



Курсовой проект  
по теме **Система Регистрации Студента**  
по предмету Технологии Конструирования П.С.  
студентов 4-го курса, гр. 4901BD  
Института Транспорта и Связи  
Чижова Юрия и Сытник Александры.  
Вариант №8.

Проверил \_\_\_\_\_

Дата \_\_\_\_\_

Подпись \_\_\_\_\_

2002-01-20, Рига.

## Содержание:

1. Цель работы.....	3
2. Предметная область.....	3
3. Этапы проектирования.....	3
3.1. Этап «Начало».....	3
3.1.1. Итерация №1.....	3
3.1.2. Итерация №2.....	5
3.2. Этап «Развитие».....	6
3.2.1. Итерация №1.....	6
3.2.2. Итерация №2.....	7
3.2.3. Перевод USE-CASE диаграмм в диаграммы последовательности.....	8
3.2.4. Переход от диаграмм последовательности к классам .....	18
4. Диаграммы классов.....	19
5. Оценка качества логической структуры модели с помощью метрики Чидамбера – Кемерера .....	24
6. Компонентная диаграмма .....	25
7. Диаграмма размещения.....	25
8. Процесс написания программы.....	26
9. Выводы.....	33
10. Распечатка программы.....	33
11. Список используемой литературы.....	33

## **1. Цель работы**

Целью данной курсовой работы является разработка программного обеспечения для регистрации студентов в объектно-ориентированной базе данных в соответствии с требованиями, предъявляемыми к унифицированному процессу разработки программного обеспечения. Визуальная разработка производилась в среде Rational Rose 2001, программная реализация (кодирование) в среде программирования Borland Delphi 7.0, связным элементом между двумя средами является мост Ensemble Rose – Delphi Link.

## **2. Предметная область**

В данном проекте требуется разработать программное обеспечение ориентированное для использования на локальном компьютере. В институте осуществляется регистрация всех студентов в объектно-ориентированной базе данных, в которой может храниться следующая информация:

- Фамилия, имя, отчество студента
- Адрес проживания студента
- Телефон студента
- Персональный код студента
- Код группы
- Программа (факультет)
- Номер курса

Разрабатываемая система поддерживает добавление, удаление, редактирование и поиск записей в объектно-ориентированной базе данных.

## **3. Этапы проектирования**

### **3.1. Этап «Начало»**

#### **3.1.1. Итерация №1.**

Определим набор требований, предъявляемых к разрабатываемому продукту, после первой встречи с заказчиком.

Программный продукт представляет собой ООСУБД предназначенную для регистрации студентов в институте. Данная система должна выполнять следующие основные функции:

- Добавление сведений о новом студенте ООБД
- Удаление сведений о студенте из ООБД
- Исправление неверных сведений о студенте в ООБД
- Возможность сохранения данных ООБД
- Просмотр данных ООБД
- Поиск информации в ООБД по определённом набору полей.

Определение внешнего окружения системы.

В соответствии с поставленными требованиями, определим актера, фиксирующего роли внешних объектов, взаимодействующего с системой.

Актер User.

Описание актёра.

Актёр User – описывает пользователя работающего с базой данных, выполняющего редактирование объектно-ориентированной базы данных, добавление и удаление записей из объектно-ориентированной базы данных, поиск в базе данных, сохранение объектно-ориентированной базы данных .

Для актера User определим соответствующие USE-CASE'ы, путем рассмотрения к актера и его взаимодействия с системой. На рисунке 1 представлена Use Case диаграмма построенная на основании требований заказчика.

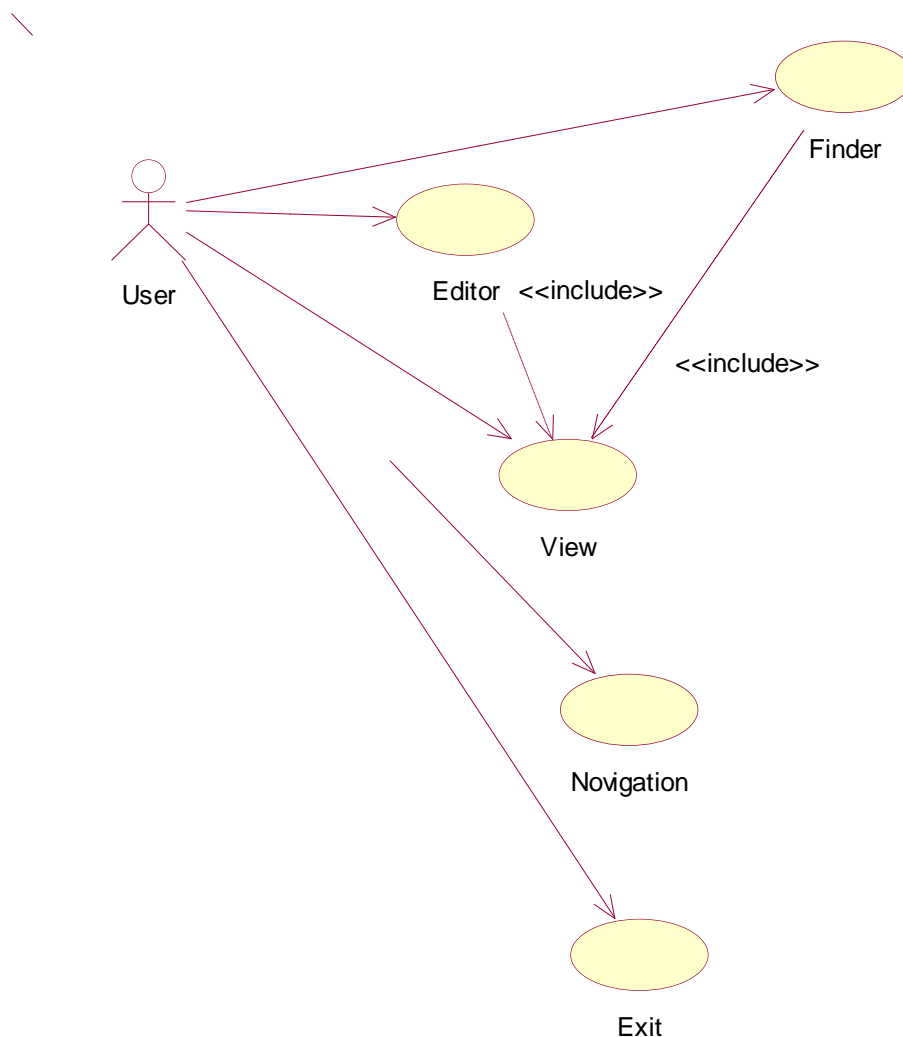


Рис.1

Актёр User инициирует следующие USE-CASE'ы: Finder, Editor, View, Navigation, Exit.

Описание USE-CASE'ов:

- View – функция вывода базы данных на экран;
- Finder – функция поиска записи в базе данных по заданному критерию:
  - Фамилия
  - Имя
  - Персональный код
  - Телефон
  - Группа
  - Факультет
  - Номер курса
- Editor – функция редактирования записей в базе данных
- Exit – функция выхода из программы
- Navigation – функция управления окнами.

### 3.1.2. Итерация №2.

В процессе разработки были уточнены и детализированы некоторые особенности функционирования системы, пересмотрены отношения между Use-Case диаграммами, направление взаимодействия.

Описание актёров.

- Актёр User – описывает пользователя работающего с объектно-ориентированной базой данных на всех уровнях: уровне просмотра полей Базы Данных, формирования запроса к записям Базы Данных.

Описание USE-CASE'ов .

- Exit – функция выхода из программы.
- Finder – функция формирования запроса, поиска записей по запросу и вывод результата на экран:
  - Фамилия
  - Имя
  - Персональный код
  - Телефон
  - Группа
  - Факультет
  - Номер курса

Finder включаемый USE-CASE для базового USE-CASE'а: View.

- Editor – функция редактирования записей в базе данных.  
Editor включаемый USE-CASE для базового USE-CASE'а: View.
- Novigation – функция управления окнами.

View – функция просмотра базы данных.

Окончательный вариант Use-Case диаграммы представлен на рисунке 2.

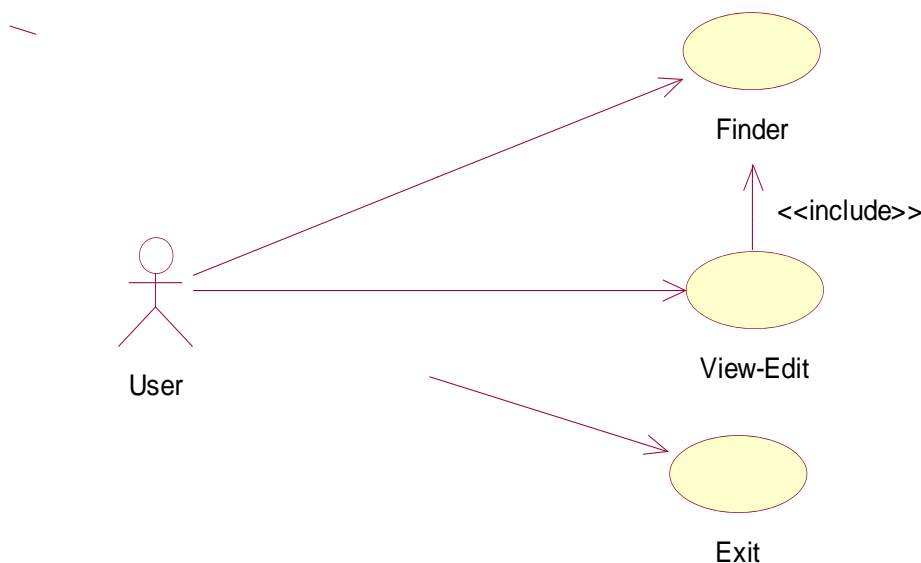


Рис.2

### **3.2. Этап «Развитие»**

#### **3.2.1. Итерация №1.**

- Сценарий для Use-Case'a Exit:  
При нажатии на кнопку Exit на главном окне, главное окно закрывается, пользователь выходит из системы.
- Сценарий для Use-Case'a Finder:  
При нажатии на кнопку Finder на главном окне, главное окно становится недоступным, пользователь попадает в окно DataBase в режиме поиска.
- Сценарий для Use-Case'a View:  
При нажатии на кнопку View на главном окне, главное окно становится недоступным, пользователь попадает в окно DataBase. Пользователь просматривает базу данных с помощью кнопок вперед, назад. При нажатии на кнопку Exit, окно DataBase становится невидимым, пользователь попадает в главное окно.
- Сценарий для Use-Case'a Editor:  
При нажатии на кнопку Editor на главном окне, главное окно становится недоступным, пользователь попадает в окно DataBase. Пользователь выбирает поля для редактирования, в которых необходимо произвести изменения, нажимает на кнопку Exit, появляется окно подтверждения сохранения, затем окно DataBase закрывается, главное окно становится доступным.
- Сценарий для Use-Case'a Navigator:  
При нажатии на любую кнопку в главном окне происходит переключение окон. Главное окно становится недоступным, окно DataBase становится доступным. Всегда вызывается окно DataBase, но в зависимости от нажатой кнопки на главном меню, окно DataBase вызывается с разными параметрами.

### 3.2.2. Итерация №2.

- Развитие описания Use-Case'a Finder:  
Действие начинается с главного окна, когда пользователь нажимает на кнопку Finder. Главное окно становится недоступным, окно DataBase становится доступным. Пользователь заполняет поля, по которым будет производиться поиск. При нажатии на кнопку Exit окно DataBase закрывается, главное окно становится доступным.
- Сценарий для Use-Case'a Finder:  
Действие начинается с главного окна, когда пользователь нажимает на кнопку Finder. Главное окно становится недоступным, окно DataBase становится доступным. Окно DataBase появляется с новыми свойствами, присущими ему, как "окну Finder". Пользователь выбирает поля и ключи, по которым производиться поиск. При нажатии на кнопку Exit окно DataBase закрывается, главное окно становится доступным.
- Сценарий для Use-Case'a View:  
Действие начинается с главного окна, пользователь нажимает на кнопку View. Главное окно становится недоступным, окно DataBase становится видимым. Окно DataBase появляется с новыми свойствами, присущими ему, как "окну View". Пользователь находится в режиме просмотра, указатель устанавливается на первую запись в базе данных. При нажатии на кнопку вперед, назад пользователь продвигается по объектам(записям) базы данных. При нажатии на кнопку Exit окно DataBase закрывается, главное окно становится доступным.
- Развитие описания Use-Case'a Editor:  
При нажатии на кнопку Editor на главном окне, главное окно становится недоступным, пользователь попадает в окно DataBase. Пользователь выбирает нужную запись с помощью поиска или самостоятельно, передвигаясь по объектам (записям) с помощью кнопок вперед и назад, выбирает поля для редактирования, в которых необходимо произвести изменения, устанавливает курсор и делает необходимые изменения, нажимает на кнопку Close, появляется окно подтверждения сохранения, затем окно DataBase закрывается, главное окно становится доступным.
- Сценарий для Use-Case'a Editor:  
При нажатии на кнопку Editor на главном окне, главное окно становится недоступным, пользователь попадает в окно DataBase. Пользователь выбирает нужную запись в окне DataBase с помощью поиска или передвигаясь по объектам (записям) с помощью кнопок вперед, назад, выбирает поля для редактирования, в которых необходимо произвести изменения, устанавливает курсор и делает необходимые изменения. Для добавления записи в БД надо нажать на кнопку Add, для удаления записи из БД, надо нажать на кнопку Delete. Затем нажимаем на кнопку Exit, появляется окно подтверждения сохранения, при нажатии на кнопке Да, сделанные изменения сохраняются, при нажатии на кнопке Нет, изменения не сохраняются. Затем окно DataBase закрывается, главное окно становится доступным.

- Развитие сценария для Use-Case'a Navigation:  
При нажатии на определенную кнопку в главном окне происходит переключение окон. Главное окно становится недоступным, окно DataBase становится доступным.  
Всегда вызывается окно DataBase, но в зависимости от нажатой кнопки на главном меню, окно DataBase вызывается с разными параметрами.
- Сценарий для Use-Case'a Novigation:  
При нажатии на кнопку View, в главном окне происходит переключение окон, главное окно становится недоступным, окно DataBase становится доступным с определенными свойствами (видимыми становятся кнопки: First, Prev, Next, Last, Exit).  
При нажатии на кнопку Editor в главном окне происходит переключение окон, главное окно становится недоступным, окно DataBase становится доступным с определенными свойствами (видимыми становятся кнопки: First, Prev, Next, Last, Add, Delete, Exit).  
При нажатии на кнопку Finder в главном окне происходит переключение окон, главное окно становится недоступным, окно DataBase становится доступным с определенными свойствами (видимыми становятся кнопки: FindForward, FindBackward, Exit).  
При нажатии на кнопку Exit в главном окне происходит переключение окон, главное окно закрывается, выход из программы.

На следующем шаге сценарии Use-Case'ов преобразуются в диаграммы последовательности.

### 3.2.3. Перевод USE-CASE диаграмм в диаграммы последовательности

Для перевода, вернёмся на шаг назад и в сценариях для каждого USE-CASE'a подчеркнем глаголы двойным подчёркиванием, существительные одинарным подчёркиванием, а свойства волнистой чертой. Сопоставим глаголы с методами, существительные с объектами, свойства со свойствами, присущими конкретным объектам:

- Превращение сценария для Use-Case'a Exit в диаграмму последовательности ExitUser:  

При <u>нажатии на кнопку Exit</u>	→	ExitButtonClick
<u>главное окно</u>	→	MainForm
<u>закрывается</u>	→	CloseMainForm



Диаграмма последовательности приведена на рисунке 3.

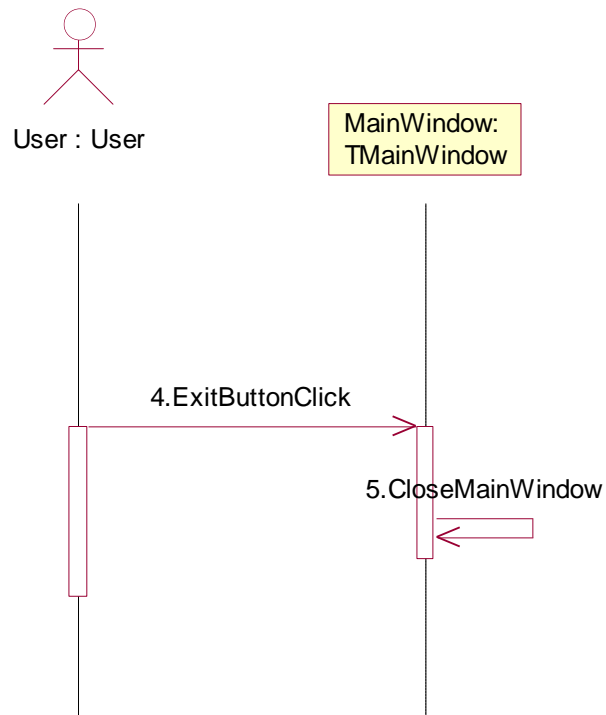


Рис.3

Превращение сценария для Use-Case'а Finder в диаграмму последовательности FinderInDB:

<u>с главного окна</u>	→ MainWindow
<u>нажимает на кнопку Finder</u>	→ FinderButtonClick
<u>Главное окно</u>	→ MainWindow
<u>окно DataBase</u>	→ DataBase
<u>становиться доступным.</u>	→ ShowDataBase
<u>Окно DataBase</u>	→ DataBase
<u>При нажатии на кнопку Exit</u>	→ ExitButtonClick
<u>окно DataBase</u>	→ DataBase
<u>закрывается</u>	→ CloseDatABase

Диаграмма последовательности приведена на рисунке 4.

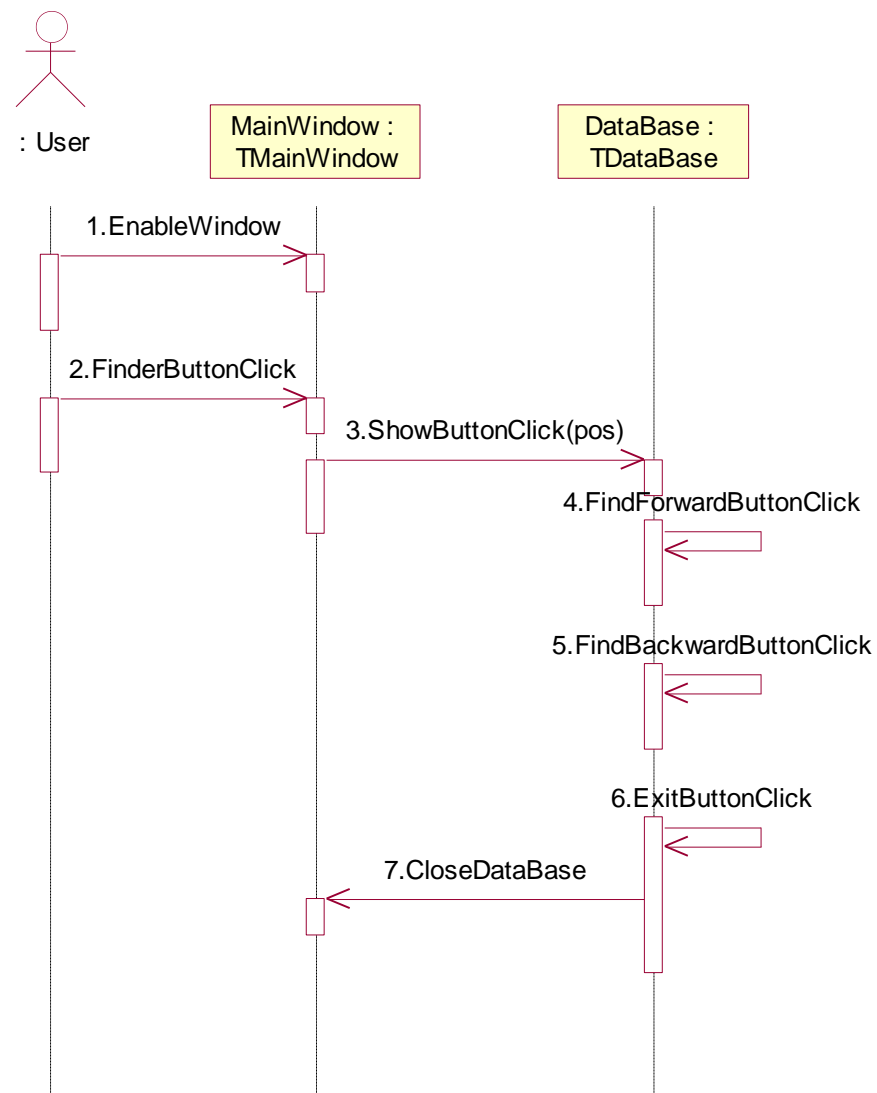


Рис.4

Превращение сценария для Use-Case'а View в диаграмму последовательности ViewDB:

<u>начинается</u>	→ EnableWindow
<u>с главного окна</u>	→ MainWindow
<u>нажимает на кнопку View</u>	→ ViewButtonClick
<u>Главное окно</u>	→ MainWindow
<u>окно DataBase</u>	→ DataBase
<u>становиться видимым</u>	→ ShowDataBase
<u>Окно DataBase</u>	→ DataBase
<u>появляется с новыми свойствами</u>	

При нажатии на кнопку вперед, назад → NextButtonClick, PrevButtonClick  
При нажатии на кнопку Exit → NoButtonClick  
окно DataBase → DataBase  
закрывается, → CloseDataBase  
главное окно → MainWindow  
становиться доступным → EnableWindow

Диаграмма последовательности приведена на рисунке 5.

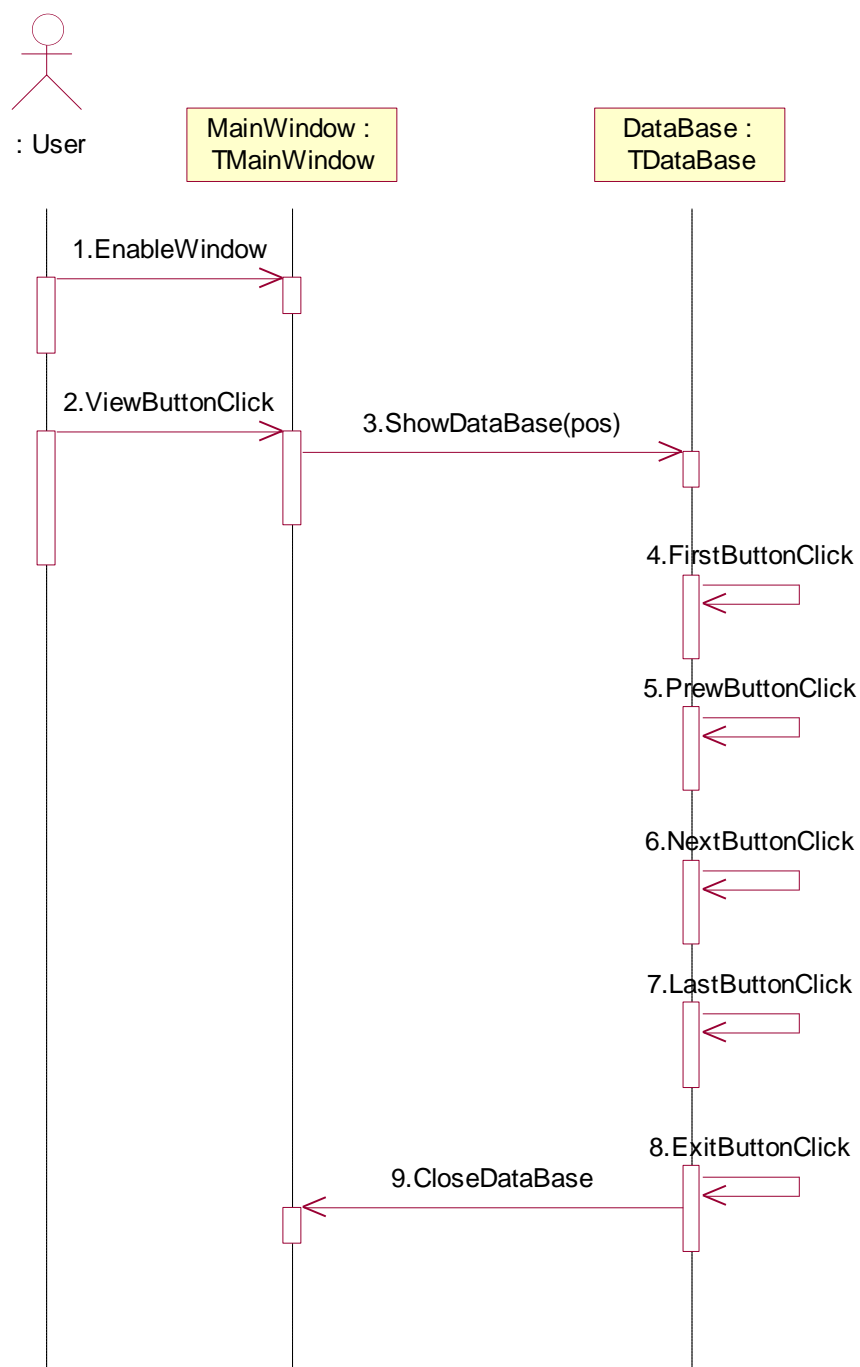


Рис.5

Превращение сценария для Use-Case'a Editor в диаграмму последовательности EditDB:

<u>При нажатии на кнопку Editor</u>	→ EditButtonClick
<u>главное окно</u>	→ MainWindow
<u>окно DataBase</u>	→ DataBase
<u>передвигаясь с помощью кнопок вперед, назад</u>	→ NextButtonClick, PrevButtonClick
<u>нажатие на кнопку Add</u>	→ AddButtonClick
<u>нажатие на кнопку Delete</u>	→ DeleteButtonClick
<u>нажимает на кнопку Exit</u>	→ ExitButtonClick
<u>появляется</u>	→ ShowConfWindow
<u>окно подтверждения сохранения</u>	→ ConfirmationWindow
<u>нажатии на кнопке Да</u>	→ YesButtonClick
<u>нажатии на кнопке Нет</u>	→ NoButtonClick
<u>окно DataBase</u>	→ DataBase
<u>закрывается</u>	→ CloseConfWindow
<u>главное окно</u>	→ MainWindow
<u>становиться доступным</u>	→ EnableWindow

Диаграмма последовательности приведена на рисунке 6.



Рис.6

Превращение сценария для Use-Case'a Novigation в диаграмму последовательности NovigationDB:

<u>При нажатии на кнопку View</u>	→ ViewButtonClick
<u>в главном окне</u>	→ MainWindow
<u>становиться недоступным</u>	→ NoEnableWindow
<u>окно DataBase</u>	→ DataBase
<u>становиться доступным</u>	→ ShowDataBase
с определенными свойствами (видимые кнопки: <u>First, Prev, Next, Last</u> )	
<u>При нажатии на кнопку Editor</u>	→ EditorButtonClick
<u>главное окно</u>	→ MainWindow
<u>становиться недоступным</u>	→ NoEnableWindow
<u>окно DataBase</u>	→ DataBase
<u>становиться доступным</u>	→ EnableWindow
с определенными свойствами (видимые кнопки: <u>First, Prev, Next, Last, Add, Delete, Exit</u> )	
<u>При нажатии на кнопку Finder</u>	→ FinderButtonClick
<u>главное окно</u>	→ MainWindow
<u>становиться недоступным</u>	→ NoEnableWindow
<u>окно DataBase</u>	→ DataBase
<u>становиться доступным</u>	→ EnableWindow
с определенными свойствами (видимые кнопки: <u>FindForward, FindBackward, Exit</u> )	
<u>При нажатии на кнопку Exit</u>	→ ExitButtonClick
<u>главное окно</u>	→ MainWindow
<u>закрывается,</u>	→ CloseMainWindow
<u>выход из программы</u>	→ CloseProgram

Диаграмма последовательности приведена на рисунке 7.

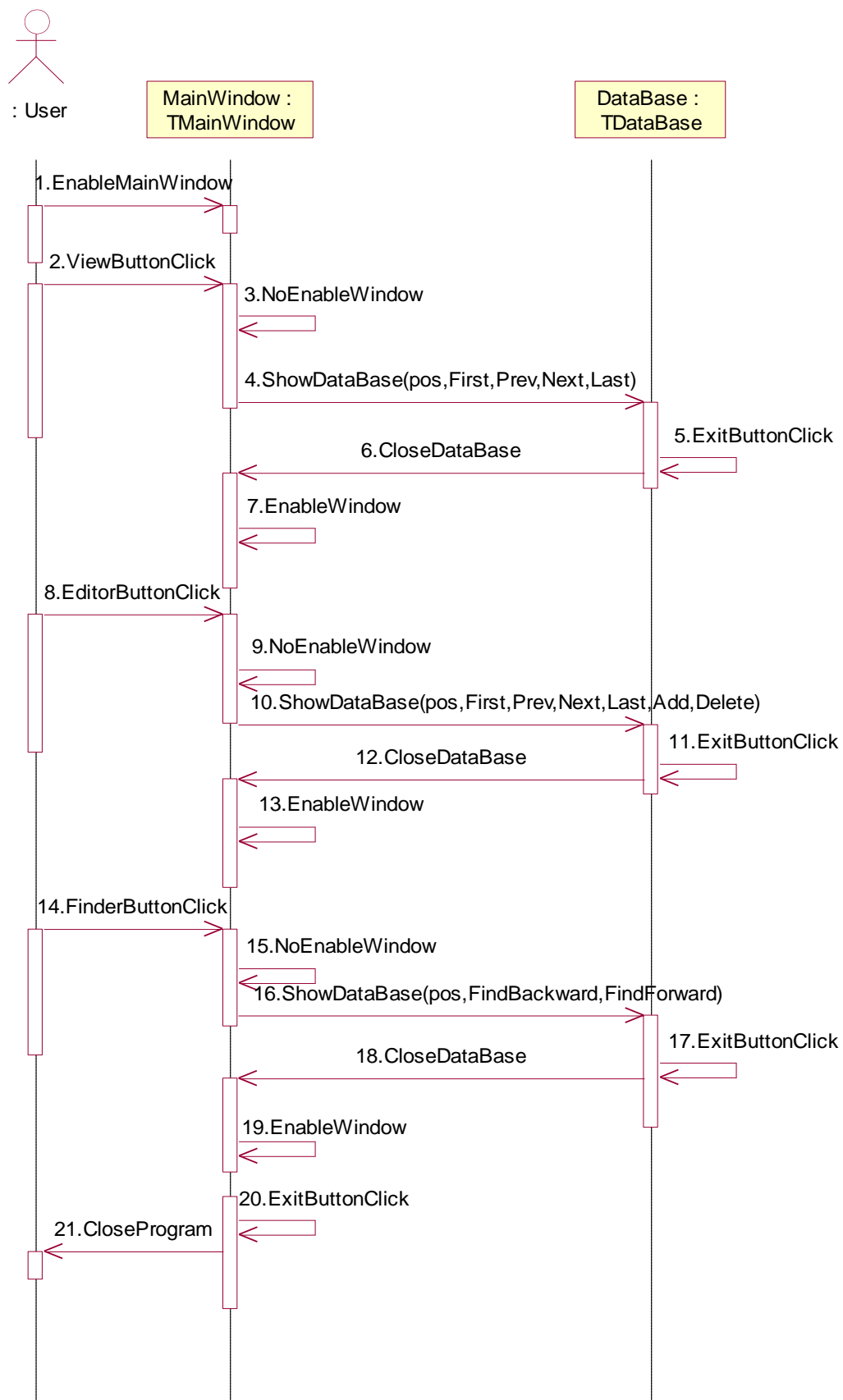


Рис.7

При очередной встрече с заказчиком, проанализировав еще раз требования к программному обеспечению, выяснилось, что программный продукт можно упростить. Для этого надо вернуться к пункту **3.1. Этапа «Начало»** и **3.1.2. Итерации №2.**

Упрощение разработки можно свести к объединению USE-CASE'ов View и Editor.

Описание актёров.

- Актёр User – описывает пользователя работающего с объектно-ориентированной базой данных на всех уровнях: уровне просмотра полей базы данных, формирования запроса к записям базы данных, сохранения базы данных.

Описание USE-CASE'ов .

- Exit – функция выхода из программы.
- Finder – функция формирования запроса, поиска записей по запросу и вывод результата на экран:
  - Фамилия
  - Имя
  - Персональный код
  - Телефон
  - Группа
  - Факультет
  - Номер курса

Finder включаемый USE-CASE для базового USE-CASE'a: View.

- View-Edit – функция просмотра и редактирования базы данных.

Окончательный вариант Use-Case диаграммы представлен на рисунке 8.

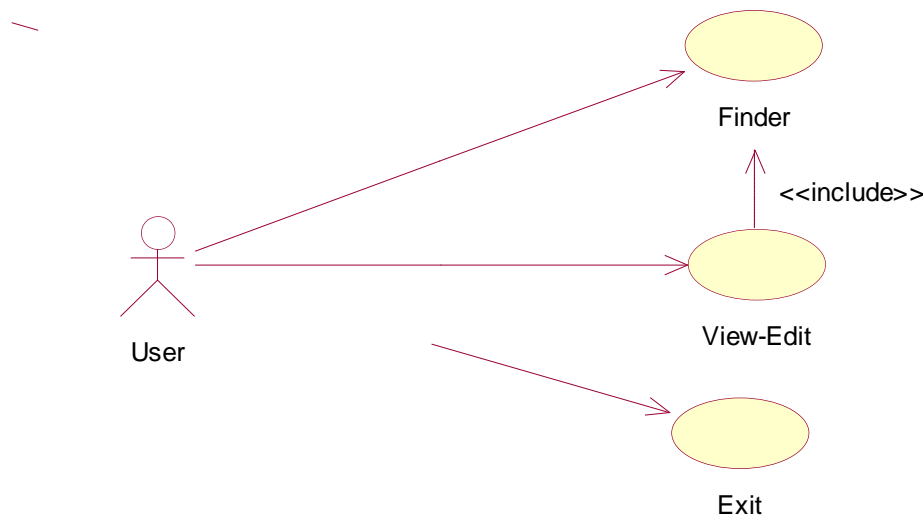


рис. 8.

Сценарии и диаграммы последовательностей для USE-CASE Exit остаётся без изменений после Итерации №2.

Сценарий и диаграмма последовательности для USE-CASE'a Editor объединяется со сценарием и диаграммой последовательности USE-CASE'a View, в результате объединения получается один USE-CASE View-Edit.

Диаграмма последовательности для ViewEditDB приведена на рисунке 9.





рис. 9.

Окончательная диаграмма последовательности для Finder приведена на рисунке 10.

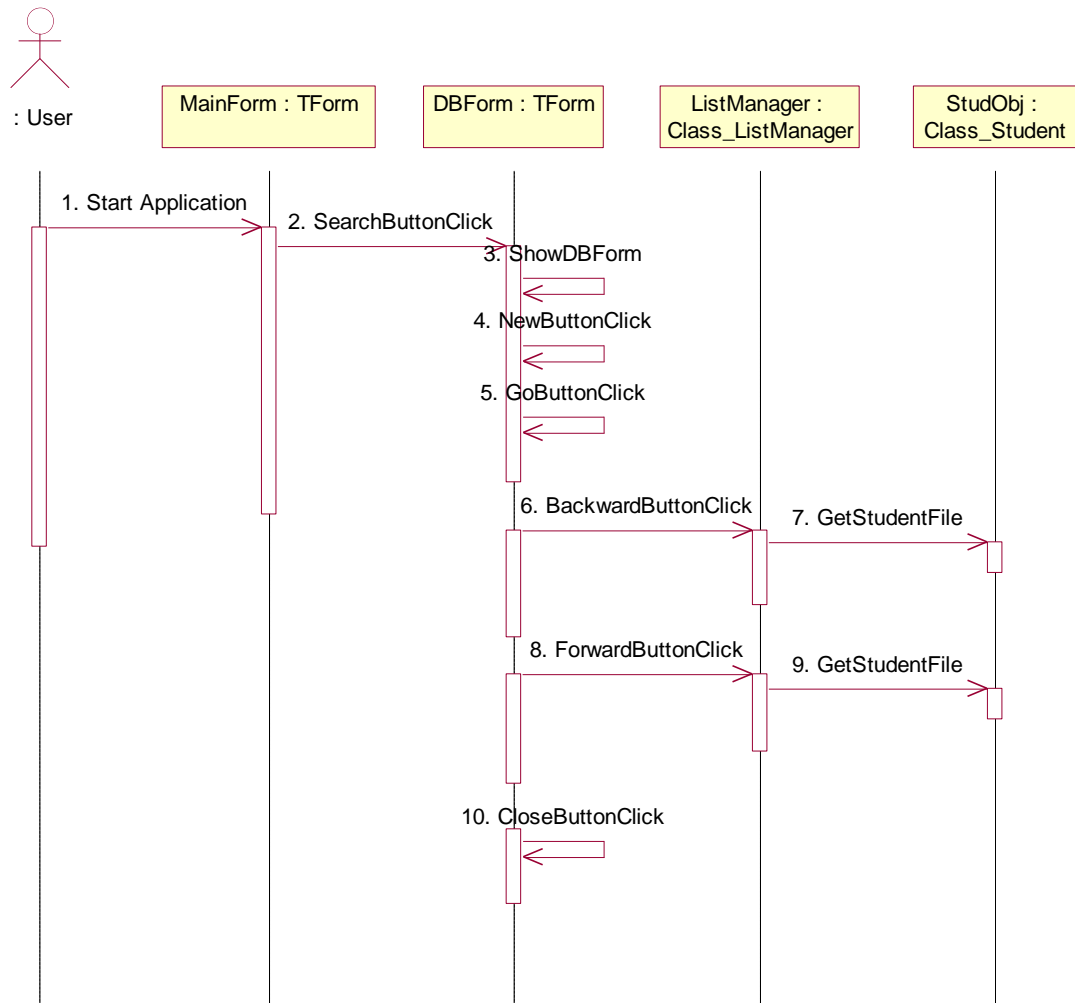


рис.10.

### 3.2.4. Переход от диаграмм последовательности к классам

Объекты из диаграмм последовательности группируются в классы. Основываясь на наших диаграммах последовательности, мы можем идентифицировать следующие объекты и классы:

- MainForm является объектом класса TForm
- ConfirmForm является объектом класса TForm
- DataBaseForm является объектом класса TForm

Т.к. мы работаем с окнами, введем класс TForm – класс-родитель для всех окон.

Сообщения из диаграмм последовательности преобразуются в методы класса. Стрелка с сообщением, которая входит в линию жизни объекта, преобразуется в метод класса, которому принадлежит объект.

#### 4. Диаграммы классов.

Диаграмма классов для модуля **Confirm** приведена на рисунке 11 (в сокращённом виде). Модуль Confirm предоставляет пользователю сохранить изменения или отказаться от сохранения при выходе из программы, либо вернуться в диалог редактирования БД.

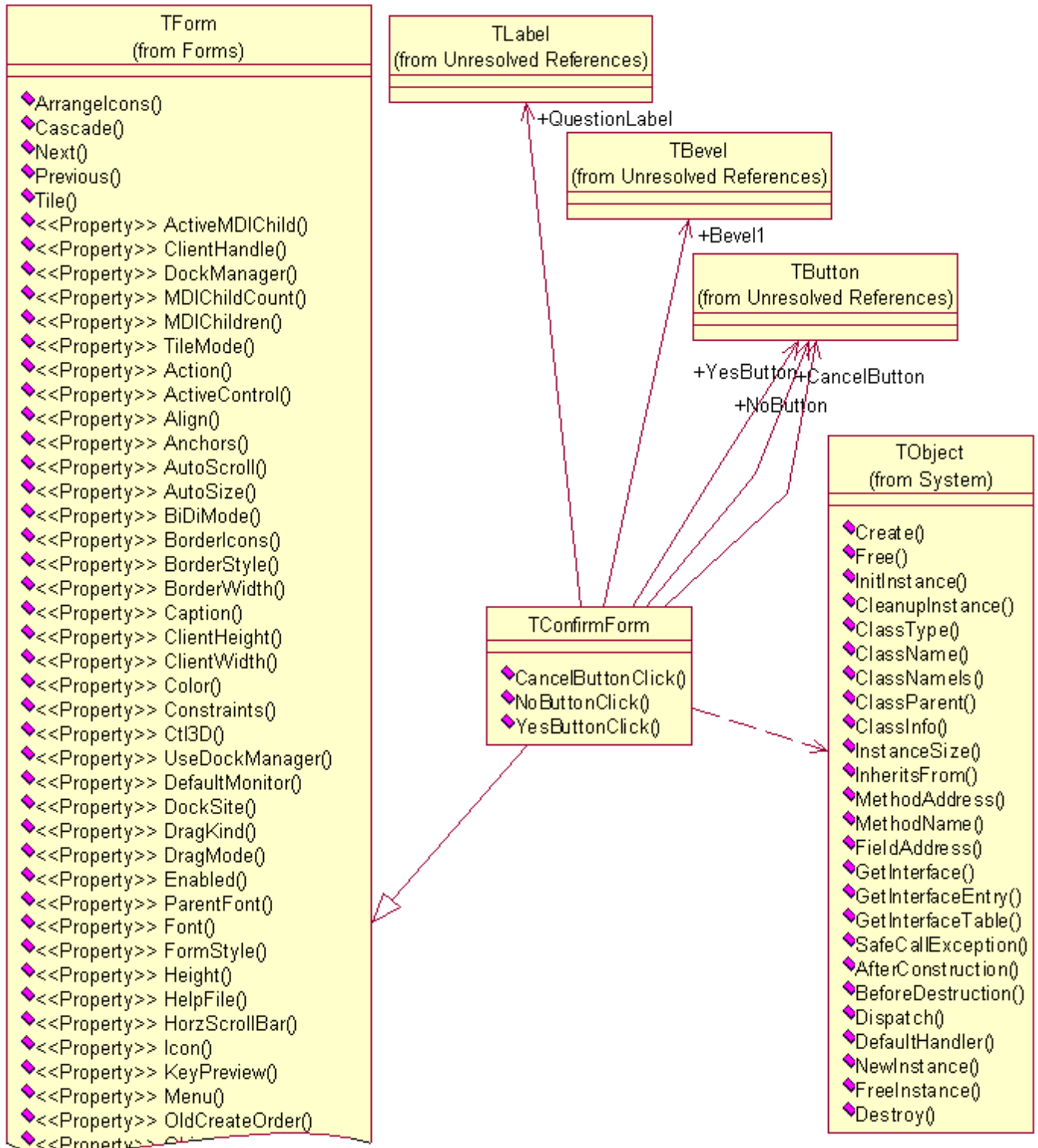


Рис. 11.

**CancelButtonClick()** – метод реализующий отказ от прекращения редактирования БД и возврат пользователя в диалог редактирования БД.

NoButtonClick() – метод позволяет пользователю отказаться от сохранения изменений и перейти в главное меню программы.

YesButtonClik() – метод позволяет сохранить изменения в БД на диск и переводит пользователя в главное меню.

Диаграмма классов для модуля **ДБ** приведена на рисунке 12 (в сокращённом виде).

DBForm – универсальное окно предоставляющее возможность, в зависимости от параметров, использовать его как поисковое средство, либо как редактор с просмотром.

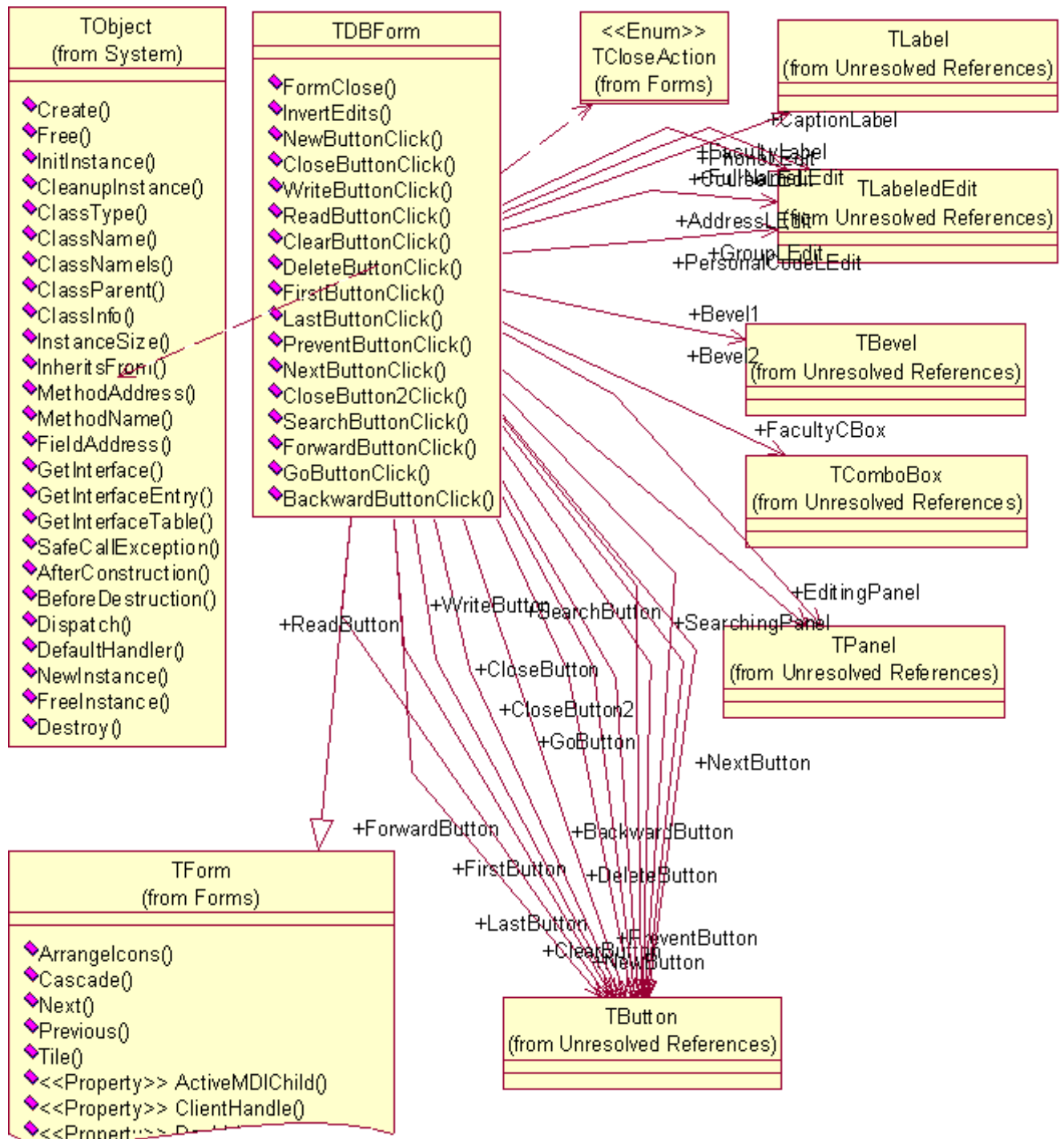


Рис. 12.

FormClose() – метод корректного закрытия формы.

InvertEdits() – инвертирует значения параметров Enable всех контролей ввода/вывода информации. Используется в поисковом режиме окна.

NewButtonClick() – обработчик нажатия на кнопку NewButton. Обращается к ListManager с целью создания в БД нового объекта.

CloseButtonClick() - обработчик нажатия на кнопку CloseButton. Используется для закрытия окна в режиме редактирования. Перед непосредственным закрытием выводит окно ConfirmForm.

WriteButtonClick - обработчик нажатия на кнопку WriteButton. Производит запись содержимого полей ввода/вывода в БД (в память).

ReadButtonClick - обработчик нажатия на кнопку ReadButton. Производит чтение содержимого БД и заполняет им поля пользовательского интерфейса.

ClearButtonClick() - обработчик нажатия на кнопку ClearButton. Очищает текстовые значения полей ввода/вывода. При этом содержимое БД остаётся неизменным.

DeleteButtonClick() - обработчик нажатия на кнопку DeleteButton. Производит удаление текущего(того, чьи данные отображены пользователю на экране) объекта из БД.

FirstButtonClick() - обработчик нажатия на кнопку FirstButton. Перемещает текущую позицию на начало списка. Обновляет поля ввода/вывода.

LastButtonClick() - обработчик нажатия на кнопку LastButton. Перемещает текущую позицию в конец списка. Обновляет поля ввода/вывода.

PrevButtonClick() - обработчик нажатия на кнопку PrevButton. Перемещает текущую позицию на предыдущий элемент списка (объект). Обновляет поля ввода/вывода.

NextButtonClick() - обработчик нажатия на кнопку NextButton. Перемещает текущую позицию на следующий элемент списка (объект). Обновляет поля ввода/вывода.

CloseButton2Click() - обработчик нажатия на кнопку CloseButton2. Используется для закрытия окна в режиме поиска.

SearchButtonClick() - обработчик нажатия на кнопку NewSearchButton. Задаёт новую сессию поиска. Предоставляет возможность заполнить поля поиска.

GoButtonClick() - обработчик нажатия на кнопку GoButton. Фиксирует поля поиска.

ForwardButtonClick() - обработчик нажатия на кнопку ForwardButton. Производит поиск до первого попавшегося элемента вперёд.

BackwardButtonClick() - обработчик нажатия на кнопку BackwardButton. Производит поиск до первого попавшегося элемента в обратном направлении.

Диаграмма классов для модуля **Main** приведена на рисунке 13а и 13б (в сокращённом виде). Диаграмма разбита на две независимые части и приведена на двух рисунках.

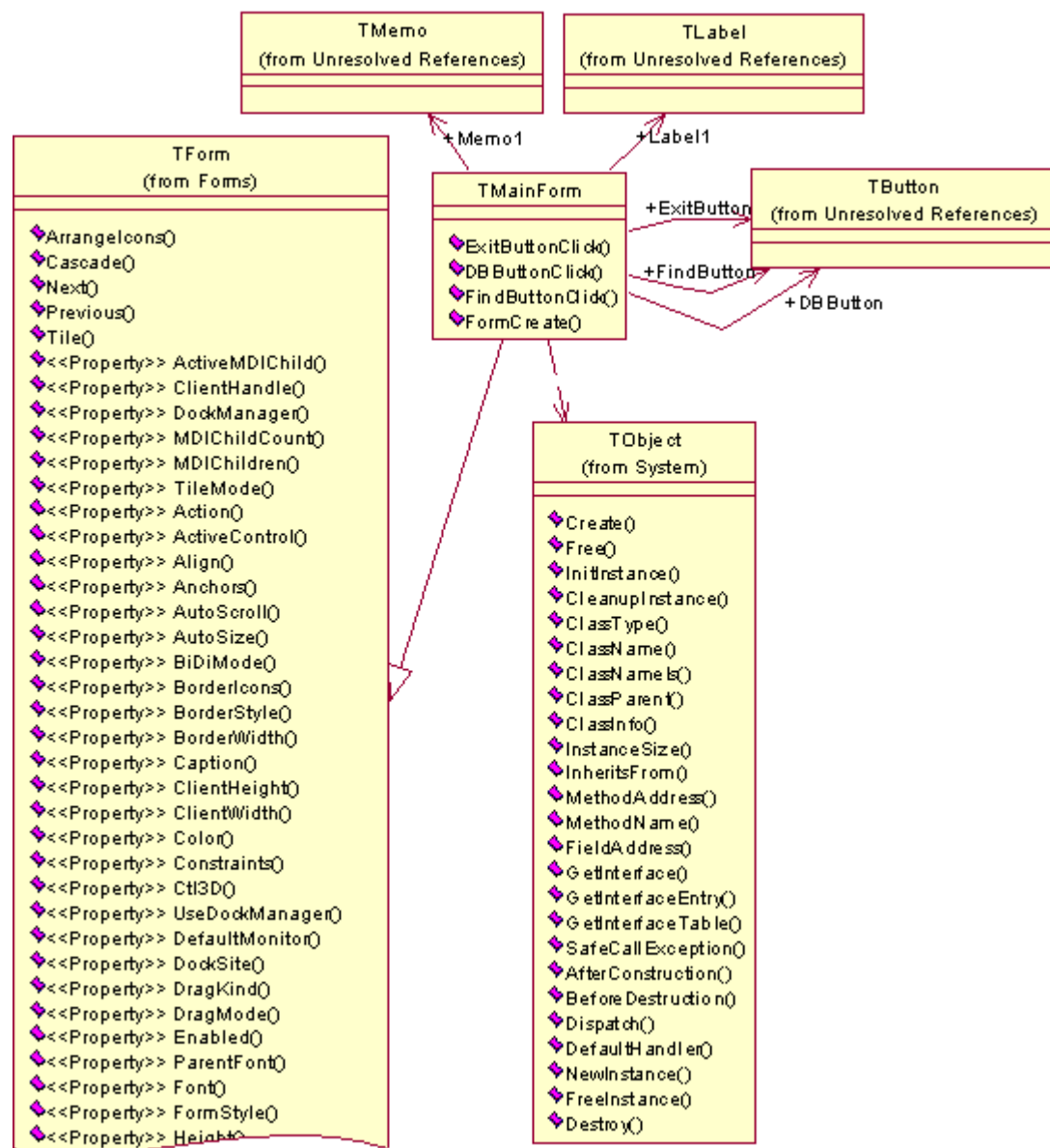


рис. 13а

`ExitButtonClick()` - обработчик нажатия на кнопку `ExitButton`. Закрывает приложение.

`DBButtonClick()` - обработчик нажатия на кнопку `DBButton`. Открывает пользователю диалог редактирования и просмотра БД.

`FindButtonClick()` - обработчик нажатия на кнопку `FindButton`. Открывает пользователю диалог поиска объектов в БД.

`FormCreate()` – конструктор формы.

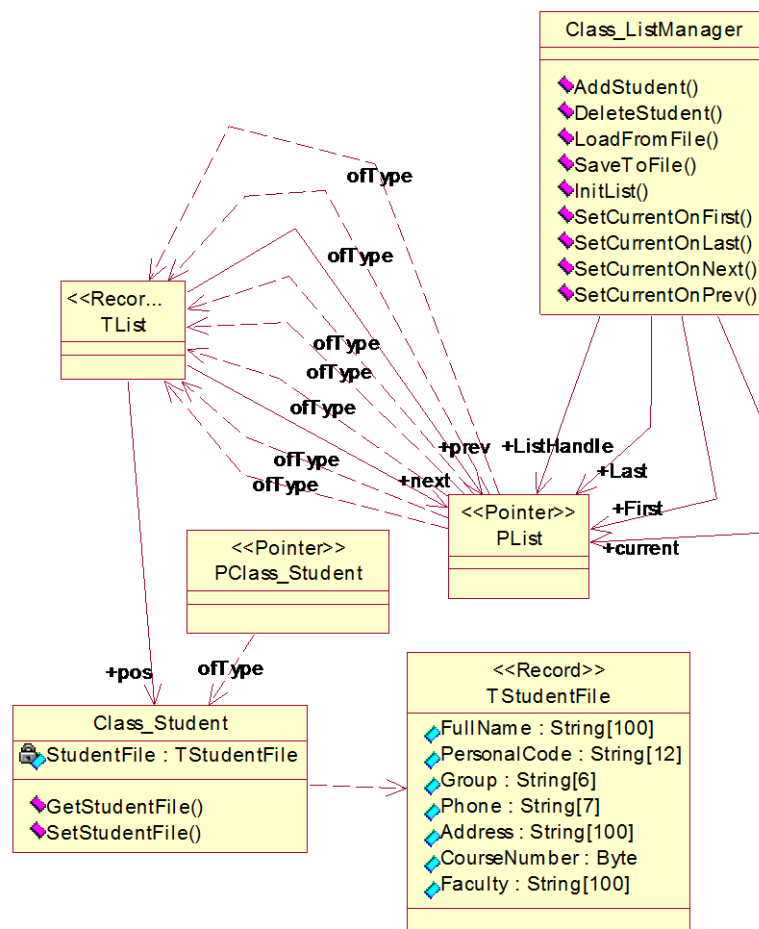


рис. 13а

**Class\_ListManager** – экземпляр этого класса представляет собой управляющий объект. Включает в себя структуру TList - список обрабатываемых объектов. Предоставляет средства работы с БД.

AddStudent() – создание и внесение нового объекта в ООБД.

DeleteStudent() – корректное удаление объекта из ООБД.

LoadFromFile() – загрузка всей БД с файла.

SaveToFile() – сохранение всей БД в файле.

InitList() – метод подготовки списка к работе. Выполняется один раз, при старте приложения.

SetCurrentOnFirst() – установка текущей позиции на первый элемент списка.

SetCurrentOnLast() - установка текущей позиции на последний элемент списка.

SetCurrentOnNext() - установка текущей позиции на следующий элемент списка.

SetCurrentOnPrev() - установка текущей позиции на предыдущий элемент списка.

## 5. Оценка качества логической структуры модели с помощью метрики Чидамбера – Кемерера

- WMC – взвешенные методы на класс (количество методов в классе);
- DIT – высота дерева наследования (длина максимального пути от данного класса до корневого класса в иерархии классов);
- NOC – количество детей (количество непосредственных наследников класса);
- CBO – сцепление между классами объектов (количество сцеплений класса; сцепление образует вызов метода или свойства в другом классе);
- RFC – отклик для класса (мощность набора всех методов, которые могут выполняться в ответ на прибытие сообщения в объект класса – количество методов класса плюс количество методов других классов, вызываемых из данного класса);
- LCOM – недостаток связности в методах (количество пар методов, не связанных по свойствам класса, минус количество пар методов, имеющих такую связь);

### 5.1. Описание процесса извлечения метрик.

**Нахождение WMC:** Пусть, начальный вес каждого метода равен единице. Тогда, вес взвешенных методов будет равен их количеству.

**Нахождение NOC:** Ни один класс не имеет дочерних форм.

**Нахождение количества сцеплений между классами объектов (CBO метрика):**

CBO- количество сотрудничеств, предусмотренных для класса(количество классов с которыми он соединен). Методы данного класса используют методы или экземплярные переменные другого класса. CBO для каждого класса должно иметь разумно низкое значение.

**Нахождение RFC:** Из-за того, что классы не используют переменные других классов, то метрика RFC будет равна сумме WMC+CBO(из определения метрики).

**Нахождение LCOM:** т.к. все методы обращаются к общим свойствам, то LCOM во всех случаях равен 0.

Оценка качества с помощью метрик Чидамбера-Кемерера

Имя класса	WMC	DIT	NOC	CBO	RFC	LCOM
Class_Student	2	1	0	0	2	0
Class_Manager	9	1	0	2	11	0
TMainForm	4	5*	0	9	13	0
TConfirmForm	3	5*	0	1	4	0
TDBForm	17	5*	0	16	33	0
Среднее	7	3.4	0	5.6	12.4	0

\* - высота до вершины 5, т.к. структура имеет вид: TObject → TPersistent → TComponent → TControl → TWinControl → TForm.

Прокомментируем результаты. Учитывая тот факт, что программа сама по себе не сложна, к тому же реализована всего на трёх модулях, оценки качества довольно низки, притом, что классы среды Delphi составляют основную весовую долю при подсчёте среднего.



Можно так же подчеркнуть, что столь низкие величины оценок подчёркивают и простоту реализации – удалось реализовать требуемую заказчиком функциональность программы и при этом, сохранить простоту кода.

## 6. Компонентная диаграмма

На рисунке 14 приведена компонентная диаграмма.

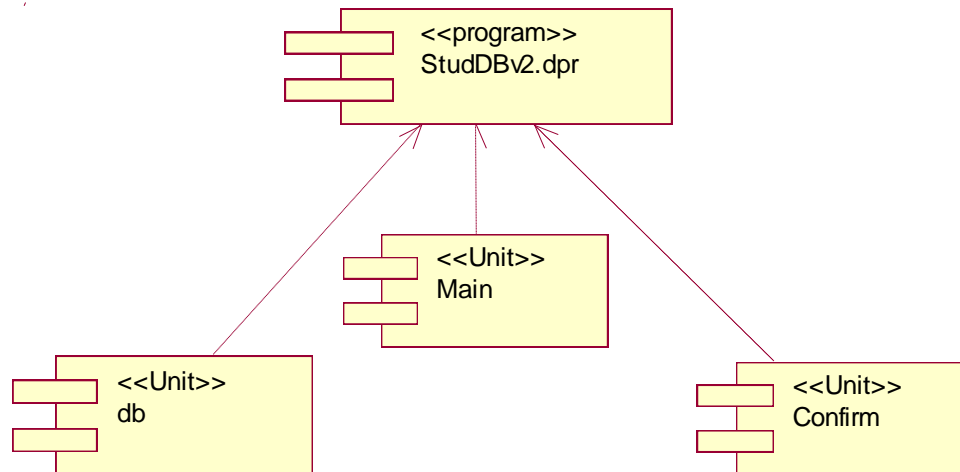


рис. 14.

На компонентной диаграмме показана реализация системы, группируемая вокруг главного программного элемента StudDB2.dpr. В свою очередь, он использует компоненты:

Main - реализует интерфейс главного окна.

DB - реализует интерфейс окна для вывода базы данных и её редактирования.

Confirm - реализует интерфейс окна для предложения не забыть сохранить изменения.

## 7. Диаграмма размещения

В конечном итоге, программа представляет собой единственный запускаемый файл, рассчитанный на работу в локальном режиме на стандартном однопроцессорном компьютере. Диаграмма размещения приведена на рисунке 15.

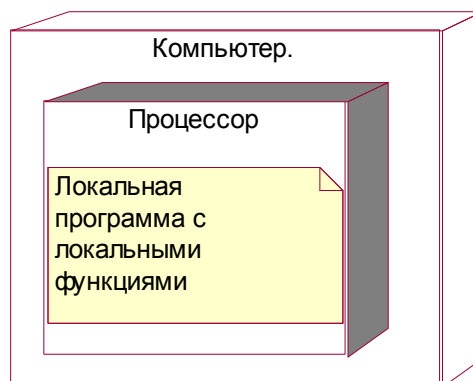


Рис. 15.

## 8. Процесс написания программы

Помимо основной цели написания программы, разработчики поставили себе цель попытаться следовать классическому пути написания программы с применением технологии визуального проектирования на начальных этапах развития проекта. О том, насколько это удалось, и какие выводы можно сделать описано ниже.

Опишем основные моменты стадии визуального проектирования. Для начала, в среде Delphi создадим новый проект File:New:Application. Добавим необходимое количество модулей и форм. В нашем случае 2 модуля с формами. Назвав их своими именами, получим первое представление нашего проекта. На данный момент программный код будет следующим:

### Программа StudDbv2

```
program StudDbv2;
uses
  Forms,
  main in 'main.pas' {MainForm},
  db in 'db.pas' {DbForm};

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TMainForm, MainForm);
  Application.CreateForm(TDbForm, DbForm);
  Application.Run;
end.
```

#### Модуль Main

```
unit main;
interface
uses Windows, Messages, SysUtils,
  Variants, Classes, Graphics,
  Controls, Forms, Dialogs;

type
  TMainForm = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var MainForm: TMainForm;

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


#### Модуль db

```
unit db;
interface
uses Windows, Messages, SysUtils,
  Variants, Classes, Graphics,
  Controls, Forms, Dialogs;

type
  TDbForm = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var DbForm: TDbForm;


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

Визуально это выглядит в виде двух форм. Сохранив и закрыв проект Delphi, приступаем к проектированию в среде Rational Rose. При создании нового проекта была выбрана модель Delphi Framework, т.к. она содержит полностью обратную проектируемую модель следующих библиотечных модулей Delphi4: Classes, Controls, Dialogs, Forms, Graphics, Messages, SysUtils, System, Windows. К тому же, если следует использовать прочие модули, то необходимо лишь импортировать соответствующий \*.cat файл. В результате использования Delphi Framework до начала визуального проектирования в разделе Logical View был автоматически создан пакет Delphi Framework со следующими модулями:

Classes, Controls, Dialogs, Forms, Graphics, Messages, SysUtils, System, Windows. Если не использовать модель Delphi Framework, то при использовании моста в ветке Logical View могут появляться элементы с меткой Unresolved. Пропустим этапы составления Use-Case диаграмм и диаграмм последовательностей и перейдём сразу к проектированию классовых диаграмм и компонентного представления. Однако, так как проект и модули уже сгенерированны в среде Delphi, то что бы не повторять вручную их создание в модели Rose, воспользуемся мостом Rose Delphi Link. Для этого в Tools:Ensemble Tools:Rose Delphi Link воспользуемся элементом меню загрузки проекта(File: Load Project) и произведём синхронизацию проекта Rose(ещё пустым) с Delphi(имеющим разработанную модульно - формовую структуру) нажав на кнопку  Update All. Автоматически в модели Rose заполнятся ветви Logical View и Component View, т.е. те ветви, которые необходимо было бы заполнять вручную. Итак, на рисунке 16 представлен браузер с результатом первого применения моста Rose Delphi Link.

Все формы и модули представляются пустыми, т.е. в них нет никаких процедур и объявлений (за исключением автоматически сгенерированных средой Delphi). Теперь, будем формировать в модулях необходимые классы. Что и где добавлялось, приведено в таблице:

Модуль	Классы, структуры
Main	Class_ListManager Class_Student <<Pointer>> PClass_Student <<Poinet>> PList <<Record>> TList <<Record>> TStudentFile
Db	-

На примере Class\_ListManager распишем ход проектирования класса. Создадим новый класс. Сразу назовём его Class\_ListManager и в спецификациях в разделе Components свяжем его с модулем Main, поставив красную галочку на иконке Main (Assign из контекстного меню). А далее заполняем класс методами и атрибутами. Создав таким образом классы перечисленные в таблице, мы будем готовы воспользоваться мостом Rose Delphi Link для генерации программного кода. Для этого в Rose Delphi Link откроем проект Delphi(уже устаревший) и синхронизируем его с моделью Rose, нажав на кнопку Update All .

### Программа StudDbv2

```

program StudDbv2;
uses
  Forms,
  main in 'main.pas' {MainForm},
  db in 'db.pas' {DbForm};
{$R *.res}
begin
  Application.Initialize;
  Application.CreateForm(TMainForm, MainForm);
  Application.CreateForm(TDbForm, DbForm);
  Application.Run;
end.

```

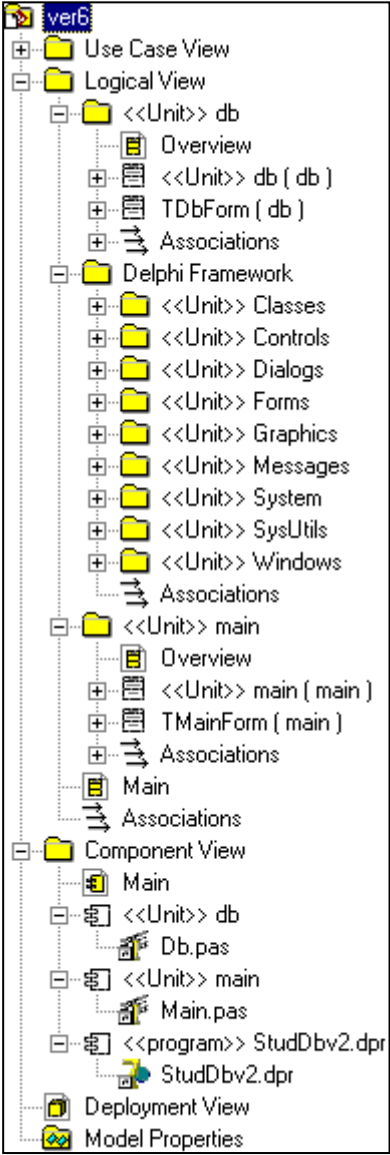


Рис. 16

## Модуль Main

```
interface
uses Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs;

type
  Class_Student = class;
  Class_ListManager = class;

  TMainForm = class (TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  Class_Student = class
  private
    StudentFile : TStudentFile;
  public
    function GetStudentFile : TStudentFile;
    // Zapisj vsej infy za odin raz
    procedure SetStudentFile (StudentFile : TStudentfile);
  end;

  PList = ^TList;
  PClass_Student = ^Class_Student;

  TList = record
    pos: PClass_Student;
    Index: integer;
    prev, next: PList;
  end;

  Class_ListManager = class
  public
    pos : PList;
    procedure AddStudent;
    procedure DeleteStudent;
    procedure LoadFromFile;
    procedure SaveToFile;
    procedure SetPosOnFirst;
    procedure SetPosOnLast;
    procedure SetPosOnNext;
    procedure SetPosOnPrev;
    function GetSizeOfList : Integer;
    // Sbros vsego spiska
    procedure ResetList;
  private
    SizeOfList : Integer;
  end;

  TStudentFile = record
    // Imja i familija studenta
    FullName : String;
    PersonalCode : String;
    Group : String;
    Phone : String;
    Address : String;
    // Kurs po schetu.
    CourseNumber : Byte;
    Faculty : String;
  end;
```

```

var MainForm: TMainForm;

implementation

procedure Class_ListManager.AddStudent;
begin
end;
procedure Class_ListManager.DeleteStudent;
begin
end;
procedure Class_ListManager.LoadFromFile;
begin
end;
procedure Class_ListManager.SaveToFile;
begin
end;
procedure Class_ListManager.SetPosOnFirst;
begin
end;
procedure Class_ListManager.SetPosOnLast;
begin
end;
procedure Class_ListManager.SetPosOnNext;
begin
end;
procedure Class_ListManager.SetPosOnPrev;
begin
end;
function Class_ListManager.GetSizeOfList : Integer;
begin
end;
function Class_Student.GetStudentFile : TStudentFile;
begin
end;
procedure Class_Student.SetStudentFile (StudentFile : TStudentfile);
begin
end;
procedure Class_ListManager.ResetList;
begin
end;
{$R *.dfm}
end.

```

## Модуль db

```

unit db;
interface
uses Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs;

type
  TDbForm = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var DbForm: TDbForm;

```

В модуле Main, в последствии, будут изменены некоторые классы. Теперь, вновь вернёмся к проектированию в среде Delphi. На основе классовых диаграмм, т.е. представления о том с какими данными мы будем работать, и как их следует

---

Курсовой проект. -= Технологии Проектирования П.С. -= Сытник А., Чижов Ю. 29/33

обрабатывать, спроектируем пользовательский интерфейс (внешний вид) программы. Необходимо разработать внешний вид каждого окна: разместить кнопки, элементы ввода/вывода информации и т.д. Все элементы интерфейса являются объектами (экземплярами классов), поэтому имеет смысл, по окончании разработки интерфейса, воспользоваться мостом Rose Delphi Link и перевести код в UML. Аналогично, как это делали и раньше, синхронизируем Rose модель с Delphi кодом. Конечный вид классовой диаграммы приведён на странице **YY**. Ниже приведён конечный код.

## Модуль Main

```
unit Main;
interface

uses Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, StdCtrls;

type
  Class_Student = class;
  Class_ListManager = class;
  TMainForm = class (TForm)
    DBButton: TButton;
    FindButton: TButton;
    Button1: TButton;
    Label1: TLabel;
    Memo1: TMemo;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  TStudentFile = record
    // Imja i familija studenta
    FullName : String;
    PersonalCode : String;
    Group : String;
    Phone : String;
    Address : String;
    // Kurs po schetu.
    CourseNumber : Byte;
    Faculty : String;
    FileNumber : Integer;
  end;

  Class_Student = class
  private
    StudentFile : TStudentFile;
  public
    function GetStudentFile : TStudentFile;
    // Zapisj vsej infy za odin raz
    procedure SetStudentFile (StudentFile : TStudentfile);
  end;

  PList = ^TList;
  PClass_Student = ^Class_Student;

  TList = record
    pos: PClass_Student;
    Index: integer;
    prev, next: PList;
  end;
```

```

Class_ListManager = class
public
    pos : PList;
    procedure AddStudent;
    procedure DeleteStudent;
    procedure LoadFromFile;
    procedure SaveToFile;
    procedure SetPosOnFirst;
    procedure SetPosOnLast;
    procedure SetPosOnNext;
    procedure SetPosOnPrev;
    function GetSizeOfList : Integer;
    // Sbros vsego spiska
    procedure ResetList;
private
    SizeOfList : Integer;
end;

var MainForm: TMainForm;
implementation

uses db;
procedure Class_ListManager.AddStudent;
begin
end;
procedure Class_ListManager.DeleteStudent;
begin
end;
procedure Class_ListManager.LoadFromFile;
begin
end;
procedure Class_ListManager.SaveToFile;
begin
end;
procedure Class_ListManager.SetPosOnFirst;
begin
end;
procedure Class_ListManager.SetPosOnLast;
begin
end;
procedure Class_ListManager.SetPosOnNext;
begin
end;
procedure Class_ListManager.SetPosOnPrev;
begin
end;
function Class_ListManager.GetSizeOfList : Integer;
begin
end;
function Class_Student.GetStudentFile : TStudentFile;
begin
end;
procedure Class_Student.SetStudentFile (StudentFile : TStudentfile);
begin
end;
procedure Class_ListManager.ResetList;
begin
end;
{$R *.dfm}

end.

```

## Модуль db

```
unit db;

interface

uses Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, main, StdCtrls, ExtCtrls;

type
  TDBForm = class (TForm)
    CaptionLabel: TLabel;
    FullNameLEdit: TLabelledEdit;
    Bevel1: TBevel;
    PersonalCodeLEdit: TLabelledEdit;
    GroupLEdit: TLabelledEdit;
    PhoneLEdit: TLabelledEdit;
    AddressLEdit: TLabelledEdit;
    CourseLEdit: TLabelledEdit;
    FacultyCBox: TComboBox;
    FacultyLabel: TLabel;
    Bevel2: TBevel;
    FileNumberLEdit: TLabelledEdit;
    EditingPanel: TPanel;
    SearchingPanel: TPanel;
    FirstButton: TButton;
    LastButton: TButton;
    PreventButton: TButton;
    NextButton: TButton;
    NewButton: TButton;
    DeleteButton: TButton;
    ResetButton: TButton;
    RefreshButton: TButton;
    ForwardButton: TButton;
    BackwardButton: TButton;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var DBForm: TDBForm;
implementation
{$R *.dfm}
end.
```

К этому моменту имеем “каркас” программы, полностью спроектированный визуальными средствами Rational Rose и Borland Delphi. Теперь, мы готовы для дальнейшего развития проекта. Однако, проектирование классовых диаграмм на этом не завершается, т.к. в ходе непосредственного кодирования будут появляться новые методы и обработчики. В зависимости от сложности изменений и личных предпочтений разработчиков модификации классов можно проводить как непосредственно в коде, так и в модели Rose, однако при этом код и модель всегда должны быть синхронизированы. Так же следует избегать одновременного (например, разными лицами) изменения модели и кода, так как это может привести к осложнениям при синхронизации.



## **9. Выводы**

В ходе написания курсового проекта были применены на практике и более подробно изучены технологии визуального конструирования сложных программных систем. Был получен дополнительный опыт разработки визуальной модели системы, её оценивание качества с помощью набора метрик Чедамбера и Кемерера. Ознакомились с программной средой Rational Rose и её совместного использования с Borland Delphi через “мост коммутации” Rose-Delphi Link. Можно заключить, что визуальное проектирование сложных ОО программ, на первых этапах разработки, существенно облегчает труд разработчика, предоставляет возможность зрительного восприятия и начального анализа будущего проекта. К тому же, благодаря дополнительному инструментарию, автоматизируется процесс кодирования визуально спроектированной модели в каркас будущей программы.

## **10. Распечатка программы**

Исходный текст программы, написанный на языке Borland Delphi 7, смотри в приложении.

## **11. Список используемой литературы**

- 1) Орлов С.А., Современные технологии конструирования сложных программных систем. – Рига: ИТС, 1999.-289 с.:ил.
- 2) Орлов С.А., Введение в визуальное моделирование. – Рига: ИТС, 2001. – 55с.: ил.
- 3) Орлов С.А. Конспект лекций по технологиям конструирования программных систем.
- 4) Дарахвелидзе П.Г., Марков Е.П., Программирование в Delphi 4. – СПб.: БХВ – Санкт-Петербург, 1999.- 864с., ил.