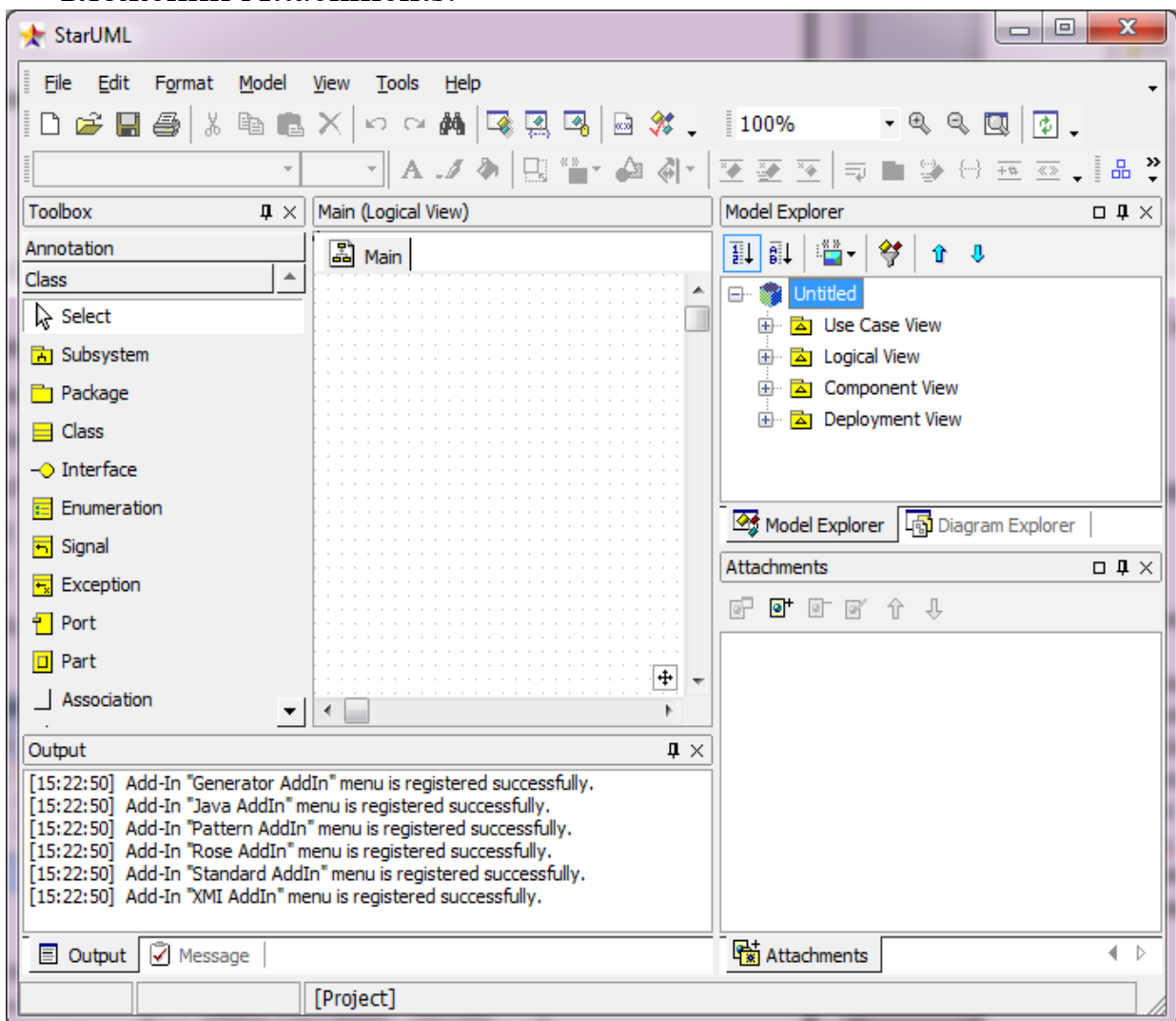


Рис. 7.10. Главное окно StarUML

Рассмотрим этапы создания модели классов.

1. Создание нового проекта. Новый проект будет автоматически создан при запуске программы StarUML, либо вызовом команды меню File→New Project. При этом нужно в диалоговом окне выбрать один из подходов поддерживаемых StarUML (см. Рис. 7.11). Выберем подход Rational Approach.

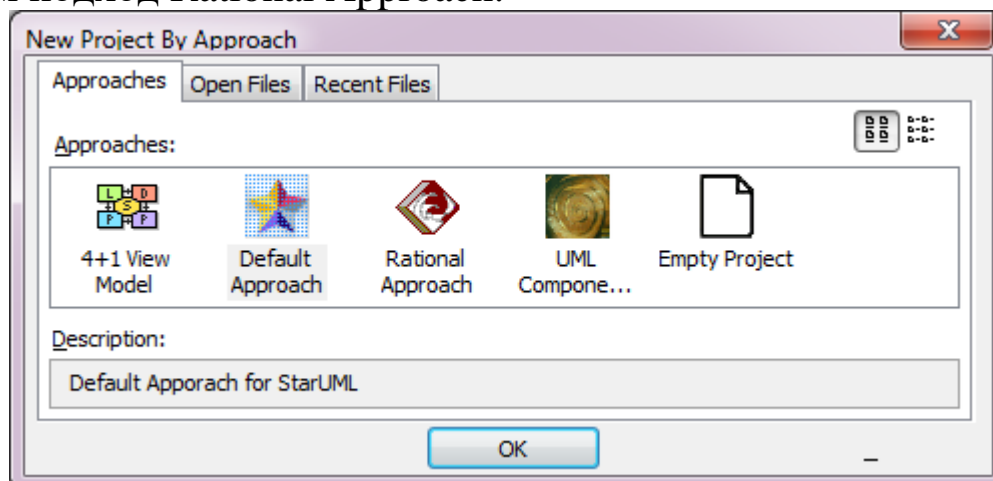
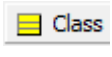
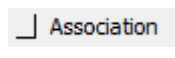


Рис. 7.11. Окно выбора подхода

2. Создание диаграммы классов: щелкнуть правой кнопкой мыши по папке Logical View в навигаторе модели, в контекстном меню выбрать пункт Add Diagram, в списке выбрать диаграмму классов Class Diagram.

3. Размещение классов на диаграмме: щелкнуть кнопку  Class на палитре инструментов, затем щелкнуть на рабочем поле и ввести имя класса, например Личность (см. Рис. 7.12).

Добавьте атрибуты с помощью пункта контекстного меню класса Add→Attribute. Таким же способом создайте второй класс Компания.

Добавьте ассоциацию с помощью кнопки  Association палитры инструментов. С помощью окна свойств Properties введите название ассоциации (Name), кратность (End1. Multiplicity и End2.Multiplicity) и роли полюсов (End1.Name и End2.Name) ассоциации.

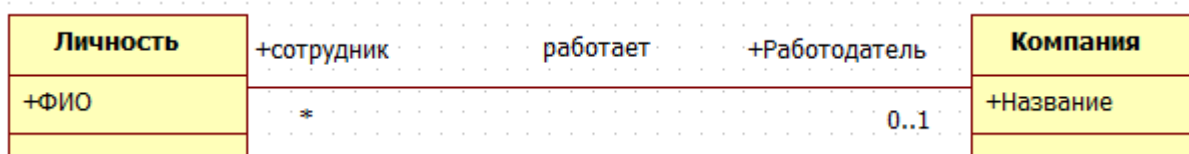


Рис. 7.12. Диаграмма классов

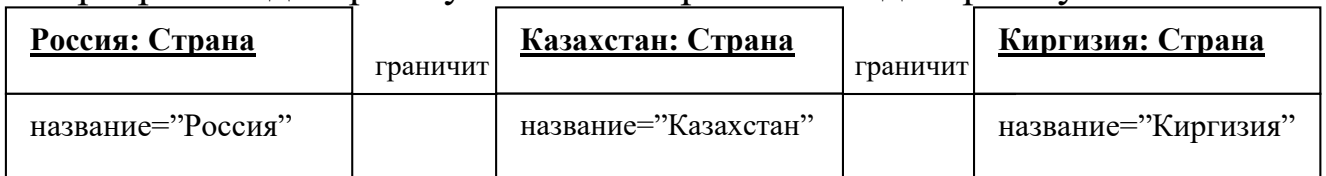
4. Сохраните модель.

## Задания для самостоятельного выполнения

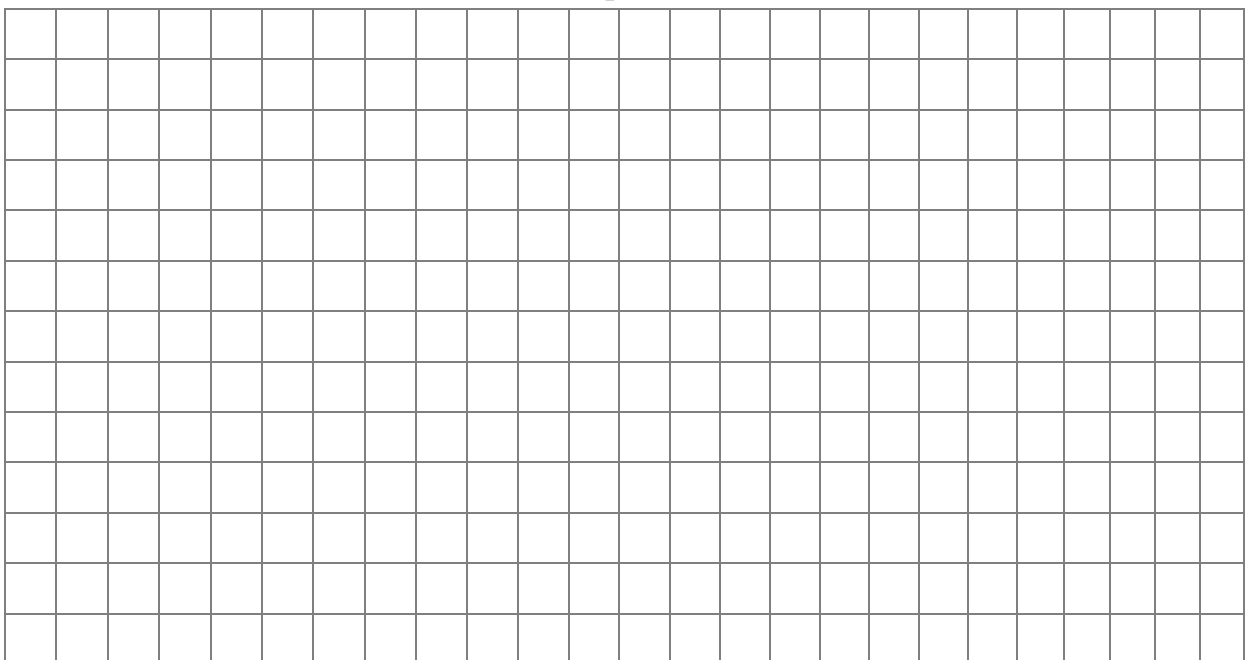
1. Ответьте на вопросы:

- a) В чем сущность объектно-ориентированного моделирования и проектирования ИС?
- b) Каковы основные характеристики объектно-ориентированного подхода?
- c) Какие типы моделей используются при объектно-ориентированном проектировании ИС?
- d) Каково назначение модели классов? модели состояний? модели взаимодействий?
- e) Что такое UML?
- f) Какие возможности дает использование UML?
- g) Что такое объект, класс? Как они обозначаются в UML?
- h) Что такое связь, ассоциация? Как они обозначаются в UML?
- i) Что такое обобщение и наследование? Как обозначается наследование в UML?
- j) Что такое агрегация и композиция? Как они обозначаются в UML?
- k) Каково назначение программного продукта StarUML?

2. Превратите диаграмму объектов с рис.4.10 в диаграмму классов.

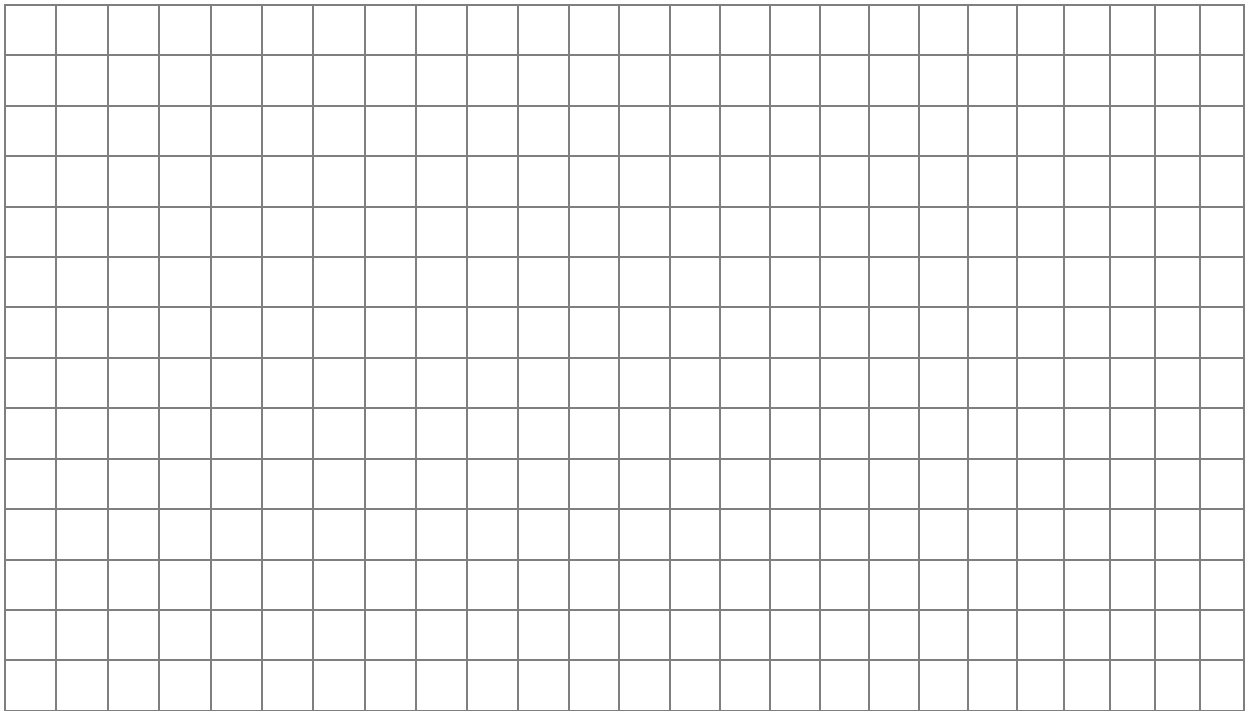


*Рис. 7.10. Диаграмма объектов*









5. Подготовьте диаграмму классов по диаграмме объектов с Рис. 7.12.

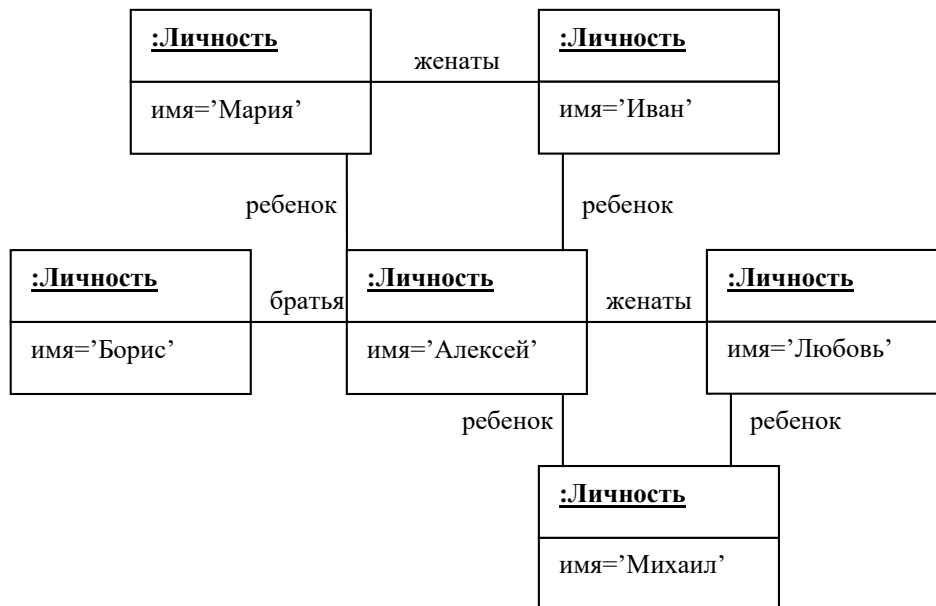
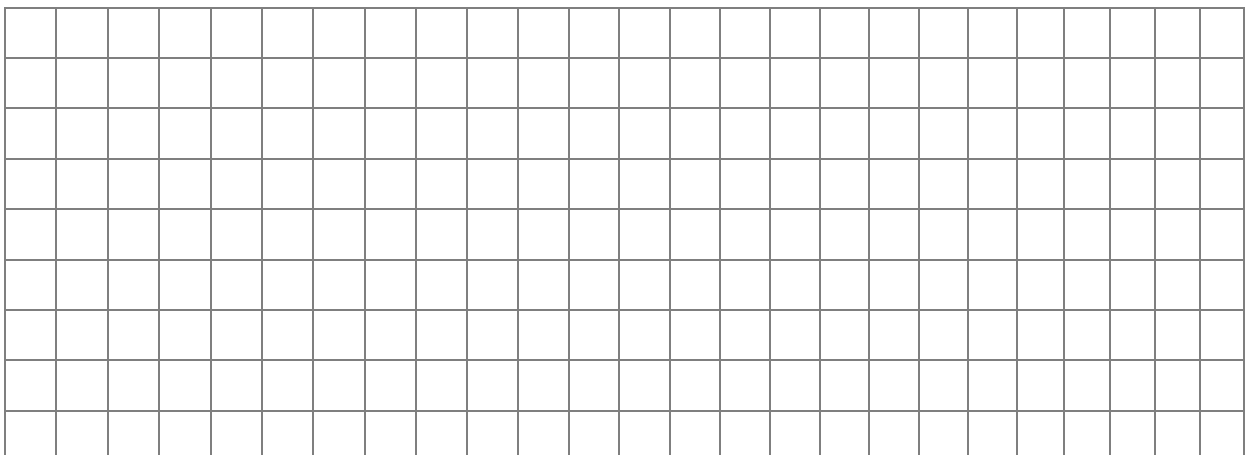


Рис. 7.12



6. Подготовьте диаграмму классов по диаграмме объектов с Рис. 7.13. Указание: при построении диаграммы следует использовать одно обобщение.

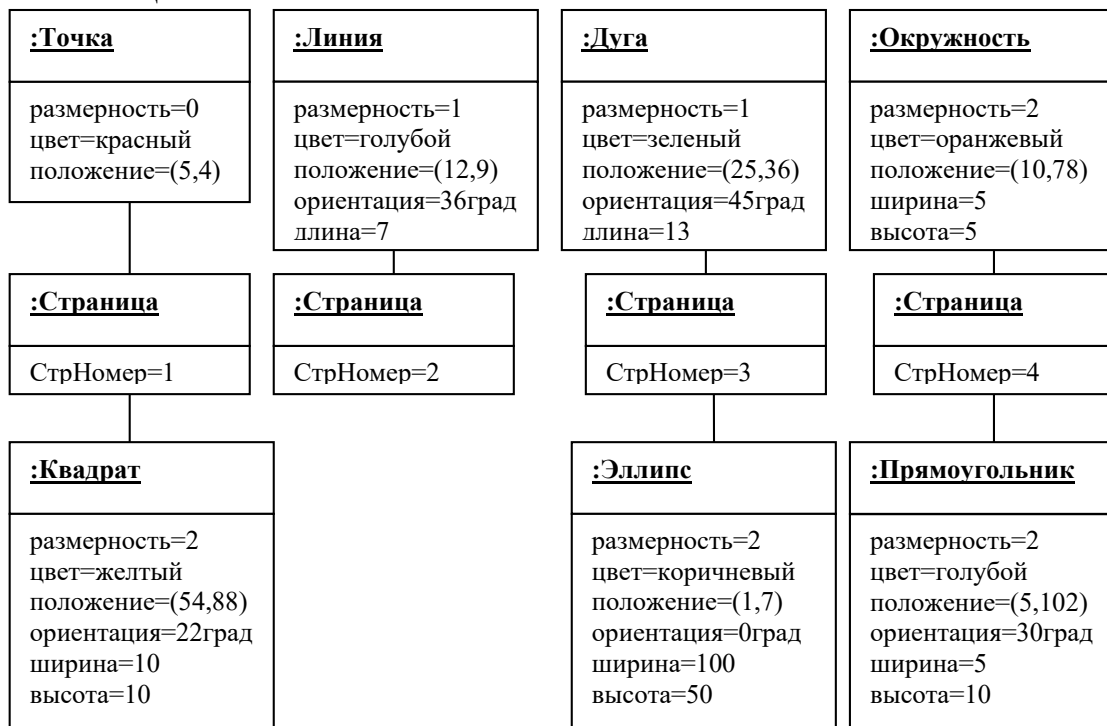
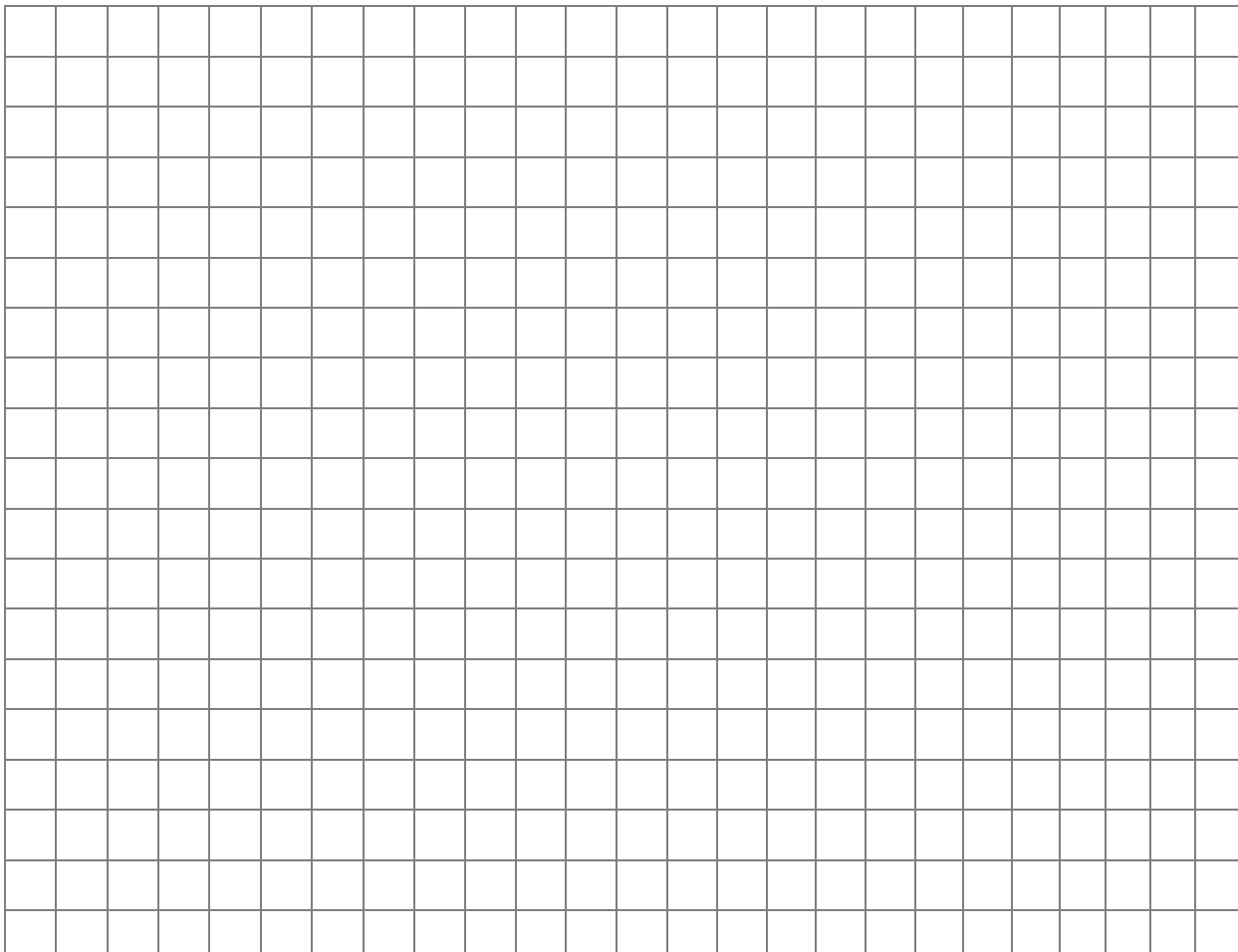
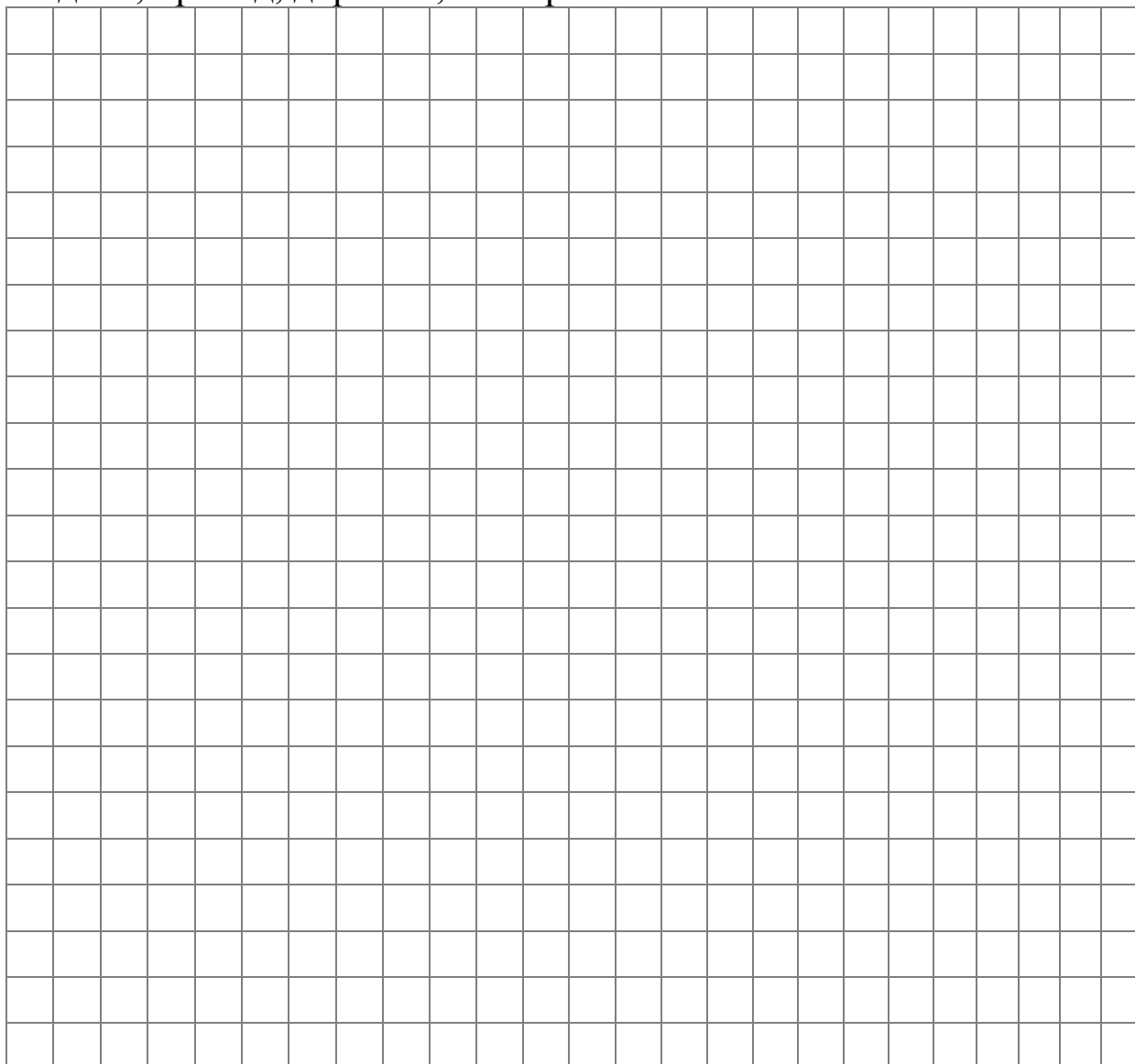


Рис. 7.13. Диаграмма объектов



7. Подготовьте диаграмму классов для каждой группы классов. Добавьте на каждую диаграмму не менее 10 отношений. При подготовке диаграмм можно добавлять дополнительные классы.

- 1) Среда разработки Delphi, язык программирования, конструктор форм, инспектор объектов, библиотека компонентов, редактор кода, проект, менеджер проекта, форма, кнопка, поле редактирования, процедура обработки события.
- 2) Графика, графическая поверхность, точка, линия, дуга, эллипс, прямоугольник, многоугольник, заливка, цвет, текст, шрифт, стиль, график, диаграмма, анимация.
- 3) База данных, СУБД, локальная БД, удаленная БД, сервер, клиент, таблица, связь, запись, запрос, язык структурированных запросов, поиск, фильтр.
- 4) Файловая система, файл, АСП-файл, двоичный файл, каталог, диск, привод, дорожка, сектор.



8. Подготовьте диаграмму классов согласно своему варианту (см. тему 2), используя StarUML.

**Письменный отчет**

A large grid for writing the report, consisting of 20 columns and 25 rows of small squares.

## Тема 8. Моделирование состояний

*Цель:* сформировать навыки моделирования состояний.

### Теоретические сведения

#### 1. Моделирование состояний

Модель состояний описывает последовательности операции, происходящих в системе в ответ на внешние воздействия. Модель состояний состоит из нескольких диаграмм состояний, по одной на каждый класс, поведение которого во времени важно для приложения.

*Диаграммы состояний* используются для описания поведения сложных систем. Они определяют все возможные состояния, в которых может находиться объект, а также процесс смены состояний объекта в результате некоторых событий. Диаграмма состояний – это стандартная концепция из информатики (графическое представление конечного автомата), связывающая события и состояния. События представляют внешние воздействия, а состояния – значения объектов.[15, 16, 18]

*Диаграмма состояний* – это граф, узлами которого являются состояния, а направленными дугами – переходы между состояниями.

Название состояния должно быть уникальным в рамках диаграммы.

Модель состояний состоит из множества диаграмм состояний, по одной на каждый класс, поведение которого с течением времени важно для приложения.

Диаграммы состояний должны быть согласованы по интерфейсам (событиям и сторожевым условиям).

Класс, имеющий несколько состояний, характеризуется важным поведением во времени. Если же класс обладает одним состоянием, его поведение во времени можно игнорировать. Диаграммы состояний с одним состоянием можно описать в простой форме без всякой графики, а именно в виде таблицы воздействий и откликов, в которой будут приводиться события и сторожевые условия, а также вызываемое ими поведение.

Одноразовые диаграммы состояний описывают объекты с конечным сроком существования. Такие диаграммы имеют начальное и конечное состояние. Сразу после создания объект

оказывается в начальном состоянии. Вход в конечное состояние означает уничтожение объекта.

### **Основные обозначения для диаграммы состояний**

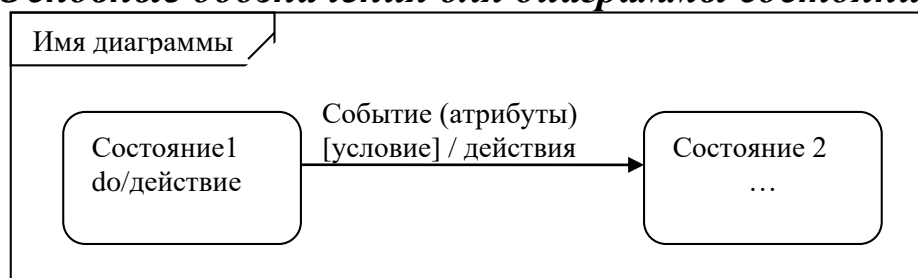


Рис. 8.1. Основные обозначения для диаграммы состояний

*Состояние* обозначается прямоугольником со скругленными углами, в котором может быть указано имя состояния. Начальное состояние обозначается сплошным кружком, конечное – «бычьим глазом».

*Переход* изображается линией, соединяющей исходное состояние с целевым.

*Событие* сигнала изображается меткой на переходе. После названия события в круглых скобках можно указать атрибуты. События изменения обозначаются ключевым словом *when*, после которого в круглых скобках указывается логическое выражение. Событие времени также указывается ключевым словом *when*, после которого в круглых скобках указывается временное выражение, или ключевое слово *after*, после которого в круглых скобках указывается интервал времени.

*Диаграмма состояний* заключается в прямоугольную рамку. Название диаграммы указывается в небольшом прямоугольнике в левом верхнем углу рамки.

*Сторожевые условия* могут быть указаны в квадратных скобках после события.

*Действия* могут прикрепляться к переходу или состоянию. Указываются после символа «/» [15, 16, 18].

## **2. Моделирование состояний с помощью StarUML**

Рассмотрим этапы создания модели состояний на примере светофора. Согласно ГОСТ Р 52289-2004 для светофоров Т.1, Т.3 любых исполнений, Т.2 и Т.9 (см. Рис. 8.2) соблюдают последовательность включения сигналов: красный - красный с желтым - зеленый - желтый - красный... При этом длительность

сигнала "красный с желтым" должна быть не более 2 с, длительность желтого сигнала во всех случаях должна быть 3 с.

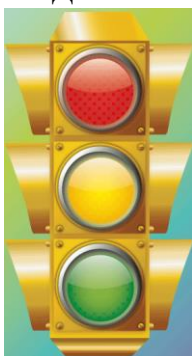


Рис. 8.2. Светофор

1. Создайте новый проект: File→New Project. При этом выберите один подход Rational Approach.

2. Создание диаграммы состояний: щелкнуть правой кнопкой мыши по папке Logical View в навигаторе модели, в контекстном меню выбрать пункт Add Diagram, в списке выбрать диаграмму классов Statechart Diagram (см. Рис. 8.3).

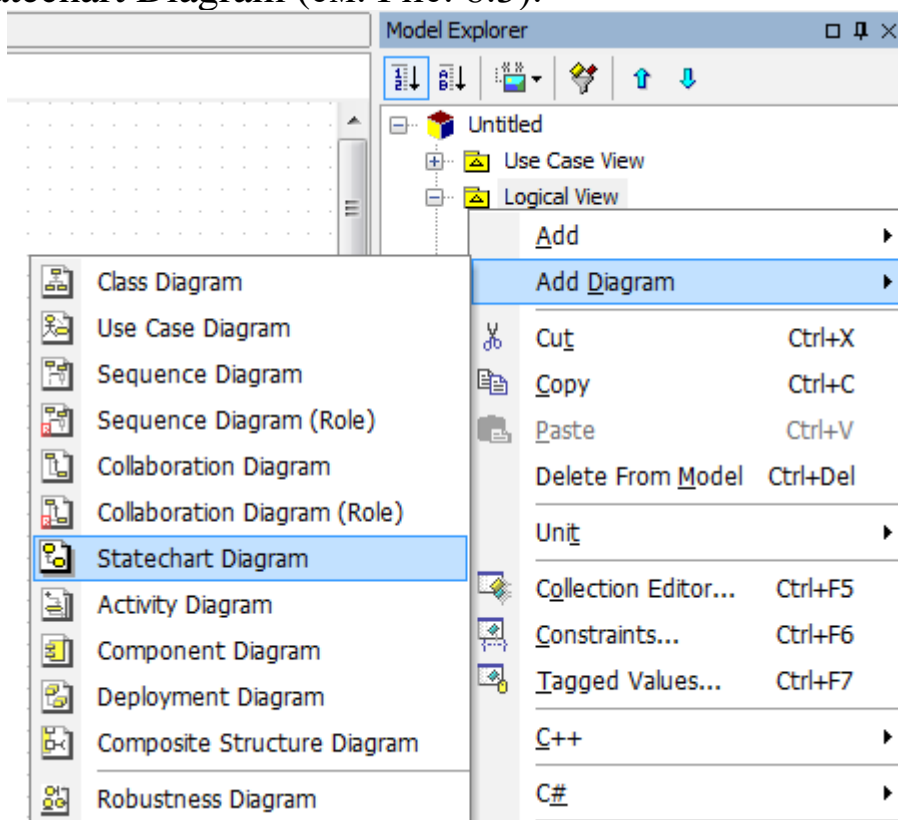

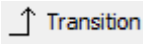


Рис. 8.3. Создание диаграммы состояний

Диаграмму состояний можно связать с классом, создав ее из контекстного меню конкретного класса.

3. Размещение состояний на диаграмме: щелкнуть кнопку  State на палитре инструментов, затем щелкнуть на рабочем поле и ввести имя состояния, например Красный (см. Рис. 8.4). Таким способом

создайте 4 состояния: красный, красный с желтым, зеленый, желтый.

Добавьте переходы с помощью кнопки  палитры инструментов.

С помощью окна свойств Properties введите названия переходов (Name).

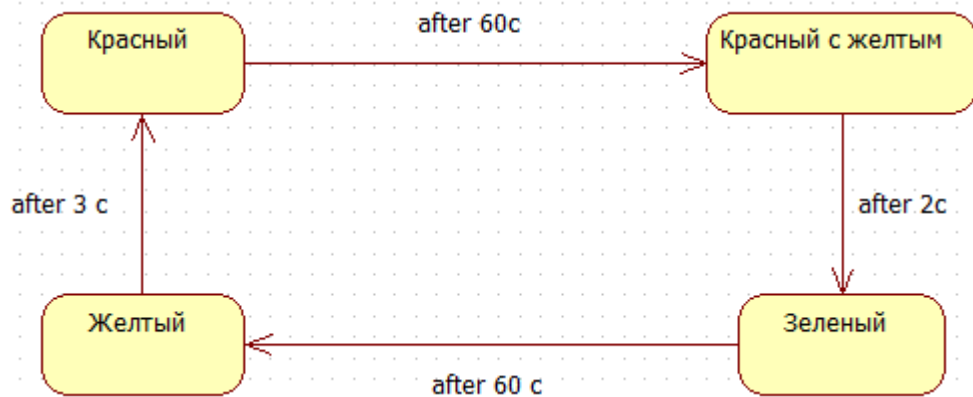


Рис. 8.4. Диаграмма состояний

4. Сохраните модель.

5. Добавим еще одно состояние, которой соответствует режиму работы светофора в ночное время – от 23:00 до 7:00 включается мигающий желтый.

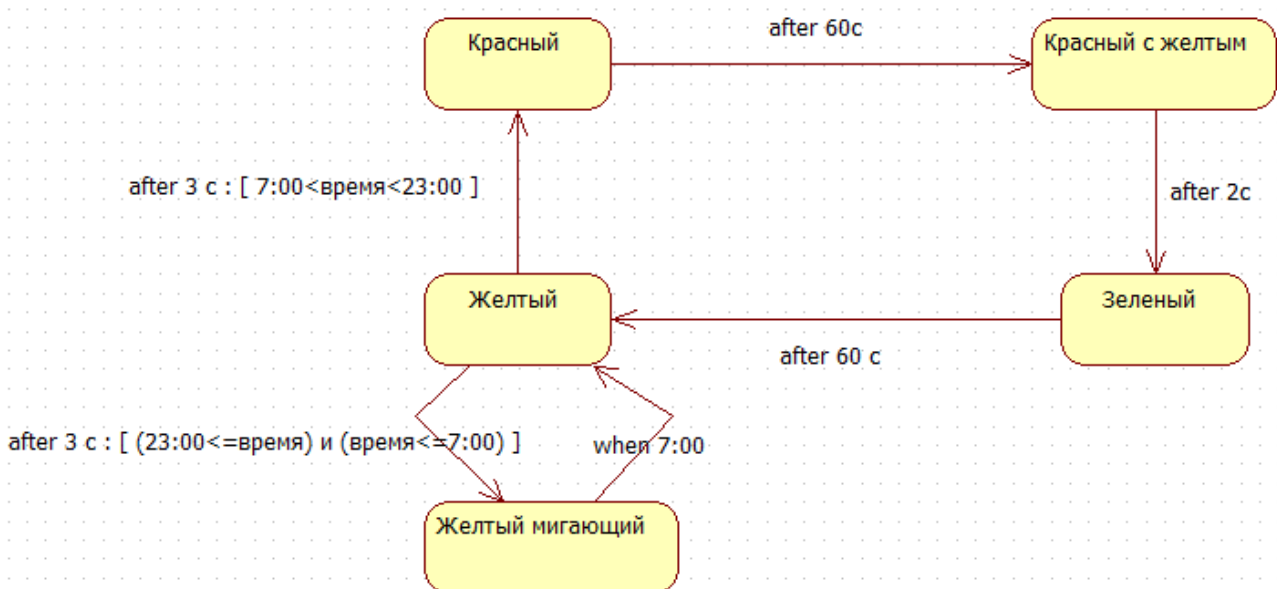


Рис. 8.5. Диаграмма состояний светофора

### Задание для самостоятельного выполнения

1. Ответьте на следующие вопросы:

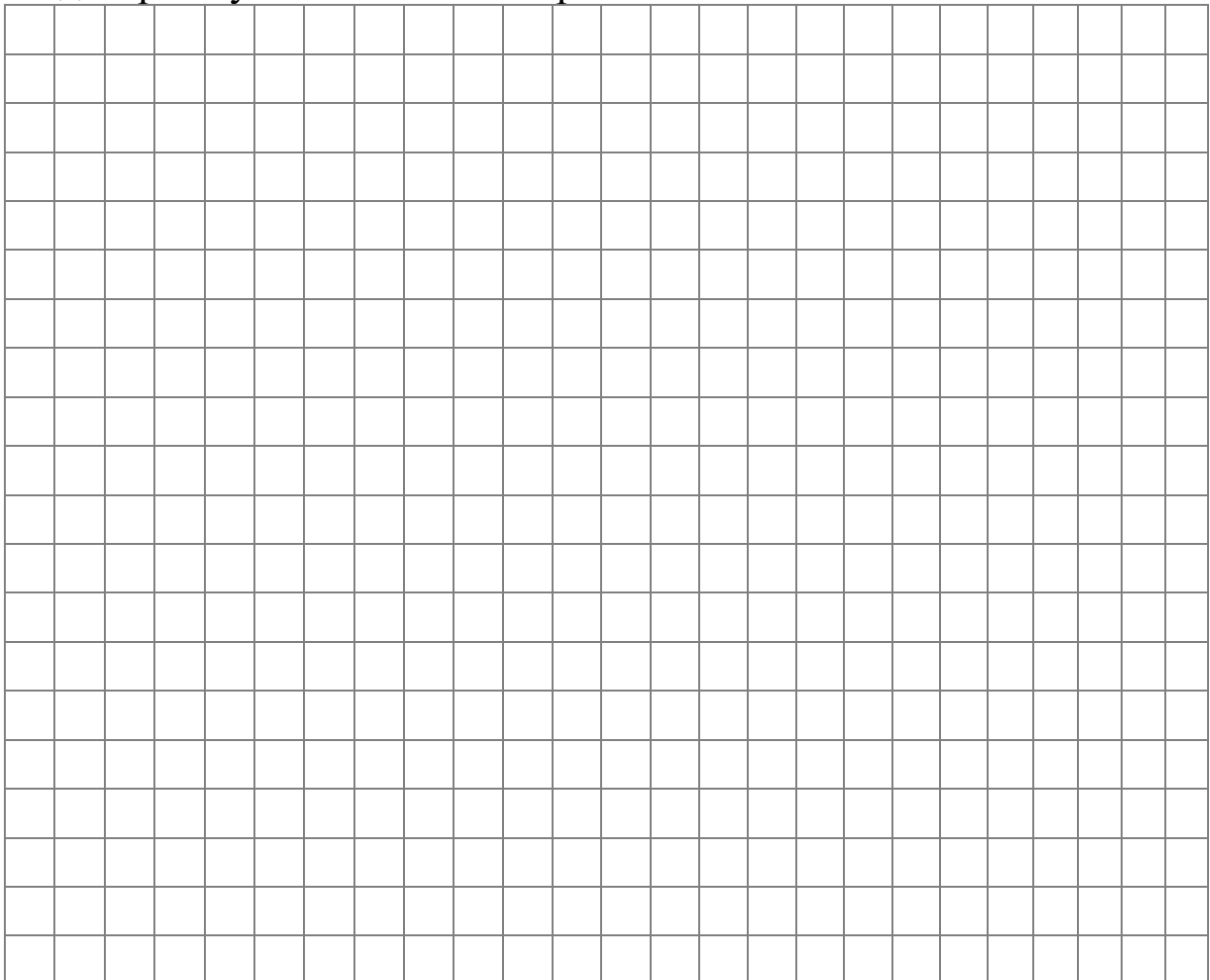
- В чем заключается назначение модели состояний?
- Что такое событие?
- Перечислите виды событий и укажите их обозначение.




d) Что такое состояние? переход? сторожевое условие?

e) Как обозначаются действия и текущая деятельность на диаграммах состояний UML?

2. Таймер – это устройства, отмеряющие заданный интервал времени с момента запуска с секундомером обратного отсчёта. Простейший таймер состоит из дисплея и двух кнопок А и В. Часы могут работать в двух режимах: установки интервала времени и отсчета. В режиме отсчета таймер показывает оставшиеся часы и минуты, между ними мигает символ двоеточия. Режим установки состоит из двух подрежимов: установка часов и установка минут. Кнопка А позволяет выбрать режим. Каждый раз при её нажатии происходит переход к очередному режиму в последовательности: отсчет, установка часов, установка минут, отсчет и т.д. Кнопка В позволяет увеличивать значение часов или минут на единицу при каждом нажатии в одном из режимов установки. Чтобы кнопка смогла породить новое событие её необходимо отпустить. Нарисуйте диаграмму состояний таймера.



3. Постройте диаграмму состояний контекстного меню. Контекстное меню – это меню, которое отображается в отдельном окне и показывает действия, которые можно произвести с объектом, для которого оно вызвано (например, на Рис. 8.7 приведено контекстное меню Корзины). Вызывается такое меню, как правило, по нажатию правой кнопки мыши, специальной клавиши (  Menu), или сочетания  $\uparrow$  Shift + F10 на клавиатуре.

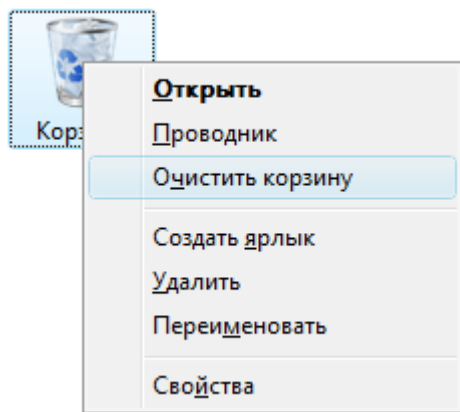
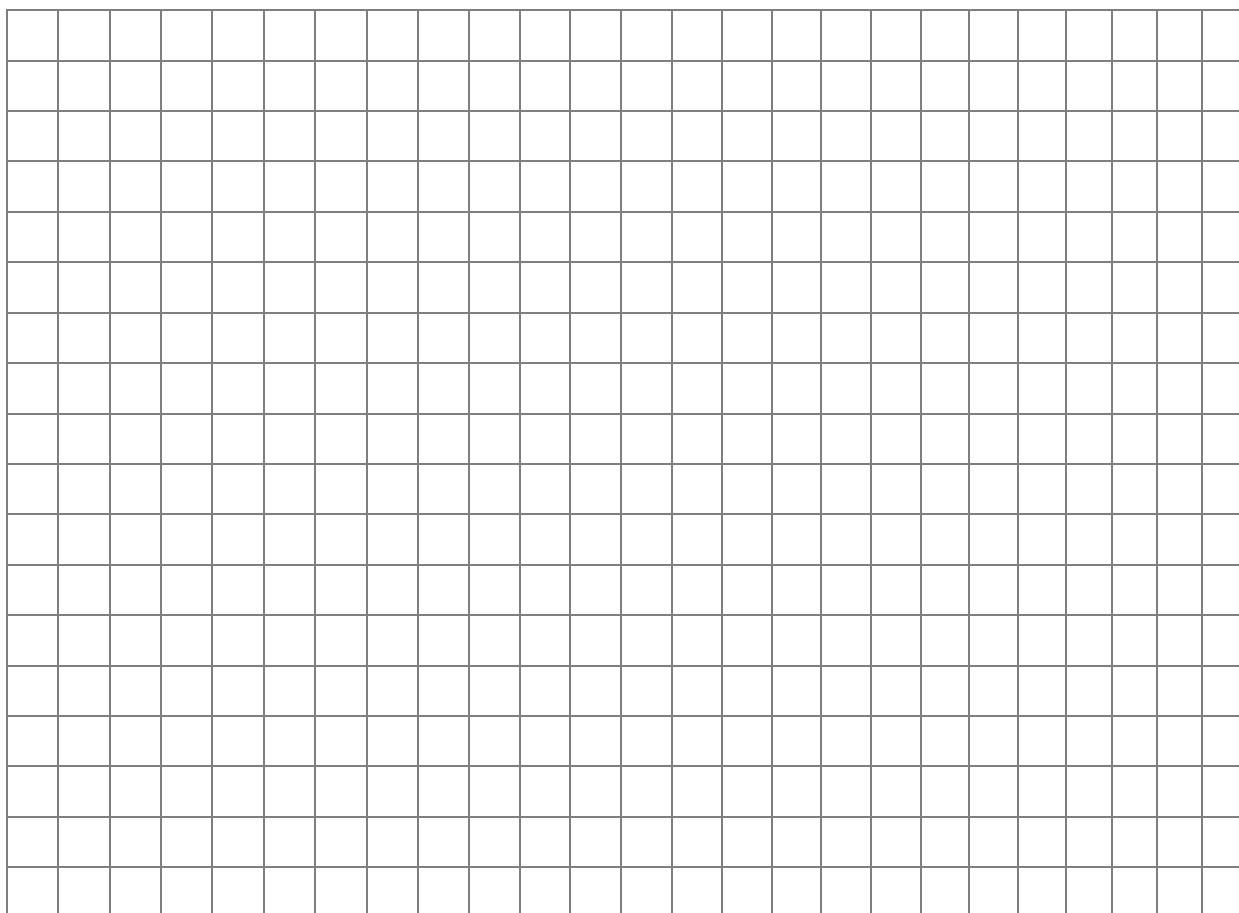
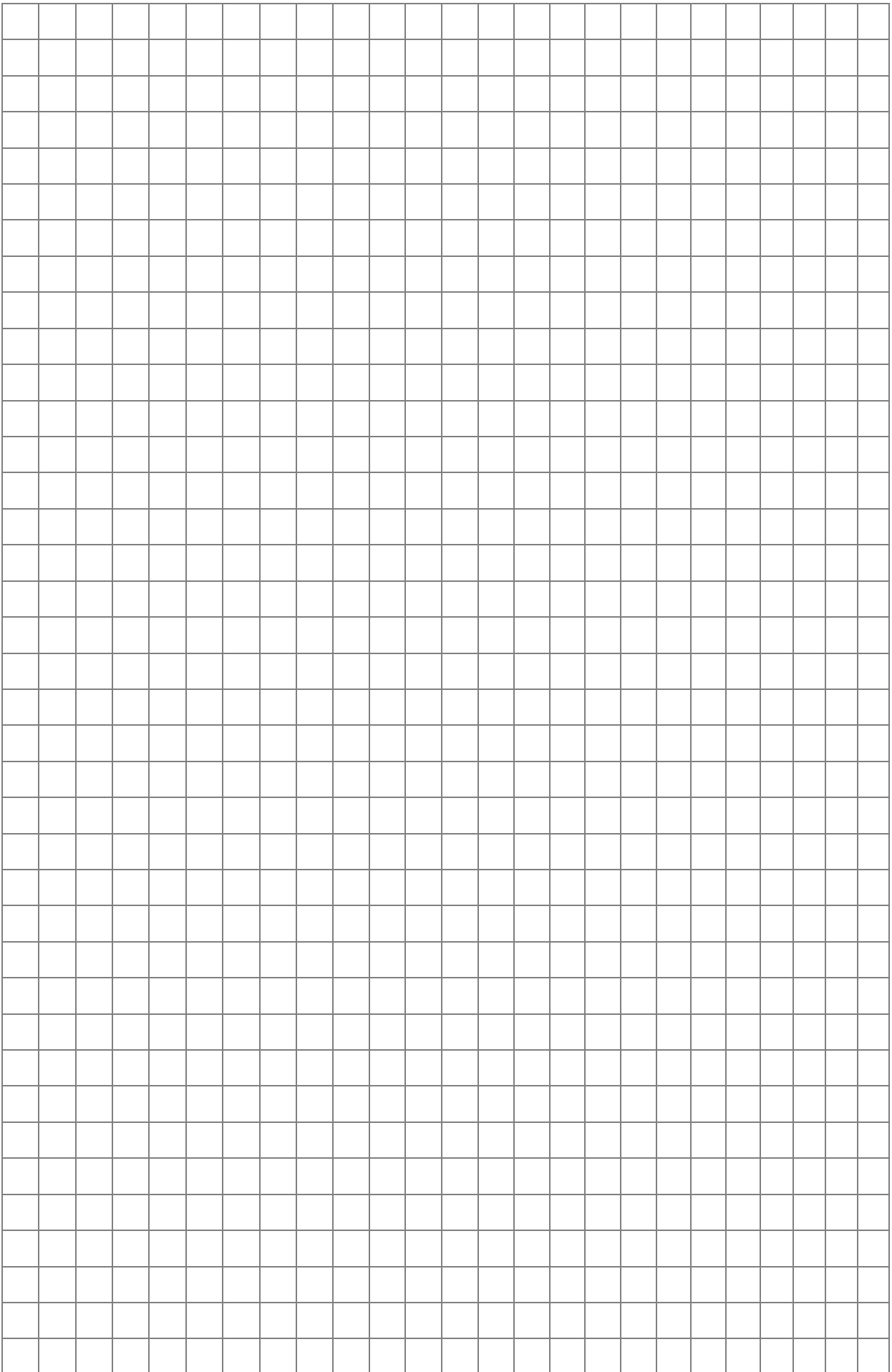


Рис. 8.7. Контекстное меню



4. Постройте с помощью StarUML диаграммы состояний для классов, разработанных в теме 4.





## Тема 9. Моделирование взаимодействий

*Цель:* сформировать навыки построения моделей взаимодействия.

### Теоретические сведения

#### 1. Варианты использования, диаграммы последовательности, диаграммы деятельности

Модель взаимодействия – это глобальный взгляд на поведение множества объектов, тогда как модель состояний – это редуцированное представление индивидуального поведения объектов. Для полного описания поведения необходимы обе модели.

Взаимодействия можно моделировать на разных уровнях абстрагирования. На самом высоком уровне взаимодействие системы со внешними действующими лицами описывается *вариантами использования*. Каждый вариант использования описывает элемент функциональности, предоставляемой системой её пользователям. Варианты использования полезны для представления в модели неформальных требований.

*Диаграммы последовательности* более детализированы, они показывают сообщения, которыми обмениваются объекты с течением времени. К сообщениям относятся асинхронные сигналы и вызовы процедур. Диаграммы последовательности полезны для демонстрации последовательностей поведения, видимых пользователям системы. [15, 16, 18]

*Диаграммы деятельности* содержат всю информацию и показывают поток управления между этапами вычислений. Диаграммы деятельности могут показывать не только потоки управления, но и потоки данных. С их помощью документируются этапы диаграмм последовательности, необходимые для реализации операций или бизнес-процесса. [15, 16, 18]

#### *Пример: Система подготовки счета на оплату*

Рассмотрим систему подготовки счета на оплату. Счет на оплату – это документ от продавца к покупателю, содержащий реквизиты для оплаты и перечень товаров и (или) услуг.

Обязательными в счете являются реквизиты, по которым плательщик будет платить:

- наименование организации или Индивидуального предпринимателя;
- ИНН и КПП;
- банковские реквизиты, расчетный или лицевой счет, кор. счет, наименование банка и его БИК;
- перечисленные товары или услуги.

1) Перечислим действующие лица, задействованные в подготовке счета к оплате.

- **Покупатель** – человек, инициирующий покупку товара или услуг.
- **Продавец** – сотрудник, авторизованный для оформления счета.
- **База данных о товарах** – хранилище сведений о имеющихся товарах и оказываемых услугах.

2) Варианты использования:

- **Создание счета на оплату.** Покупатель сообщает о товарах и услугах, которые хочет приобрести.
- **Поиск счета на оплату в архиве.** Продавец на основе счета на оплату формирует счет-фактуру в налоговые органы.

3) Диаграмма вариантов использования для системы подготовки счета на оплату.

На Рис. 9.1 показана диаграмма вариантов использования.



Рис. 9.1. Диаграмма вариантов использования

4) Типичные сценарии для каждого варианта использования. Сценарий – это пример, в котором не обязательно задействуется вся функциональность варианта использования.

Типичные сценарии для каждого варианта использования.

**Создание счета на оплату.** Покупатель сообщает продавцу о товарах, которые хочет приобрести. Продавец в базе данных о товарах проверяет их наличие и вносит сведения о них в новый счет. Продавец

спрашивает у покупателя его реквизиты. Покупатель их сообщает. Продавец распечатывает счет на оплату и передает покупателю.

**Поиск счета в архиве.** Продавец нажимает на кнопку «Поиск» в системе подготовки счета и вводит период времени, за который нужно вывести счета. Затем продавец нажимает кнопку «Ок». Система выводит все счета, соответствующие поисковому запросу.

5) Сценарии исключительных ситуации для каждого варианта использования.

**Создание счета на оплату.** Покупатель сообщает продавцу о товарах, которые хочет приобрести. Продавец в базе данных о товарах проверяет их наличие и обнаруживает их отсутствие. Продавец сообщает покупателю об этом.

**Поиск счета в архиве.** Продавец нажимает на кнопку «Поиск» в системе подготовки счета и вводит период времени, за который нужно вывести счета. Дата конца периода поиска меньше дату начала периода поиска. Система сообщает об этом продавцу.

6) Подготовьте диаграмму последовательности первого сценария из пункта 4.

На Рис. 9.2 показана диаграмма последовательности для первого сценария из пункта 4.



Рис. 9.2. Диаграмма последовательности

## 2. Моделирование состояний с помощью StarUML

Рассмотрим этапы создания диаграммы вариантов использования на примере системы подготовки счета на оплату.

1. Создайте новый проект: File→New Project. При этом выберите один подход Rational Approach.

2. Создание диаграммы вариантов использования: щелкнуть правой кнопкой мыши по папке Logical View в навигаторе модели, в контекстном меню выбрать пункт Add Diagram, в списке выбрать диаграмму классов Use Case Diagram (см. Рис. 9.3).

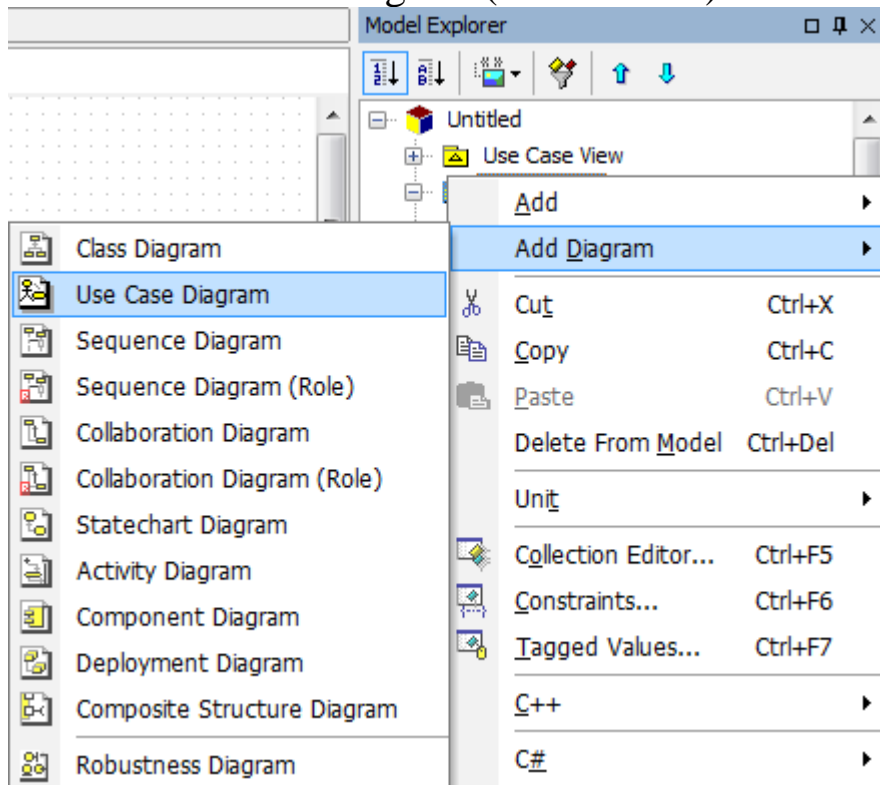



Рис. 9.3. Создание диаграммы вариантов использования

3. Размещение действующих лиц на диаграмме: щелкнуть кнопку  Actor на палитре инструментов, затем щелкнуть на рабочем поле и ввести имя действующего лица, например Покупатель (см. Рис. 9.4). Таким способом создайте 3 действующих лиц: покупатель, продавец, БД о товарах.


Добавьте ассоциации между действующими лицами и вариантами использования с помощью кнопки  Association палитры инструментов.





Рис. 9.4. Диаграмма вариантов использования

4. Сохраните модель.

5. Создание диаграммы последовательности: щелкнуть правой кнопкой мыши по папке Logical View в навигаторе модели, в контекстном меню выбрать пункт Add Diagram, в списке выбрать диаграмму классов Sequence Diagram (см. Рис. 9.5).

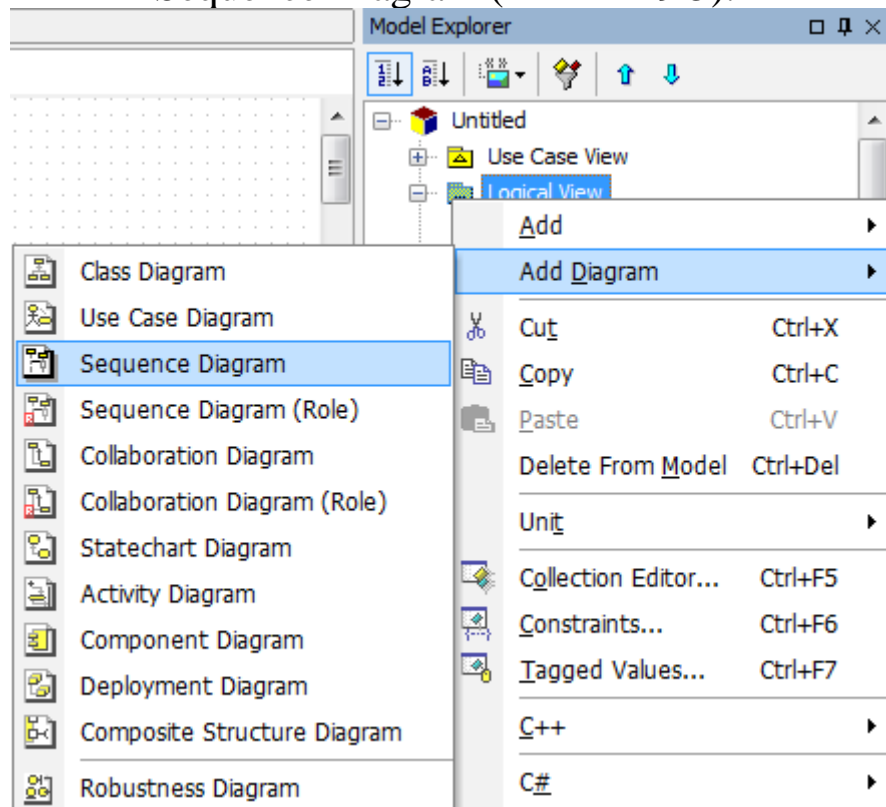


Рис. 9.5. Создание диаграммы последовательности

6. Разместите объекты **Object** и сообщения **Stimulus** на диаграмму (см. Рис. 9.6).



Рис. 9.6. Диаграммы последовательности

### Задания для самостоятельного выполнения

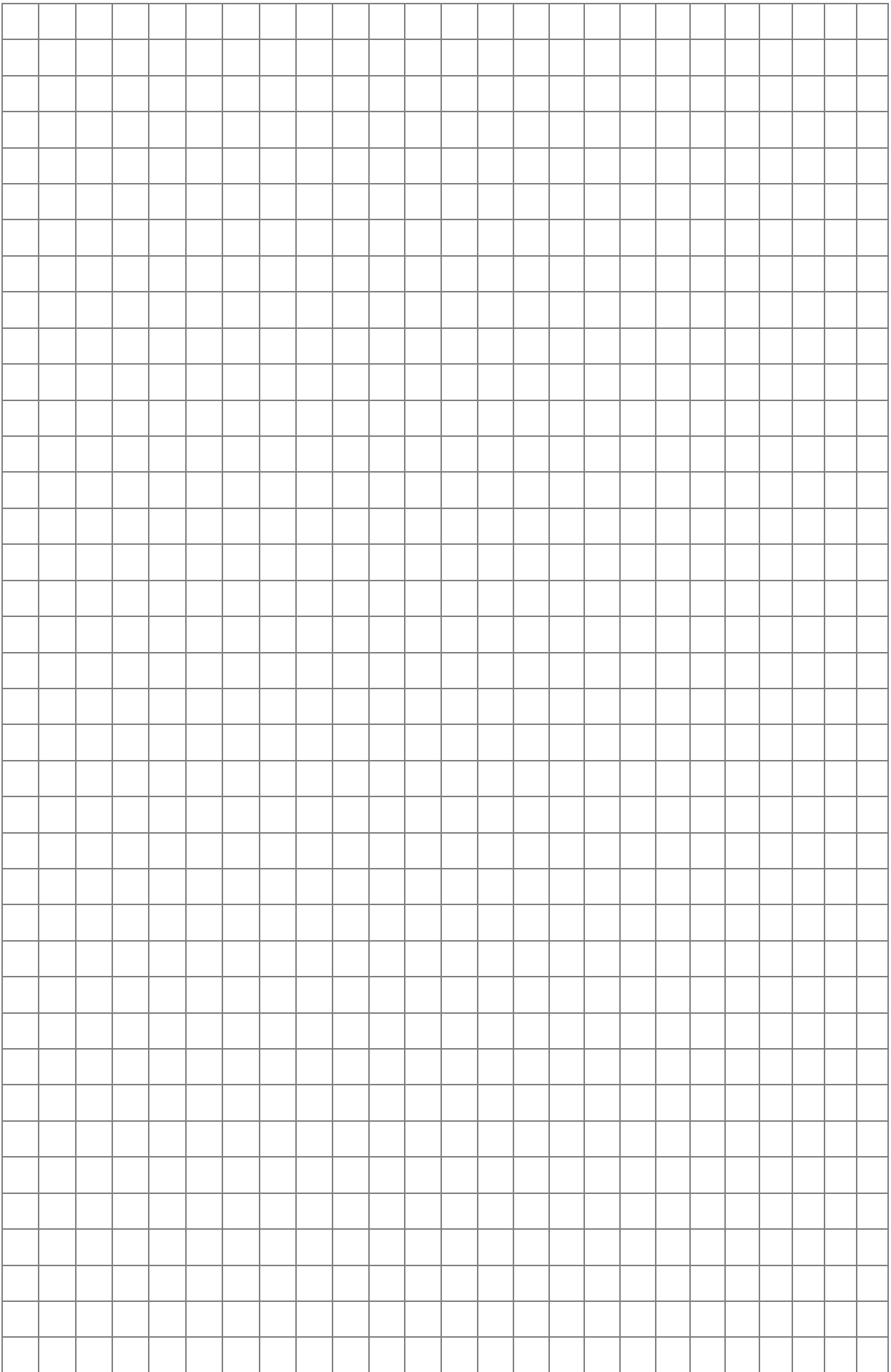
1. Подготовьте диаграммы взаимодействий для предметной области своего варианта (см. тему 2).

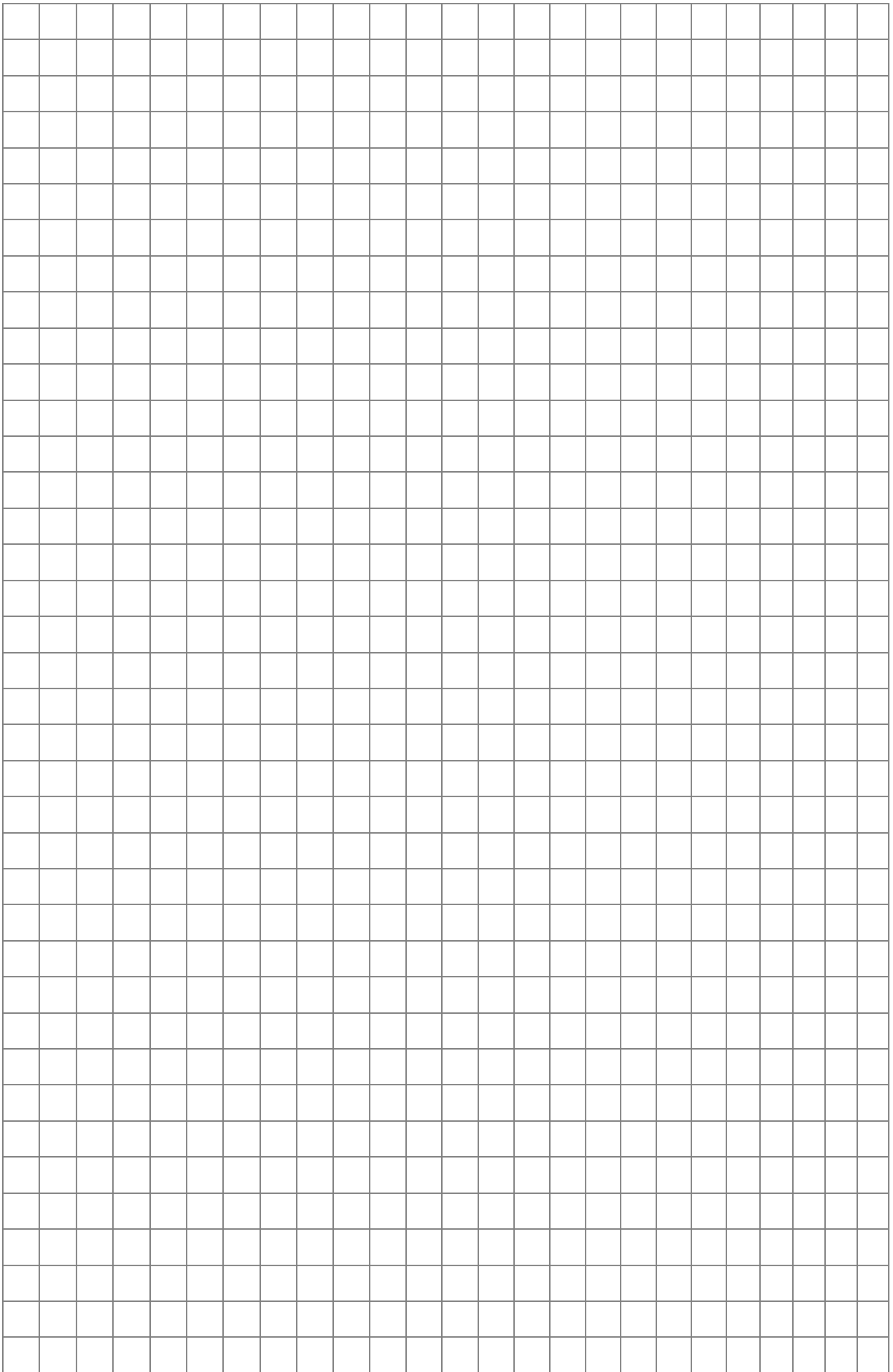
- a) Перечислите действующие лица. Объясните важность каждого из них.
- b) Подготовьте диаграмму вариантов использования.
- c) Подготовьте типичный сценарий для каждого варианта использования. Помните, что сценарий – это пример, в котором не обязательно задействуется вся функциональность варианта использования.
- d) Подготовьте сценарий исключительной ситуации для каждого варианта использования.
- e) Подготовьте диаграмму последовательности для каждого сценария.

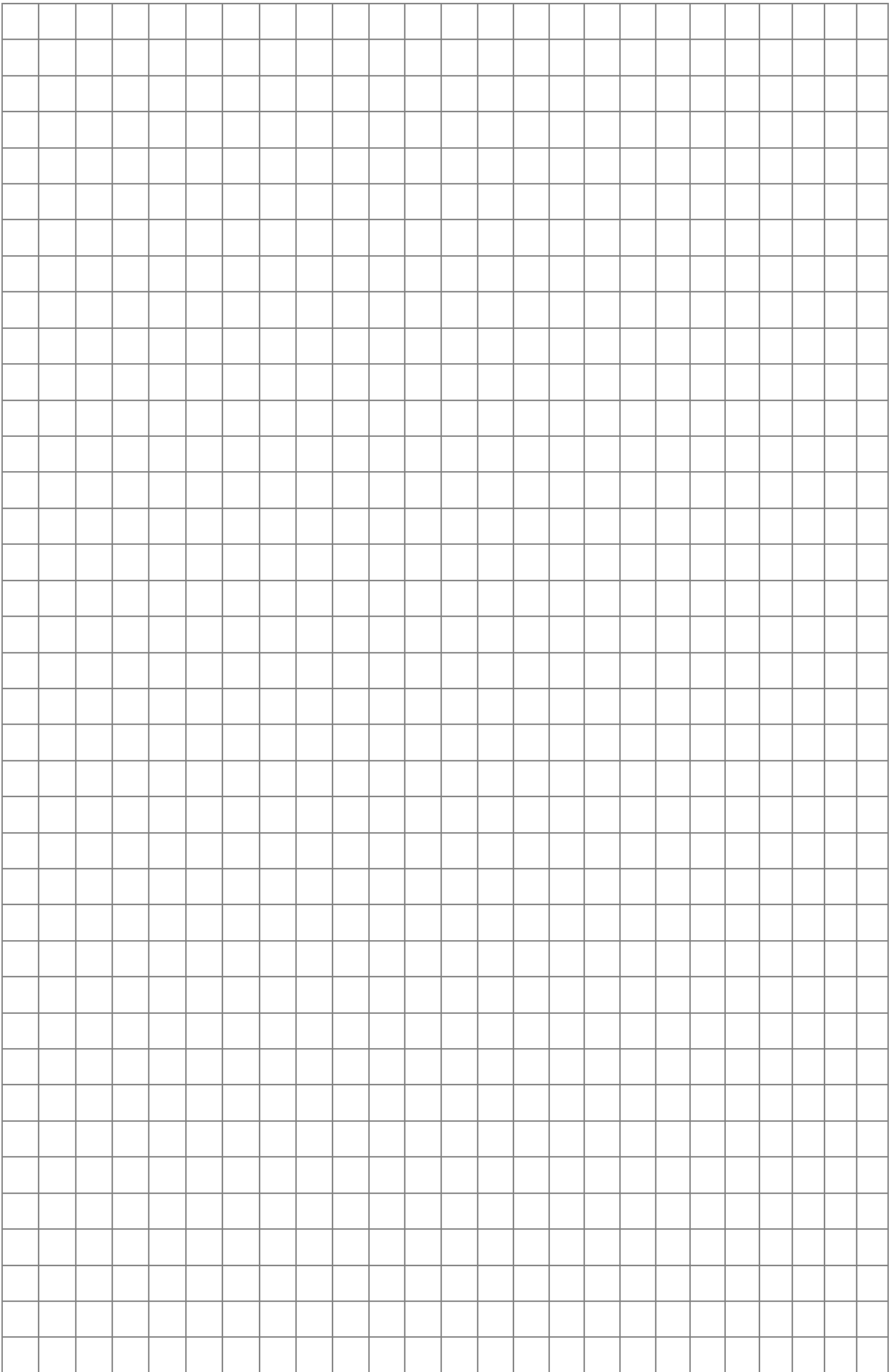
2. Ответьте на вопросы:

- a) Каково назначение моделирования взаимодействий?
- b) Что такое варианты использования?
- c) Каково назначение диаграмм последовательности и деятельности?









## Тема 10. RAD-технология разработки приложения

### Теоретические сведения

*Быстрая разработка приложений* RAD (Rapid Application Development) обеспечивает создание на ранней стадии реализации действующей интерактивной модели системы, так называемой системы-прототипа, позволяющей наглядно продемонстрировать пользователю будущую систему, уточнить его требования, оперативно модифицировать интерфейсные элементы [18].

Рассмотрим некоторые приемы работы в RAD-системе Delphi.

### 1. Компоновка приложения и управление проектом

Для приложения, предназначенного для работы с базами данных, условно выделяют три основных этапа его создания [14]:

- разработка структуры базы данных;
- определение функций, выполняемых приложением;
- разработка интерфейса пользователя.


Перечисленные этапы не обязательно должны выполняться в указанной последовательности. Работы первого и второго этапа рассматривались ранее, теперь изучим разработку интерфейса пользователя.

Разработка интерфейса пользователя обычно сводится к разработке форм, окон диалога, системы меню и панелей инструментов. На этом этапе в первую очередь необходимо учесть потребности конечного пользователя и обеспечить простой и интуитивно понятный интерфейс.

#### 1.1. Создание главного меню

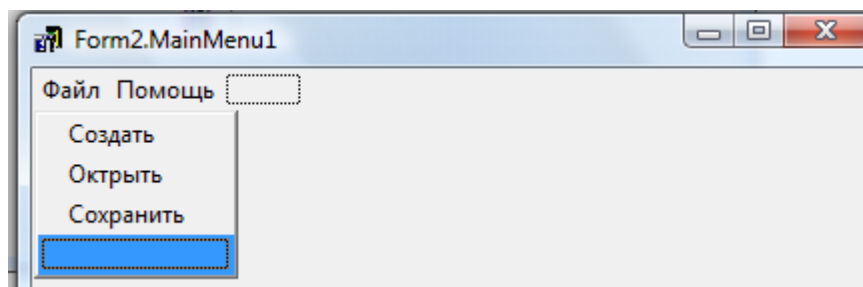
Строка главного меню располагается в верхней части формы приложения. Доступ к ее командам может осуществляться одним из трех способов:

- с помощью мыши;
- с помощью клавиатуры (для входа в главное меню зарезервирована клавиша *Alt*, а для навигации по командам меню используются клавиши перемещения курсора);
- с помощью специальных комбинаций клавиш – клавиатурных сокращений (они могут быть заданы не для всех команд меню).

Для создания главного меню в VCL Delphi имеется специальный компонент **TMainMenu** , расположенный на странице Standard палитры компонентов.

Каждый элемент меню является экземпляром класса **TMenuItem**. Причем объект **TMenuItem** может либо быть командой, либо содержать меню более низкого уровня.

Для создания меню в Delphi используется специальный редактор (Рис. 10.1), который запускается двойным щелчком на компоненте TMainMenu, помещенном на форму, или щелчком на кнопке с многоточием в поле ввода свойства *Items* этого компонента в инспекторе объектов.



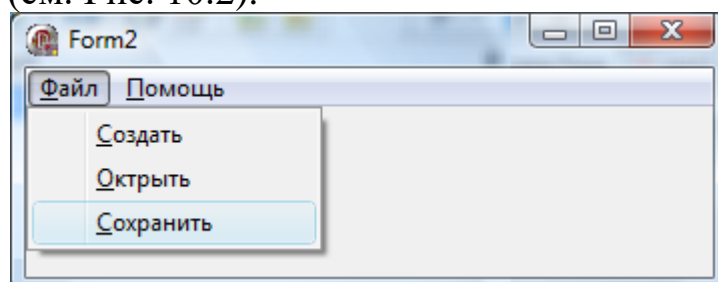
**Рис. 10.1.** Редактор главного меню

### **1.2. Создание и удаление команд меню**

Для добавления новых элементов меню (команд или новых пунктов меню):

- выберите с помощью мыши свободный элемент (такие элементы всегда есть внизу каждого выпадающего меню и в правой части строки меню);
- введите текст команды в поле ввода свойства *Caption* в инспекторе объектов.

При написании текста команды можно использовать служебный символ «&» для назначения командам меню соответствующих им клавиш ускоренного доступа. Клавиша, соответствующая букве, перед которой помещен этот символ, становится клавишей ускоренного доступа. При этом в тексте команды меню эта буква выделяется подчеркиванием (см. Рис. 10.2).



**Рис. 10.2.** Меню с указанием клавиш быстрого доступа

### **1.3. Создание подменю**

Чтобы создать подменю, выполните следующие действия:

- установите указатель мыши на команду меню;
- нажмите правую кнопку мыши;
- выберите команду *Create SubMenu* контекстного меню редактора меню.



#### 1.4. Задание реакции на выбор команды меню

Выбор команд меню во время выполнения программы сопровождается выполнением определенных действий. Для задания этих действий можно задать обработчик события OnClick элемента меню. Создать обработчик события OnClick необходимой команды меню можно, выполнив двойной щелчок на команде в редакторе меню.

При этом Delphi автоматически активизирует окно редактора кода и сгенерирует заголовок процедуры-обработчика события OnClick:


```
procedure TForm2.N3Click(Sender: TObject);  
begin  
end;
```

Для программирования реакции на выбор команды меню достаточно написать соответствующий код и вставить его между словами begin и end процедуры-обработчика.

Например, для отображения окна диалога открытия файла при выборе команды меню «Файл → Открыть» необходимо ввести следующий код:


```
procedure TForm1.N4Click(Sender: TObject);  
begin  
  if OpenFileDialog1.Execute  
  then begin  
    end;  
end.
```

#### 1.5. Создание контекстного меню

В Delphi имеется возможность связывания контекстного меню практически с любым визуальным компонентом. Для этой цели предназначено свойство *PopupMenu*. Процедура создания самого контекстного меню предполагает использование специального компонента – **TPopupMenu** . Технология реализации такого меню практически не отличается от создания обычного меню – используется тот же редактор меню, а свойства и методы класса **TPopupMenu** аналогичны свойствам и методам класса **TMenuItem**, которые уже были рассмотрены ранее [14].

#### 1.6. Панель инструментов

Панели инструментов в последнее время стали таким же привычным элементом интерфейса, как и меню. Во многих случаях панель инструментов является альтернативой меню, обеспечивая более быстрый доступ к командам.

Панели инструментов могут создаваться несколькими способами, например с помощью специального компонента **TToolBar** .

Класс **TToolBar** объединяет в одном объекте сами кнопки и контейнер для них. Каждая кнопка, расположенная на панели инструментов, является экземпляром класса **TToolButton**. Чтобы поместить кнопку на панели инструментов во время разработки программы, следует использовать команду *New Button* контекстного меню компонента **TToolBar**. Размеры всех кнопок, расположенных на панели инструментов, одинаковы и определяются свойствами *ButtonWidth* и *ButtonHeight* компонента **TToolBar**. Кнопки можно группировать, используя разделители, которые помещаются на панель инструментов с помощью команды *New Separator* контекстного меню. После размещения кнопок и разделителей на панели инструментов их можно перемещать с помощью мыши.

Из методов, инкапсулированных в классе **TToolButton**, рассмотрим два, которые используются наиболее часто:

- *function CheckMenuDropdown: Boolean* – отображает выпадающее меню, связанное с кнопкой, и возвращает значение true, если для кнопки задано свойство *DropDownMenu*. Возвращает false, если кнопка не имеет меню;

- *procedure Click* – генерирует событие *OnClick* кнопки. Используется для программного «нажатия» на кнопку.

## 2. Управление проектом и создание приложения

### 2.1. Структура проекта

Разработка нового приложения всегда начинается с создания нового проекта. Создать новый проект приложения можно с помощью одного из следующих способов:

- выберите команду *File* → *New Application* главного меню;
- выберите команду *File* → *New* и затем выберите объект *Application* в открывшемся окне хранилища объектов.

После создания нового приложения автоматически генерируются три объекта: модуль проекта, форма и модуль формы, которые при сохранении записываются в файлы с расширениями *DPR*, *DFM* и *PAS* соответственно. Кроме этих трех основных файлов проект содержит еще три файла, в которых находится служебная информация:

- файлы с расширениями *DOF* и *CFG* содержат сведения о настройках проекта, задаваемых в окне диалога *Project Options*;
- файл с расширением *RES* содержит ресурсы проекта.

### 2.2. Добавление к проекту форм и модулей

Если проект содержит несколько форм, то одна из них является главной – то есть той формой, которая будет отображаться при запуске

приложения. При каждом создании нового проекта приложения автоматически создается одна форма, которая и является главной.

Если проект должен включать несколько форм, то их необходимо добавлять к проекту, используя команду *File* → *New Form* главного меню (или команду *File*→*New* с последующим выбором объекта *Form* в окне хранилища объектов). Главной формой при этом остается та форма, которая была создана первой. Каждой форме приложения обязательно соответствует модуль, который создается автоматически в процессе генерации новой формы.

Проект может также включать в себя программные модули, не связанные с формами. Такие модули обычно являются библиотеками процедур и функций, общих для всего приложения. Для создания нового модуля используйте команду *File* → *New* главного меню и затем выберите объект *Unit* в открывшемся окне хранилища объектов.

Для добавления в проект уже существующих (созданных ранее) модулей и форм используется команда *Project* → *Add to Project*. При выполнении этой команды открывается стандартное окно диалога выбора файла, в котором указывается файл модуля, добавляемого к проекту. Если выбранный модуль связан с формой, то она также добавляется к проекту и включается в список автоматически создаваемых форм.

Для удаления из проекта модулей и/или форм применяется команда *Project* → *Remove from Project*. При ее выполнении отображается окно диалога *Remove from Project*, содержащее список всех форм и модулей, принадлежавших проекту. Чтобы удалить модуль, выделите его в списке и щелкните на кнопке *Ok*.

### ***2.3. Класс TApplication***

Любое приложение, создаваемое в среде Delphi, является экземпляром класса **TApplication**. Данный класс реализует взаимодействие приложения с операционной системой Windows. Переменная, обеспечивающая доступ к свойствам и методам класса **TApplication**, для любого приложения имеет одно и то же имя - *Application*.

## **3. Справочная система приложения**

Традиционно под справочной системой подразумевают *файл справки*, содержащий описание функций программы, который открывается при выборе соответствующего пункта меню или при нажатии на клавишу *F1*. Однако в большинстве случаев этого недостаточно, так как пользователю часто требуется получить оперативную подсказку по программе, а поиск необходимой

информации в текстовом файле может занять много времени. Поэтому наряду с описанием функций программы необходимо наличие дополнительных модулей справки. Это в первую очередь *контекстно-зависимая справка*, позволяющая быстро получать информацию о любом окне программы, диалоге, команде меню или элементе управления во время их использования.

Кроме контекстной справки приложение следует также снабдить системой *всплывающих подсказок*, которые отображаются, когда пользователь задерживает указатель мыши над каким-либо элементом управления. всплывающие подсказки особенно актуальны для кнопок панелей инструментов, так как значки, отображаемые на этих кнопках, часто не дают ясного представления об их назначении.

Полезно также предусмотреть наличие средств, информирующих пользователя о текущем состоянии приложения, о выполняемых им действиях. Это позволит, например, уведомить пользователя о том, что приложение не «зависло», а выполняет какую-либо длительную операцию. Обычно для вывода информация о приложении используется так называемая *строка состояния*, которая располагается в нижней части окна приложения. Таким образом, в наиболее завершённом виде справочная система должна включать в себя следующие элементы:

- файл справки, содержащий подробную информацию о работе с приложением, изложенную в доступной для «среднего» пользователя форме;
- контекстно-зависимую справку, вызываемую нажатием на клавишу F1. В большинстве случаев содержимое контекстно-зависимой справки включается в файл справки;
- систему всплывающих подсказок;
- строку состояния, в которой отображается информация о текущем состоянии приложения.

Реализация первых двух элементов справочной системы осуществляется с помощью специальных программ, предназначенных для разработки файла справки и его просмотра. Delphi в этом случае используется лишь для интеграции приложения с файлами справочной системы.

Последние два элемента справочной системы (всплывающие подсказки и строка состояния) обеспечиваются средствами Delphi. Вопросы разработки файлов справки и интеграции их в приложения Delphi будут рассмотрены ниже, а в первую очередь мы коснемся технологии создания всплывающих подсказок и строки состояния.

### 3.1. Создание всплывающих подсказок

Всплывающие подсказки могут создаваться для любых визуальных компонентов Delphi. Создание подсказки для какого-либо компонента сводится просто к изменению значений некоторых его свойств. Информация о свойствах визуальных компонентов Delphi, отвечающих за отображение «всплывающих» подсказок, приведена в табл. 16.1.

Таблица 16.1

Свойство	Тип	Описание
Hint	string	Текст всплывающей подсказки
ShowHint	Boolean	Определяет, отображать подсказку (true) или нет (false)
ParentShowHint	Boolean	Определяет, наследовать (true) или нет (false) параметры подсказки родительского компонента


Текст подсказки, задаваемый в свойстве *Hint*, может состоять из двух частей, которые разделяются между собой символом «|». Например, текст подсказки может иметь следующий вид:

Печать | Печать текущего документа

В этом случае во всплывающей подсказке отображается только первая часть текста. Вторая часть подсказки (обычно более развернутая) используется для отображения информации в строке состояния приложения.

### 3.2. Создание строки состояния приложения

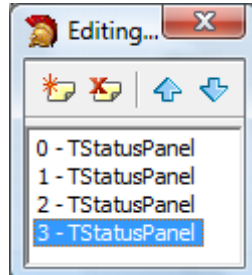
Строка состояния приложения предназначена для того, чтобы информировать пользователя о текущем состоянии базы данных, о процессе выполнения различных операций, а также для вывода различного рода подсказок.

Для создания строки состояния приложения в VCL Delphi существует специальный компонент, имеющий название **TStatusBar** и расположенный на странице Win32 палитры компонентов. Компонент **TStatusBar** , в зависимости от значения свойства *SimplePanel*, может состоять из одной (*SimplePanel* = true) или нескольких панелей (*SimplePanel* = false), на которые выводится какая-либо информация.

В том случае, когда значение свойства *SimplePanel* задано равным true, в строку состояния можно выводить только текстовые сообщения. Для вывода текста в строку состояния используется свойство *SimpleText* класса *TStatusBar*. Например, для вывода в строке состояния количества записей, выбранных в результате выполнения SQL-запроса, можно использовать следующий код:

```
StatusBar1.SimpleText:='Выбрано'+IntToStr(Query1.RecordCount)+'  
записей';
```

При создании сложных строк состояния можно использовать несколько панелей компонента TStatusBar. Для создания панелей строки состояния во время разработки приложения используется специальный редактор панелей (Рис. 10.3).



*Рис. 10.3. Редактор панелей строки состояния*

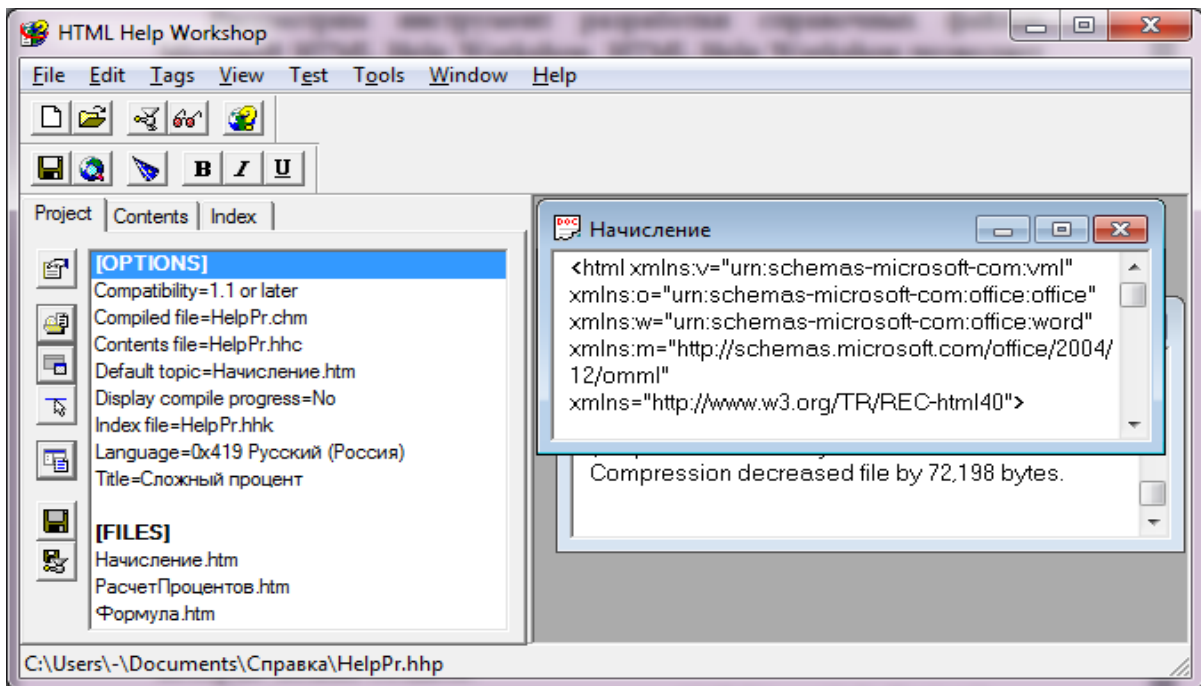
Для его открытия можно воспользоваться одним из следующих способов:

- выполнить двойной щелчок на компоненте TStatusBar, размещенном на форме;
- выбрать пункт Panels Editor контекстного меню данного компонента;
- щелкнуть на кнопке с многоточием в поле ввода свойства Panels компонента TStatusBar в инспекторе объектов.

### ***3.3. Создание справочных файлов***


Рассмотрим инструмент разработки справочных файлов Microsoft HTML Help Workshop. HTML Help Workshop позволяет создавать chm-файл, который содержит в себе набор HTML-страниц, содержание со ссылками на страницы, предметный указатель, а также базу для полнотекстового поиска по содержимому страниц.

На Рис. 10.4 приведено главное окно программы HTML Help Workshop.

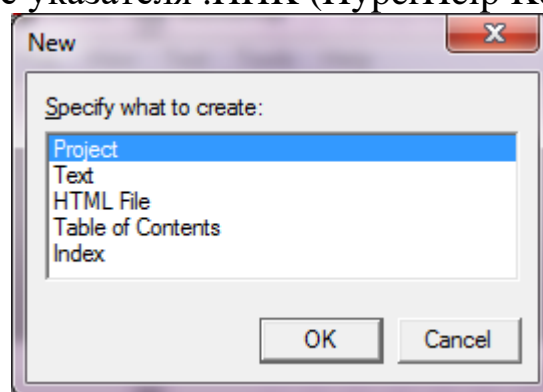


*Рис. 10.4. Окно HTML Help Workshop*

Основные команды представлены кнопками на панели инструментов:


Кнопка  вызывает диалоговое окно создания нового файла, со следующими типами файлов, которые можно создать:

- Project - создание нового проекта .ННР (HyperHelp Project);
- Text - создание текстового файла .ТХТ;
- HTML File - создание гипертекстового файла .НТМ, который будет являться страницей проекта;
- Table of Contents (ТОС) - создание таблицы содержания .ННС (HyperHelp Contents);
- Index - создание указателя .ННК (HyperHelp Keyword).




*Рис. 10.5. Окно выбора создаваемых элементов*

Кнопка  позволяет открыть существующий файл.

Кнопка  запускает компиляцию файла справки.

Кнопка  просмотра файла справки.

Кнопка  вызова справки о программе.

### ***Пример: Начисление процентов по вкладу***

Требуется разработать файл справки для приложения (см. Рис. 10.6), рассчитывающего с помощью формулы сложных процентов начисление процентов по вкладу.

Начисление процентов по вкладу. Формула сложных процентов

Годовая процентная ставка: 12 %

Количество календарных дней в периоде, по итогам которого банк производит капитализацию процентов: 30 дн.

Количество дней в календарном году:  365  366

Первоначальная сумма: 100 руб.

Количество операций по капитализации процентов: 1

Сумма вклада с процентами: 101 руб.

Сумма процентов: 1 руб.

Вычислить Очистить Справка

Нажмите кнопку Вычислить для вычисления суммы вклада с процентами

*Рис. 10.6. Приложение «Начисление процентов по вкладу»*

1. Запустите среду HTML Help Workshop.
2. Создайте новый проект справки, выбрав в меню File команду New. Появится диалоговое окно (рис. 8.5), если выбрать Project, запустится мастер создания проекта. Необходимо указать папку, в которой будет храниться проект файла (Рис. 10.7).

New Project -- Destination

Specify the name of your project file, and where you would like it to be created.

C:\Users\...\Documents\Справка\HelpPr.hhp

Browse...

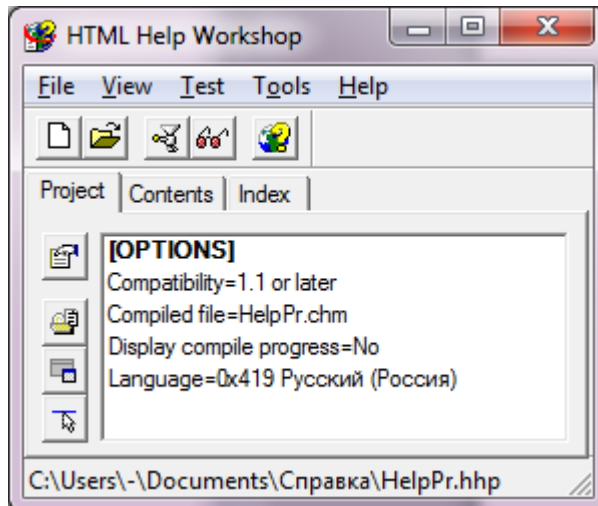
< Назад Далее > Отмена




**Рис. 10.7. Мастер создания проекта справки**

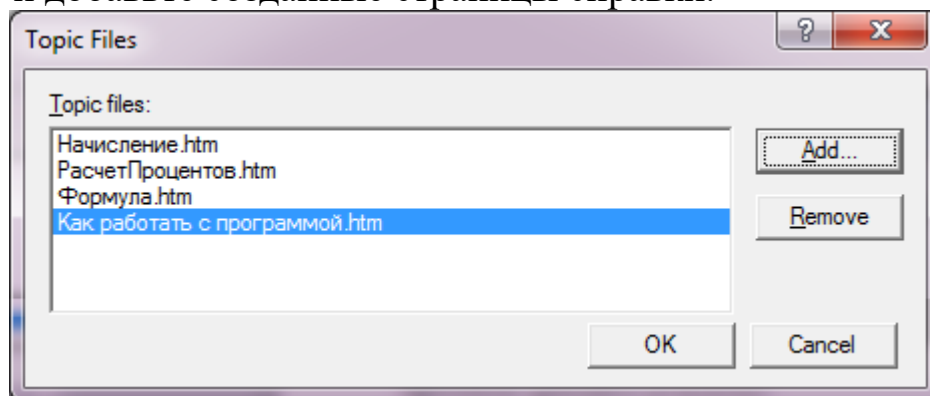
На следующем шаге мастера ничего не меняем, так как еще не созданы файлы, которые можно было бы уже добавить в проект HTML Help Workshop.

В результате работы этого мастера в главном окне программы (см. рис.16.8) появятся три закладки: Проект (Project), Содержание (Contents) и Указатель (Index).




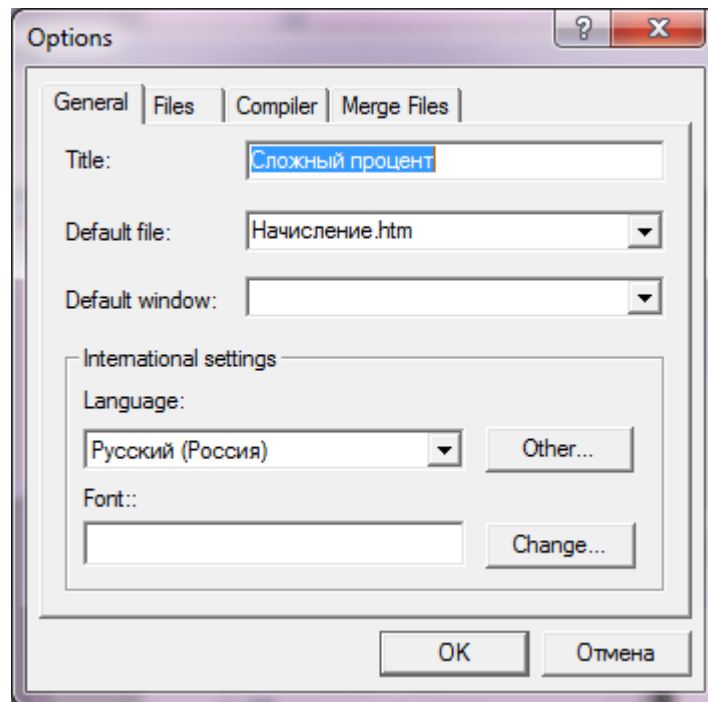
**Рис. 10.8** Закладка «Проект»

3. Теперь нужно подготовить файлы справки. Файлы справки можно создать с помощью встроенного редактора html-страниц, вызываемого командой меню File→New→HTML File, а можно набрать в MS Word и сохранить как веб-страницу в папку с проектом справки. Нажмите далее кнопку  и добавьте созданные страницы справки.




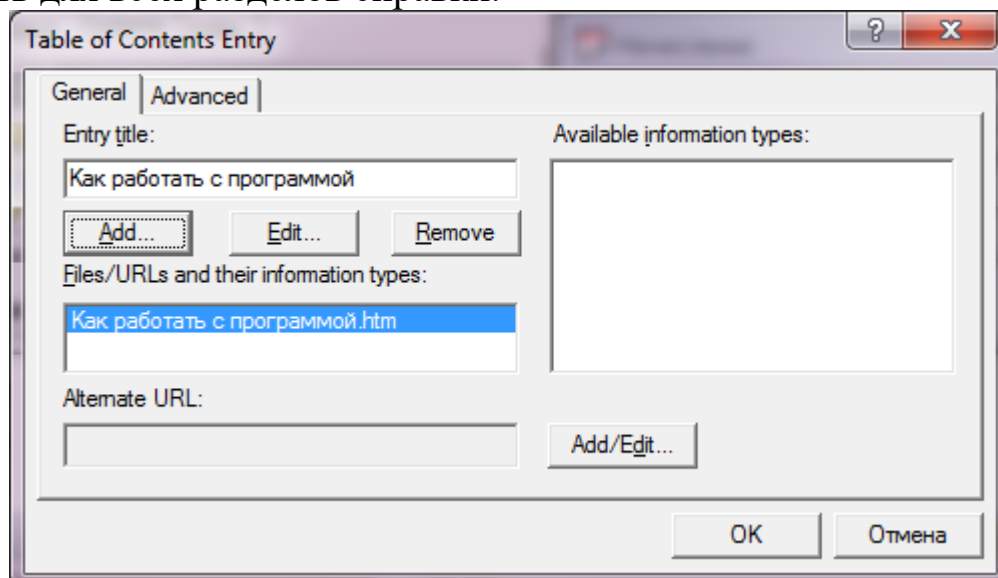
**Рис. 10.9.** Добавленные страницы

4. Нажмите на кнопку  для определения главной страницы справки и в появившемся окне (Рис. 10.10) введите заголовок и имя файла этой страницы.

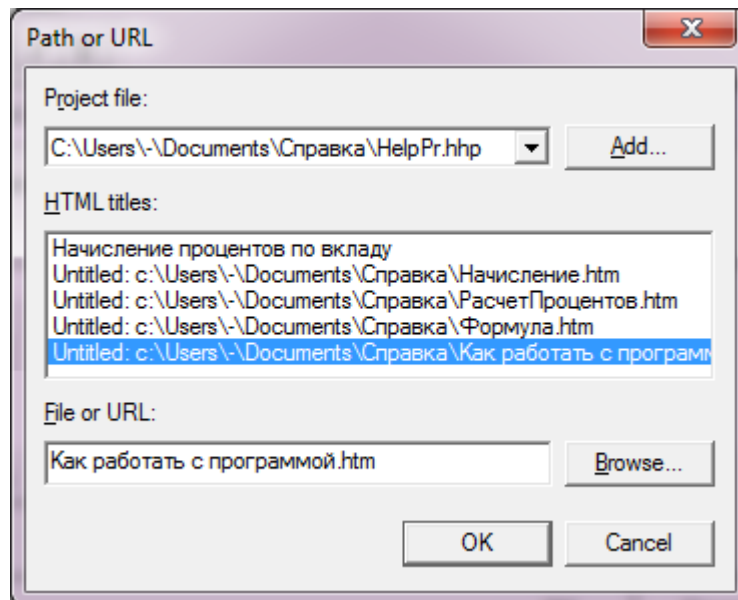


*Рис. 10.10. Добавленные страницы*

5. Для навигации по справке нужно создать файл контента. Перейдите на вкладку Contents, введите имя файла контента и добавьте ссылки на страницы справки, щелкая кнопку . В результате откроется окно Table of Contents Entry, в котором надо ввести название раздела Entry title и добавить (Add) путь к файлу Path or URL (см. Рис. 10.12). Затем нажмите кнопку Ok. Эту последовательность действий надо повторить для всех разделов справки.

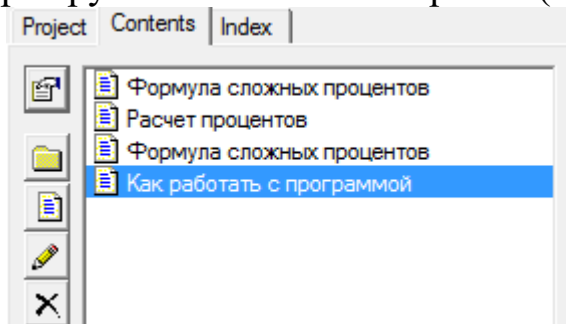


*Рис. 10.11. Создание элемента содержания*




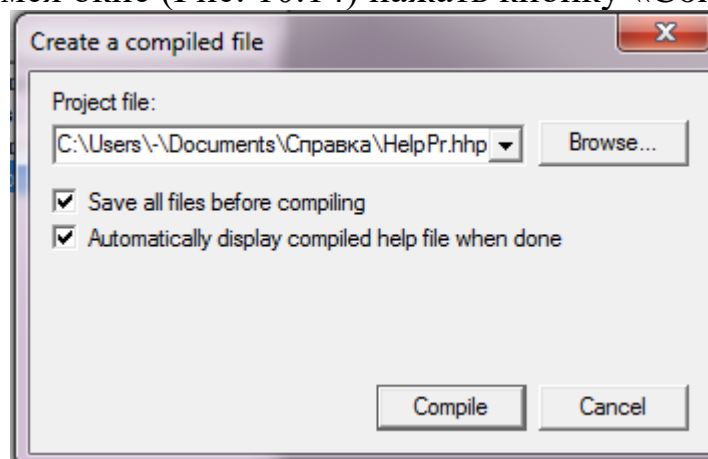
*Рис. 10.12. Определение страницы*

В результате сформируется оглавление справки (см. Рис. 10.13).

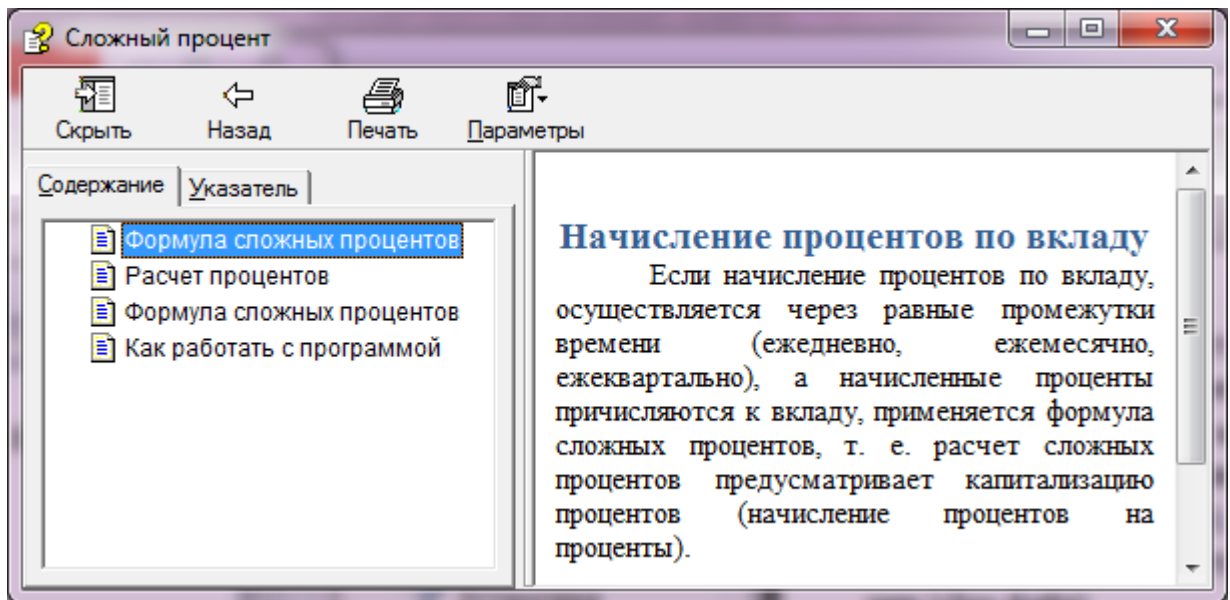


*Рис. 10.13. Оглавление справки*

6. Проведем компиляцию проекта. Для этого надо щелкнуть кнопку  и в появившемся окне (Рис. 10.14) нажать кнопку «Compile».



*Рис. 10.14. Компиляция файла справки*



*Рис. 10.15. chm-файл справки*

7. Теперь нужно запрограммировать кнопку «Справка» на форме приложения (Рис. 10.6), чтобы при её нажатии запускался созданный файл справки.

```
procedure TForm6.Button3Click(Sender: TObject);
```

```
begin
```

```
WinExec('hh.exe HelpPr.chm',sw_show);
```

```
end;
```

8. Запустите приложение и проверьте работу справочной системы.

#### 4. Заставка приложения

Форма-заставка используется при загрузке программ для того, чтобы развлечь пользователя интересной картинкой, сообщить ему полезную информацию о программе, об адресах и сайтах производителя, призвать к регистрации программы.

Добавим заставку к уже созданному приложению «Список публикаций».

#### *Пример: Заставка*

1. Откройте проект этого приложения.

2. Добавьте к проекту новую пустую форму (File → New → VCL Form Delphi), сохраните её на диске назовите под именем, например, frmSplash.

3. У формы-заставки свойство *BorderStyle* нужно установить в *bsNone*, чтобы убрать полосу заголовка вместе с кнопками. Свойству *Position* значение *poDesktopCenter* (окно будет появляться по

центру рабочего стола). Размеры и оформление формы могут быть любыми.

4. Поместите на форму компонент Image, а в него загрузите заранее подготовленную картинку.

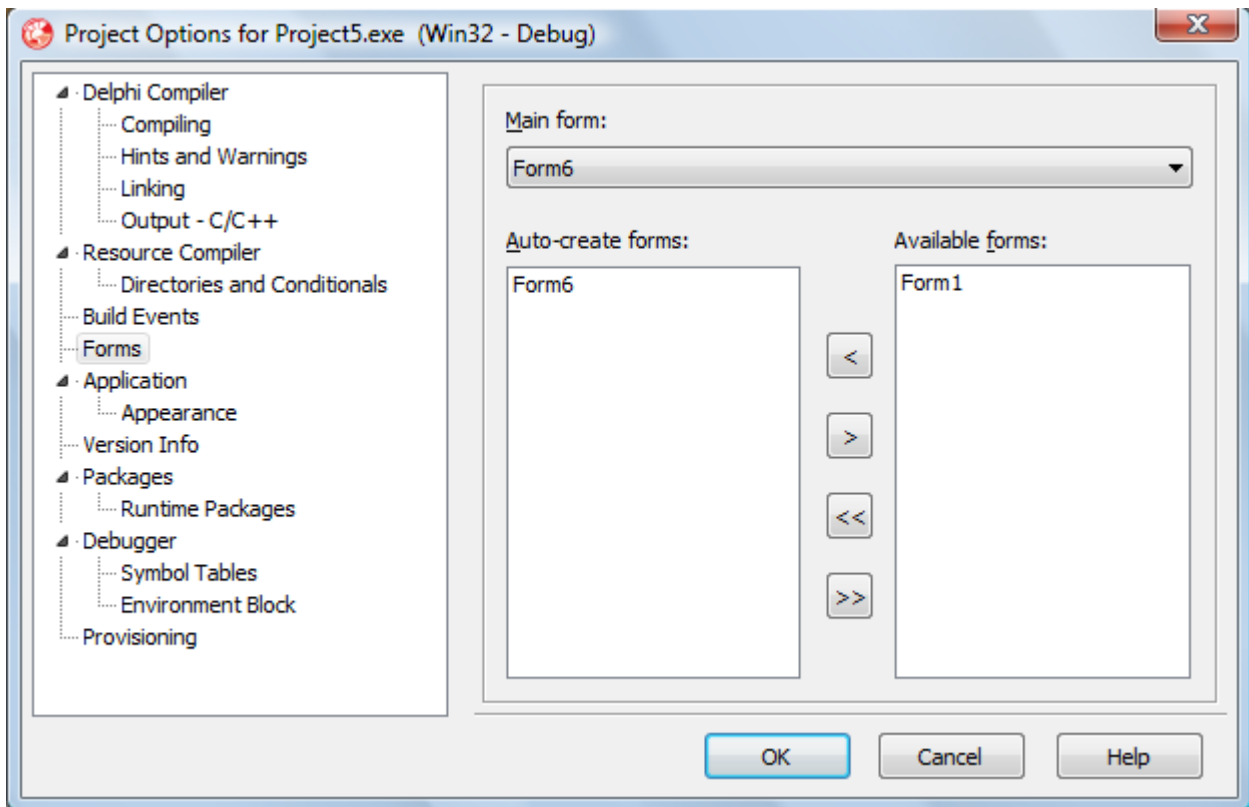


*Рис. 10.16. Окно заставки*

5. Определить срок жизни заставки удобно с помощью компонента Timer, который нужно разместить на форме, установив его свойству *Interval* значение равное 3000 (3 секунды), а в обработчике событий *OnTimer* запишем:

```
Timer1.Enabled := false;
```

6. Форму-заставку нужно создавать самостоятельно. Для этого выполните команду Главного меню Project > Options... и в диалоговом окне ProjectOptions, на странице Forms выделите название формы-заставки в левом списке и перенесите его в правый список AvailableForms. Теперь форма-заставка не будет создаваться автоматически при запуске программы.



*Рис. 10.17. Управление окнами приложения*

7. Выполните команду главного меню Project > ViewSource, файл проекта откроется в окне редактора кода. Допишите в него выделенные строки:

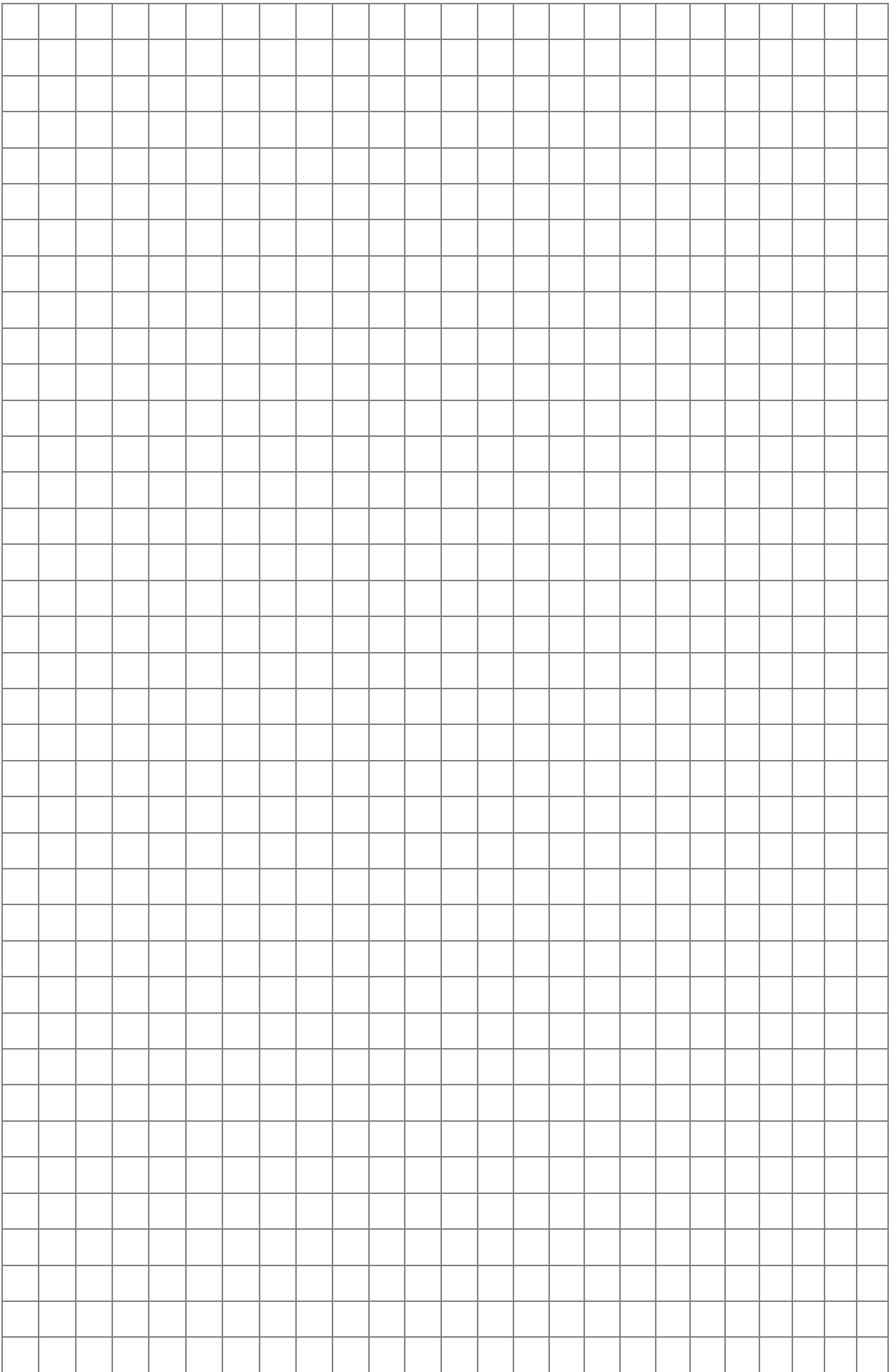
```

program Project5;
uses
  Forms,
  Unit6 in 'Unit6.pas' {Form6},
  frmSplash in 'frmSplash.pas' {Form1};
{$R *.res}
begin
  Application.Initialize;
  Form1:= TForm1.Create(Application);
  Form1.Show;
  Form1.Update;
  while Form1.Timer1.Enabled do Application.ProcessMessages;
  Application.CreateForm(TForm6, Form6);
  Form1.Hide;
  Form1.Free;
  Application.Run;
end.

```

Смысл команд: создаём форму-заставку и выводим на экран, затем стандартные операции по инициализации приложения и, когда всё







# Предметный указатель

<b>A</b>		<b>F</b>	
Activity Diagram .....	116	Form .....	25
ADO .....	77	Frames .....	27
ADOQuery .....	95		
ADOTable .....	77	<b>G</b>	
ALL .....	104	GroupBox .....	30
AND .....	104		
ANY .....	104	<b>H</b>	
<b>B</b>		HHC .....	151
BDE .....	77	HHK .....	151
BETWEEN.AND .....	102	HHP .....	151
BFD .....	60	HTM .....	151
Borland Database Engine .....	77	HTML Help Workshop .....	159
Button .....	4		
<b>C</b>		<b>I</b>	
CFG .....	146	IN 103	
CheckBox .....	5	INSERT INTO .....	106
chm .....	150	IS NULL .....	102
Class diagram .....	116		
Collaboration Diagram .....	116	<b>L</b>	
ComboBox .....	7	Label .....	5
Component Diagram .....	116	LIKE .....	103
Composite Structure Diagram .....	117	ListBox .....	6
<b>D</b>		<b>M</b>	
DataSource .....	78	Memo .....	6
DateTimePicker .....	9	MessageDlg .....	10
DBCheckBox .....	79	MessageDlgPos .....	12
DBComboBox .....	80	MonthCalendar .....	9
DBEdit .....	79		
dbGo .....	77	<b>N</b>	
DBGrid .....	79, 87	NOT .....	105
DBImage .....	80		
DBListBox .....	80	<b>O</b>	
DBMemo .....	79	OpenDialog .....	12
DBNavigator .....	81	OR 104	
DBRadioGroup .....	80	ORDER BY .....	105
DBText .....	79		
DELETE .....	107	<b>P</b>	
Deployment Diagram .....	117	PageControl .....	29, 30
DFD .....	60	PAS .....	146
DFM .....	146	PopupMenu .....	145
DOF .....	146		
DPR .....	146	<b>Q</b>	
<b>E</b>		Query .....	95
Edit .....	6		
Entity .....	45	<b>R</b>	
ERD .....	60	RAD .....	143
ER-модель .....	45	RadioButton .....	5
EXISTS .....	103		

Relation .....	45
Relationship .....	47
RES .....	146

## S

SaveDialog .....	13
SELECT .....	100
Sequence Diagram .....	116
Sequence Role Diagram .....	116
ShowMessage .....	10
SQL .....	95
SSD .....	60
StarUML .....	116
Statechart Diagram .....	116
STD .....	60
StringGrid .....	8

## T

TApplication .....	147
TMainMenu .....	143
TMenuItem .....	143
TPopupMenu .....	145
TStatusBar .....	149
TToolBar .....	145
TToolButton .....	146
TXT .....	151

## U

UML .....	116
UNIQUE .....	103
Unit .....	147
Use case diagram .....	116

## A

Агрегация .....	115
Ассоциация .....	113
Атрибут .....	112

## Б

Быстрая разработка приложений .....	143
-------------------------------------	-----

## В

Всплывающие подсказки .....	148, 149
-----------------------------	----------

## Г

Главное меню .....	143
--------------------	-----

## Д

Диаграмма «сущность-связь» .....	62, 63, 68
Диаграмма иерархии функций .....	63, 65
Диаграмма переходов состояний .....	61, 63, 67
Диаграмма потоков данных .....	61, 63, 65
Диаграмма системной структуры .....	63, 68
Диаграмма структуры программного приложения .....	62
Диаграмма функциональных спецификаций .....	60
Диаграммы классов .....	112

Диалоговое окно .....	10
-----------------------	----

## З

Значение .....	112
----------------	-----

## И

Интерфейса пользователя .....	143
-------------------------------	-----

## К

Клавиша ускоренного доступа .....	144
Класс .....	112
Композиция .....	116
Кратность .....	114, 118

## М

Меню .....	143
Меню контекстное .....	145
Метод .....	113
Многостраничный блокнот .....	29
Модель «сущность-связь» .....	45
Модуль .....	147

## Н

Наследование .....	115
--------------------	-----

## О

Обобщение .....	115
Объект .....	112
Окно .....	25
Оператор отрицания .....	105
Операторы логические .....	102
Операторы объединения .....	104
Операторы сравнения .....	102
Операция .....	113

## П

Панель инструментов .....	145
Подменю .....	144
Прототип .....	143

## Р

Репозиторий .....	63
-------------------	----

## С

Связь .....	45, 47, 113
Справка .....	147
Справка контекстно-зависимая .....	148
Строка состояния .....	148, 149
Сущность .....	45

## Ф

Форма .....	25, 147
Фрейм .....	27

## Литература

1. Архангельский, А.Я. Delphi 7. Справочное пособие / А.Я. Архангельский. – М.: ООО «Бином-Пресс», 2004 г. – 1024 с.
2. Базы данных. Интеллектуальная обработка информации / В.В. Корнеев, А.Ф. Гареев, С.В. Васютин, В.В. Райх. – М.: Нолидж, 2000. – 352 с.
3. Вендеров, А.М. Практикум по проектированию программного обеспечения экономических информационных систем: учеб. пособие / А.М. Вендеров. – М.: Финансы и статистика, 2004. – 192 с.
4. Гвоздева, В.А. Основы построения автоматизированных информационных систем: учебник / А.В. Гвоздева, И.Ю. Лаврентьева. – М.: ФОРУМ, ИНФРА-М, 2007. – 320 с.
5. Гвоздева, Т.В. Проектирование информационных систем / Т.В. Гвоздева. – Р-н-Д: Феникс, 2009. – 508 с.
6. Дарахвелидзе, П.Г. Delphi 2005 для Win32 / П.Г. Дарахвелидзе, Е.П. Марков. – СПб.: БХВ-Петербург, 2005. – 1136 с.
7. Диго, С.М. Базы данных: проектирование и использование: учебник / С.М. Диго. – М.: Финансы и статистика, 2005. – 592 с.
8. Корняков, В.Н. Программирование документов и приложений MS Office в Delphi / В.Н. Корняков. – СПб.: БХВ-Петербург, 2005. – 496 с.
9. Кузин, А.В. Базы данных: Учеб. пособие для студ. высш. учеб. заведений / А. В. Кузин, С.В. Левенисова.-М.: ИЦ " Академия", 2008.-320 с.
10. Кульбицкий, С.А. 1С: Бухгалтерия 7.7. Ведение бухгалтерского и налогового учета / С.А. Кульбицкий. – М.: Триумф, 2009 – 416 с.
11. Маклаков, С.В. Моделирование бизнес-процессов с VFPwin 4.0 / С.В. Маклаков. – М.: ДИАЛОГ-МИФИ, 2002.
12. Мацяшек, Л. А. Анализ требований и проектирование систем. Разработка информационных систем с использованием UML/ Л.А. Мацяшек. – М.: Вильямс, 2002. – 432 с.
13. Мишенин, А.П. Теория экономических информационных систем / А.П. Мишенин. - М.: Финансы и статистика, 2002 – 140 с.
14. Петров, В.Н. Информационные системы: учебник для вузов / В.Н. Петров. – СПб.: Питер, 2003. – 688 с.
15. Проектирование информационных систем: учеб. пособие / В.И. Грекул, Г.Н. Денищенко, Н.Л. Коровкина. – 2-е изд., испр. – С.: Интернет-Университет Информационных технологий; БИНОМ. Лаборатория знаний, 2008. – 300 с.

16. Рамбо, Дж. UML 2.0. Объектно-ориентированное моделирование и разработка / Дж. Рамбо, М. Блаха. – СПб.: Питер, 2007. – 544 с.
17. Сеницын, С.В. Программирование на языке высокого уровня: учебник для студ. высш. учеб. заведений / С.В. Сеницын. – М.: ИД "Академия", 2010. – 400 с.
18. Смирнова, Г.Н. Проектирование экономических информационных систем: учебник / Г.Н. Смирнова, А.А. Сорокин, Ю.Ф. Тельнов. – М.: Финансы и статистика, 2003. – 512 с.
19. Сорокин, А.В. Delphi. Разработка баз данных / А.В. Сорокин. – СПб.: Питер, 2005. – 477 с.
20. Таненбаум, Э. Распределенные системы. Принципы и парадигмы / Э. Таненбаум, М. Стен. – СПб.: Питер, 2003. – 877 с.
21. Тейксейра, С. Delphi 5. Руководство разработчика / С. Тейксейра, К. Пачеко. – М.: Вильямс, 2001. – 832 с.
22. Фельдман, Я. А. Создаем информационные системы / Я.А. Фельдман. – М.: СОЛОН-ПРЕСС, 2006. – 120 с.
23. Филимонова, Е.В. 1С: Предприятие 8.1. Пошаговый самоучитель по бухгалтерскому учету на компьютере. Шаг за шагом / Е.В. Филимонова. – М.: Эксмо, 2009. – 352 с.
24. Фуфаев, Д.Э. Разработка и эксплуатация автоматизированных информационных систем / Д.Э. Фуфаев, Э.В. Фафаев. – М.: Издательский центр «Академия», 2010. – 304 с.
25. Фуфаев, Д.Э. Разработка и эксплуатация удаленных баз данных: Учеб. пособие для студ. высш. учеб. заведений / Д.Э. Фуфаев, Э.В. Фуфаев. – М.: ИЦ "Академия", 2008. – 256 с.
26. Хомоненко, А.Д. Базы данных: учебник для высших учебных заведений / А.Д. Хомоненко, В.М. Цыганков, М.Г. Мальцев. – СПб.: КОРОНА принт, 2002 – 672 с.
27. Хорев, П.Б. Технологии объектно-ориентированного программирования: учеб. пособие для студ. высш. учеб. заведений / П.Б. Хорев. – 2-е изд. – М.: ИЦ "Академия", 2008. – 448 с.
28. Черенков, А.П. Информационные системы для экономистов / А.П. Черенков. – Экзамен, 2003.
29. Юркин, А.Г. Задачник по программированию / А.Г. Юркин. – СПб.: Питер, 2002. – 192 с.

# Содержание

<b>Предисловие</b> .....	<b>3</b>
<b>Тема 1. Компоненты ввода и редактирования данных Delphi</b> .....	<b>4</b>
1. Стандартные элементы интерфейса.....	4
1.1. Кнопки TButton.....	4
1.2. Надписи TLabel.....	5
1.3. Флажки TCheckBox.....	5
1.4. Переключатели TRadioButton.....	5
1.5. Текстовое поле TEdit.....	6
2. Стандартные компоненты Delphi для ввода и редактирования данных.....	6
2.1. Компонент TMemo.....	6
2.2. Списки TListBox.....	6
2.3. Комбинированные поля TComboBox.....	7
2.4. Таблица строк TStringGrid.....	8
2.5. Ввод даты и времени.....	8
3. Диалоговые окна Delphi.....	10
3.1. Диалоговые окна для вывода сообщений в Delphi (ShowMessage, MessageDlg и MessageDlgPos).....	10
3.2. Диалоговые окна выбора имени файла OpenFileDialog и SaveDialog.....	12
Пример: Список публикаций.....	14
Задания для самостоятельного выполнения.....	18
Письменный отчет.....	21
<b>Тема 2. Создание форм для ввода и редактирования данных</b> .....	<b>25</b>
1. Формы в Delphi.....	25
2. Фреймы.....	27
3. Методы работы с элементами управления.....	27
3.1. Размещение и удаление элементов управления.....	27
3.2. Выравнивание компонентов на форме.....	28
3.3. Выделение группы элементов управления.....	28
3.4. Команды выравнивания компонентов.....	28
3.5. Порядок обхода элементов.....	28
4. Компонент TPageControl.....	29
Пример: Счет-фактура.....	29
Задания для самостоятельного выполнения.....	37
Письменный отчет.....	42
<b>Тема 3. Проектирование фактографических баз данных</b> .....	<b>45</b>
1. Концептуальное моделирование структуры данных. Модель «сущность-связь».....	45
2. Дatalogическое проектирование БД.....	48
Пример: Справочник коммерческих банков.....	52
Задания для самостоятельного выполнения.....	53
Письменный отчет.....	55
<b>Тема 4. Функционально-ориентированное проектирование ИС</b> .....	<b>60</b>
1. Технология функционально-ориентированного проектирования ИС.....	60
2. Этапы функционально-ориентированного проектирования ИС.....	63
Пример: Счет-фактура.....	63
Задания для самостоятельного выполнения.....	69
Письменный отчет.....	73
<b>Тема 5. Организация доступа к базам данных из Delphi</b> .....	<b>77</b>
1. Доступ к данным с использованием ADO.....	77
2. Компоненты Delphi для отображения и редактирования данных.....	78
2.1. Класс TDataSource.....	78

2.2. Класс TDBGrid.....	79
2.3. Компоненты для доступа к отдельным полям .....	79
<i>Пример 1: Анкета сотрудника.....</i>	82
<i>Пример 2: Анкета сотрудника (табличная форма) .....</i>	87
<i>Пример 3: Справочник компаний и их сотрудников (связанные таблицы) .....</i>	88
Задания для самостоятельного выполнения.....	90
Письменный отчет .....	91
<b>Тема 6. Выборка данных.....</b>	<b>95</b>
1. Использование SQL для выборки данных из таблицы.....	95
<i>Пример: Поискový запрос.....</i>	96
2. Язык запросов SQL.....	100
2.1. Простейшая форма оператора SELECT.....	100
2.2. Задание условий при выборке данных .....	101
2.3. Упорядочение данных.....	105
2.4. Вставка данных INSERT INTO .....	106
2.4. Удаление данных DELETE FROM.....	107
Задания для самостоятельного выполнения.....	107
Письменный отчет .....	108
<b>Тема 7. Моделирование классов.....</b>	<b>112</b>
1. Концепции объекта и класса.....	112
2. Концепции связи и ассоциации .....	113
3. Обобщение и наследование .....	115
4. Агрегация и композиция .....	115
5. Моделирование классов с помощью StarUML .....	116
Задания для самостоятельного выполнения.....	119
Письменный отчет .....	124
<b>Тема 8. Моделирование состояний.....</b>	<b>125</b>
1. Моделирование состояний.....	125
2. Моделирование состояний с помощью StarUML .....	126
Задание для самостоятельного выполнения.....	128
Письменный отчет .....	131
<b>Тема 9. Моделирование взаимодействий.....</b>	<b>133</b>
1. Варианты использования, диаграммы последовательности, диаграммы деятельности ....	133
<i>Пример: Система подготовки счета на оплату.....</i>	133
2. Моделирование состояний с помощью StarUML .....	136
Задания для самостоятельного выполнения.....	138
Письменный отчет .....	139
<b>Тема 10. RAD-технология разработки приложения.....</b>	<b>143</b>
1. Компоновка приложения и управление проектом.....	143
1.1. Создание главного меню .....	143
1.2. Создание и удаление команд меню.....	144
1.3. Создание подменю.....	144
1.4. Задание реакции на выбор команды меню .....	145
1.5. Создание контекстного меню .....	145
1.6. Панель инструментов .....	145
2. Управление проектом и создание приложения .....	146
2.1. Структура проекта .....	146
2.2. Добавление к проекту форм и модулей .....	146
2.3. Класс TApplication .....	147
3. Справочная система приложения.....	147
3.1. Создание всплывающих подсказок.....	149
3.2. Создание строки состояния приложения.....	149
3.3. Создание справочных файлов.....	150

<i>Пример: Начисление процентов по вкладу</i> .....	152
4. Заставка приложения.....	156
<i>Пример: Заставка</i> .....	156
Задания для самостоятельного выполнения .....	159
Письменный отчет .....	159
<b><i>Предметный указатель</i></b> .....	<b>161</b>
<b><i>Литература</i></b> .....	<b>163</b>
<b><i>Содержание</i></b> .....	<b>165</b>

Узденова Аминат Магомедовна

Рабочая тетрадь  
Проектирование информационных систем  
*Учебно-методическое пособие*

**План университета 2021, поз.**

Редактор Н.В. Ефрюкова

Корректор М.М. Бостанова

Компьютерная вёрстка и набор А.М. Узденова

Подписано в печать

Бумага офисная

Формат 60x84/16.

Объем: 8 уч-изд. л.

Тираж 100 экз.

**Издательство Карачаево-Черкесского  
государственного университета:  
369202, г. Карачаевск, ул. Ленина, 29.  
ЛР №040310 от 21.10.1997.**