## Тема 3. Отношение между классами.

## Включение, вложение и наследование классов

3.1. Отношение включения и вложение классов

3.2. Отношение наследования

3.3. Абстрактные классы

[3.4. Общие сведения об интерфейсных классах](file:///F:\2015УчП-Раз-5-Инф-Шакин-П+Л\КНИГА-4\Тема-05-00-Огл-Пред-Введ%20ПРОВ-23_11_2014.docx#_5.4.5.__Задачи)

[3.5. Задачи для самостоятельного решения](file:///F:\2015УчП-Раз-5-Инф-Шакин-П+Л\КНИГА-4\Тема-05-00-Огл-Пред-Введ%20ПРОВ-23_11_2014.docx#_5.4.5.__Задачи)

3.6. Лабораторная работа по теме «Отношение между классами.

Включение, вложение и наследование классов»

3.7. Контрольные вопросы

### 3.1. Отношение включения и вложение классов

Объекты (экземпляры) классов, создаваемые в ходе выполнения программы, взаимодействуют между собой по определенным в классе правилам. В результате взаимодействия между объектами возникают отношения, которые должны быть заранее определены на этапе проектирования объектно-ориентированного приложения. В зависимости от задачи, которая решается проектируемой программой, классы, входящие в эту программу, могут находиться в разных отношениях. В традиционном ООП предусмотрены четыре типа отношений между классами:

* Отношение независимости – самое простое отношение между классами, т.е. независимость порождаемых ими объектов.
* Отношение использования – непосредственная зависимость между классами.
* Отношение включения – реализует логические связи типа «является составной частью».
* Отношение наследования – реализует логические связи типа «является частным случаем» (подробно рассматривается в следующем параграфе).

***Отношение использования (зависимости)*** – самое очевидное и распространенное отношение между классами, которое означает, что один класс зависит от другого. Во всех ситуациях, когда один объект посылает сообщение другому, можно говорить о зависимости между этими объектами. В общем случае класс **А** использует класс **Б**, если:

* член класса А отправляет сообщение объекту класса B;
* член класса А создает или возвращает объекты класса B.

***Отношение включения*** означает, что объект класса **А** содержит внутренние объекты класса **B**. Включение называют взаимоотношением типа **«имеет» (has-a)**. Мы говорим, что библиотека имеет книги (в библиотеке есть книги) или лес имеет деревья (в лесу есть деревья). Включение также называют взаимоотношением типа «часть целого»: книга является частью библиотеки. Таким образом, включение появляется, когда один объект является атрибутом (полем) другого объекта.

В теории объектно-ориентированного анализа различают две формы отношения включения - ***композицию*** и ***агрегацию***:

* При ***отношении композиции*** объекты одного или нескольких разных классов входят как поля в объект другого (включающего) класса. Таким образом, включенные объекты не существуют без включающего их объекта.
* При ***отношении агрегации*** объект одного класса объединяет уже существующие объекты других классов. То есть и включающий объект, и включаемые в него объекты, могут существовать в некотором смысле независимо друг от друга; при уничтожении включающего объекта входившие в него объекты сохраняются.

|  |
| --- |
| **Public Class ТОЧКА**  **Private Fx, Fy As Double** *'координаты точки - закрытые поля*  *'Свойства для доступа к полям*  **Public Property X() As Double**  **Get**  **Return Fx**  **End Get**  **Set(ByVal value As Double)**  **Fx = value**  **End Set**  **End Property**  **Public Property Y() As Double**  **Get**  **Return Fy**  **End Get**  **Set(ByVal value As Double)**  **Fy = value**  **End Set**  **End Property**  **End Class** |

*Рис. 3.1-1. Программный код класса* ***ТОЧКА***

Для рассмотрения особенностей реализации ***отношений композиции и агрегации на языке VB***, определим для точки на плоскости класс **ТОЧКА** с двумя закрытыми полями – координатами точки на плоскости **Fx , Fy** и открытыми свойствами для доступа к этим координатам (Рис. 3.1-1).

В классе **ТОЧКА** нет явно определенного конструктора, и компилятор добавит конструктор по умолчанию без параметров. При создании объекта класса **ТОЧКА** координаты создаваемой точки получат нулевые значения.

Объекты класса **ТОЧКА** можно по-разному включать в более сложные классы. Рассмотрим в качестве включающего класс **ОКРУЖНОСТЬ**, объект которого представляет окружность на плоскости, а объект класса **ТОЧКА** будем использовать в качестве центра окружности.

**Пример 3.1-1. Создать программный проект для реализации отношения композиции двух классов: ТОЧКА и ОКРУЖНОСТЬ.**

Программный код класса **ОКРУЖНОСТЬ** представлен на рис. 3.1-2.

|  |
| --- |
| **Option Strict On**  **Imports System.Math**  **Public Class ОКРУЖНОСТЬ**  *'=== Закрытые поля =======*  **Private Frad As Double** *' радиус окружности*  *'центр окружности - ссылка на объект класса ТОЧКА*  *'с инициализацией(созданием) этого объекта*  **Private Fcentre As New ТОЧКА**  *'==== Свойства=========*  *'свойство для радиуса окружности*  **Public Property Rad() As Double**  **Get**  **Return Frad**  **End Get**  **Set(ByVal value As Double)**  **Frad = value**  **End Set**  **End Property**  *'свойство для центра окружности*  **Public Property Centre() As ТОЧКА**  **Get**  **Return Fcentre**  **End Get**  **Set(ByVal value As ТОЧКА)**  **Fcentre = value**  **End Set**  **End Property**  *'свойство для значения длины окружности*  **Public ReadOnly Property Len() As Double**  **Get**  **Return 2 \* PI \* Frad**  **End Get**  **End Property**  *' ====Метод для вывода характеристик окружности=====*  *'собираем характеристики в одну строку s для вывода в текстовое поле*  **Public Sub Display(ByRef s As String)**  **s = "координаты центра: x="**  **s = s + CStr(Fcentre.X) + " y=" + CStr(Fcentre.Y) + vbNewLine**  **s = s + "радиус = " + CStr(Frad) + vbNewLine**  **s = s + "длина окружности = " + CStr(Len)**  **End Sub**  **End Class** |

*Рис. 3.1-2. Программный код* ***Примера 3.1-1*,***содержащий**класс* ***ОКРУЖНОСТЬ*** *для случая композиции с классом* ***ТОЧКА***

В классе **ОКРУЖНОСТЬ** два закрытых поля **Frad** – радиус окружности и **Fcentre** – ссылка на объект класса **ТОЧКА**. Для реализации этой ссылки при объявлении поля **Fcentre** сразу вызывается конструктор класса **ТОЧКА**.

|  |
| --- |
| **Private Fcentre As New ТОЧКА** |

Таким образом, при создании каждого объекта «окружность» в качестве поля этого объекта всегда создается объект «точка», определяющий центр окружности.

Для доступа к характеристикам объекта - окружности в классе имеются три открытых свойства **Rad** – радиус окружности, **Centre** – центр окружности и свойство только для чтения **Len**, определяющее длину окружности, которая вычисляется из её радиуса.

В классе также определен метод **Display()** для вывода координат центра и значений других характеристик объекта, представляющего окружность. В методе значения характеристик окружности «склеиваются» в одну строку **s**, которая возвращается через параметр по ссылке.

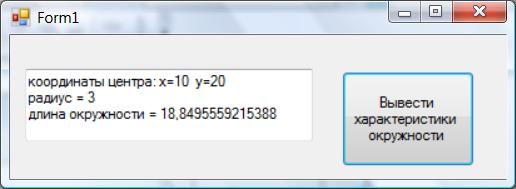
Так как в классе **ОКРУЖНОСТЬ** нет явно определенных конструкторов, то в него неявно добавляется конструктор без параметров, и поля объекта, который будет создаваться с его помощью, получат значения по умолчанию.

В событийной процедуре, код которой представлен на рис. 3.1-3, создается объект **rim** класса **ОКРУЖНОСТЬ**. Затем с помощью свойствRad и Centre изменены значения его полей, и с использованием метода **Display()** сведения о характеристиках объекта выводятся в текстовое поле на форме.

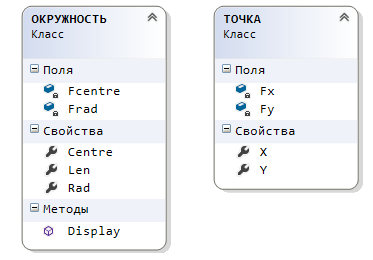
Основное, на что следует обратить внимание, - в программе нет отдельно существующего объекта класса **ТОЧКА**. Именно это является основным признаком композиции классов. Объект класса **ТОЧКА** явно создается только при создании объекта класса **ОКРУЖНОСТЬ**. Результаты работы программы, реализующей композицию классов, представлены на рис. 3.1-4.

|  |
| --- |
| **Public Class Form1**  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button1.Click**  **Dim s As String = ""** *'строка для вывода в текстовое поле*  **Dim rim As New ОКРУЖНОСТЬ**  **rim.Centre.X = 10**  **rim.Centre.Y = 20**  **rim.Rad = 3**  **rim.Display(s)**  **TextBox1.text = s**  **End Sub**  **End Class** |

*Рис. 3.1-3. Программный код событийной процедуры**для случая композиции двух классов*



*Рис. 3.1-4. Результаты работы проекта* ***Примера 3.1-1*,***использующего**композицию двух классов*



*Рис. 3.1-5. Схема классов в отношении композиции*

На рис. 3.1-5 представлена Схема классов, находящихся в отношении композиции. Конструкторы по умолчанию, которые добавляются компилятором автоматически, на схемах классов не показываются. Тот факт, что ссылка на объект класса **ТОЧКА** является значением поля **Fcentre** объекта класса **ОКРУЖНОСТЬ**, также никак явно не обозначен.

Заметим, что в объектной модели решения задачи в лабораторной работе (п. 2.7 и 3.6) реализуется как раз отношение композиции двух классов **СТУДЕНТ** и **ГРУППА**.

**Пример 3.1-2. Используя агрегацию классов, изменить класс ОКРУЖНОСТЬ, сохранив неизменным класс ТОЧКА.**

Сокращенный программный код класса **ОКРУЖНОСТЬ** для реализации отношения агрегации двух классов представлен на рис. 3.1-6.

|  |
| --- |
| **Option Strict On**  **Imports System.Math**  **Public Class ОКРУЖНОСТЬ**  *'=== Закрытые поля =======*  **Private Frad As Double** *' радиус окружности*  *'центр окружности - ссылка на объект класса ТОЧКА*  *'без инициализации(создания) этого объекта*  **Private Fcentre As ТОЧКА**  *'=======Конструктор ==========*  **Public Sub New(ByRef p As ТОЧКА, ByVal r As Double)**  **Fcentre = p**  **Frad = r**  **End Sub**  *'==== Свойства=========*  *'свойство для радиуса окружности*  **Public Property Rad() As Double**  **. . .**  **End Property**  *'свойство для центра окружности*  **Public Property Centre() As ТОЧКА**  **. . .**  **End Property**  *'свойство для значения длины окружности*  **Public ReadOnly Property Len() As Double**  **. . .**  **End Property**  *' ====Метод для вывода характеристик окружности=====*  **Public Sub Display(ByRef s As String)**  **. . .**  **End Sub**  **End Class** |

*Рис. 3.1-6. Сокращённый программный код класса* ***ОКРУЖНОСТЬ*** *для случая агрегации с классом* ***ТОЧКА***

В тексте нового класса **ОКРУЖНОСТЬ** показаны полностью только объявления полей и конструктор, первый параметр которого – ссылка на объект класса **ТОЧКА**. Свойства и метод **Display()** остались неизменными.

Теперь для создания объектакласса **ОКРУЖНОСТЬ** необходимо, чтобы уже существовал объект класса **ТОЧКА**, так как значение поля **Fcentre** будет определено с его помощью.

Схема классов, находящихся в отношении агрегации, представленная на рис. 3.1-7 практически та же, что и схема композиции. Только в классе **ОКРУЖНОСТЬ** явно присутствует конструктор **New** общего вида (конструктор не по умолчанию).

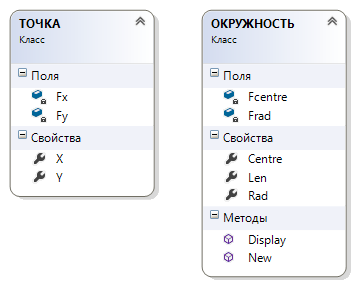


Рис. 3.1-7. Схема классов в отношении агрегации

|  |
| --- |
| **Public Class Form1**  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button1.Click**  **Dim s As String = ""** *'строка для вывода в текстовое поле*  **Dim pt As New ТОЧКА** *'создание объекта класса**ТОЧКА*  **pt.X = 10**  **pt.Y = 20**  *'создание объекта класса ОКРУЖНОСТЬ*  **Dim rim As New ОКРУЖНОСТЬ(pt, 3)**  **rim.Rad = 3**  **rim.Display(s)**  **TextBox1.text = s**  **End Sub**  **End Class** |

*Рис. 3.1-8. Программный код событийной процедуры**для случая   
агрегации двух классов*

В событийной процедуре, код которой представлен на рис. 3.1-8, создан объект **pt** класса **ТОЧКА**. Затем с помощью свойств **X** и **Y** изменены значения его полей. На основе этого объекта конструктор класса **ОКРУЖНОСТЬ** формирует объект **rim** своего класса. С использованием метода **Display()** сведения о характеристиках построенного объекта выводятся в текстовое поле на форме.

Результаты работы этой программы, реализующей агрегацию классов, полностью совпадают с представленными на рис. 3.1-4 результатами программы композиции классов.

В отличие от композиции, при агрегации в классе **ОКРУЖНОСТЬ** нет явной инициализации поля **Fcentre**. Для обеспечения включения объекта класса **ТОЧКА** в объект класса **ОКРУЖНОСТЬ** в классе **ОКРУЖНОСТЬ** явно определен конструктор, одним из параметров которого служит ссылка на объект класса **ТОЧКА**. В теле конструктора значение этой ссылки присваивается полю **Fcentre** класса **ОКРУЖНОСТЬ**.

Заметим, что программный проект для объектной модели, имитирующей движение автомобилей на шоссе, рассмотренной в примере 2.2-2 (рис. 2.2-5 и рис. 2.2-6), реализует как раз агрегацию классов **TROAD (ДОРОГА)** и **TCAR (МАШИНА)**.

В описание класса в качестве его члена может войти описание типа, которым может быть другой класс. Этот внутренний класс называют *вложенным* классом, а включающий его класс – *внешним*. Особенностью вложенного класса является то, что ему доступны все члены внешнего класса, даже если эти члены закрытые (**Private**). Обычно вложенный класс вводится только для выполнения действий внутри внешнего класса и «не виден» вне включающего его класса. Однако, вложенный класс может быть объявлен с модификатором **Public** и тогда он становится доступен везде, где виден внешний класс. Если открытый класс **ДЕРЕВО** вложен в класс **ЛЕС**, то для внешнего обращения к вложенному классу следует использовать имя с точечной нотацией: **ЛЕС.ДЕРЕВО**.

**Пример 3.1-3.** **Описать класс ОКРУЖНОСТЬ, центр которой представляет объект вложенного класса ТОЧКА.**

Чтобы возможности внешнего класса **ОКРУЖНОСТЬ** были близки к возможностям уже рассмотренных классов, реализующих композицию и агрегацию, сделаем вложенный класс **ТОЧКА** открытым. Программный код класса **ОКРУЖНОСТЬ** с вложенным классом **ТОЧКА**, а также текст событийной процедуры представлены на рис. 3.1-9.

|  |
| --- |
| **Option Strict On**  **Imports System.Math**  *'Описание внешнего открытого класса ОКРУЖНОСТЬ*  **Public Class ОКРУЖНОСТЬ**  *'Описание вложенного открытого класса ТОЧКА*  **Public Class ТОЧКА**  **Private Fx, Fy As Double** *'координаты точки - закрытые поля*  *'Свойства для доступа к полям*  **Public Property X() As Double**  **Get**  **Return Fx**  **End Get**  **Set(ByVal value As Double)**  **Fx = value**  **End Set**  **End Property**  **Public Property Y() As Double**  **Get**  **Return Fy**  **End Get**  **Set(ByVal value As Double)**  **Fy = value**  **End Set**  **End Property**  **End Class***'Конец описания класса ТОЧКА*  *'Поля и методы внешнего класса ОКРУЖНОСТЬ*  *'=== Закрытые поля =======*  **Private Frad As Double** *' радиус окружности*  *'центр окружности - ссылка на объект*  *'класса ТОЧКА*  *'с инициализацией(созданием) этого объекта*  **Private Fcentre As New ТОЧКА**  *'==== Свойства=========*  *'свойство для радиуса окружности*  **Public Property Rad() As Double**  **Get**  **Return Frad**  **End Get**  **Set(ByVal value As Double)**  **Frad = value**  **End Set**  **End Property**  *'свойство для центра окружности*  **Public Property Centre() As ТОЧКА**  **Get**  **Return Fcentre**  **End Get**    **Set(ByVal value As ТОЧКА)**  **Fcentre = value**  **End Set**  **End Property**  *'свойство для значения длины окружности*  **Public ReadOnly Property Len() As Double**  **Get**  **Return 2 \* PI \* Frad**  **End Get**  **End Property**  *'==Метод для вывода характеристик =====*  *'окружности собираем характеристики в*  *'одну строку s для вывода в текстовое поле*  **Public Sub Display(ByRef s As String)**  **s = "координаты центра: x="**  **s = s + CStr(Fcentre.X) + " y=" + CStr(Fcentre.Y) + vbNewLine**  **s = s + "радиус = " + CStr(Frad) + vbNewLine**  **s = s + "длина окружности = " + CStr(Len)**  **End Sub**  **End Class**  *'Код модуля формы*  **Public Class Form1**  *'Событийная процедура нажатия на кнопку*  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button1.Click**  **Dim s As String = ""** *'строка для вывода в текстовое поле*  **Dim rim As New ОКРУЖНОСТЬ**  **rim.Centre.X = 10**  **rim.Centre.Y = 20**  **rim.Rad = 3**  **rim.Display(s)**  **TextBox1.text = s**  **End Sub**  **End Class** |

*Рис. 3.1-9. Программный код проекта с классом* ***ОКРУЖНОСТЬ****, вложенным классом* ***ТОЧКА*,** *событийной процедуры**и Схемой вложенных классов*

Результаты работы этого проекта, демонстрирующего вложение классов, также полностью совпадают с представленными на рис. 3.1-3 результатами программ композиции и агрегации классов.

В классе **ОКРУЖНОСТЬ** для инициализации поля **Fcentre** используется конструктор без параметров (по умолчанию) класса **ТОЧКА**. Префикс **ОКРУЖНОСТЬ** при этом не требуется, хотя его использование к ошибке не приведет, и то же самое относится к обозначению типа свойства **Centre**.

|  |
| --- |
| *'Использование префикса ОКРУЖНОСТЬ. необязательно*  **Private Fcentre As New ОКРУЖНОСТЬ.ТОЧКА** |

Во внешнем классе **ОКРУЖНОСТЬ** нет явного определения конструкторов, поэтому в событийной процедуре для создания объекта, ассоциированного со ссылкой **rim**, используется конструктор по умолчанию этого класса **ОКРУЖНОСТЬ**.

|  |
| --- |
| **Dim rim As New ОКРУЖНОСТЬ** |

Для обращения к свойствам, определяющим центр окружности, требуется использовать имена с двойной квалификацией:

|  |
| --- |
| *'Обращение к свойствам вложенного класса*  **rim.Centre.X = 10**  **rim.Centre.Y = 20** |

В отличие от композиции и агрегации при вложении классов внутренний класс (не только объект!) не существует независимо от внешнего. На Схеме классов, которая также приведена на рис.3.1-9, вложенный класс изображается именно внутри внешнего класса.

### 3.2. Отношение наследования

#### 3.2.1 Наследование классов

***Отношение наследования классов*** обеспечивает важную возможность для повторного использования кода и полиморфизма. Для него используют название **«является» (is-a)**.

***Наследование*** – возможность создавать класс с полями, свойствами и методами, которые могут быть использованы в других классах. В терминах ООП базовый класс называется наследуемым (родительским), а производный от него – наследующим (дочерним, производным). При наследовании объект производного класса является частным случаем объекта базового класса. Например, автомобиль является частным случаем транспортного средства. Производный класс получает все возможности базового класса, но может быть также усовершенствован за счет добавления собственных возможностей. Возможности базового класса при этом остаются неизменными.

Цель наследования заключается в создании ба­зового класса, где инкапсулируются поля, свойства и методы, необходимые множеству производных классов того же типа. Так, можно создать базовый класс   
**БАНКОВСКИЙ\_СЧЁТ** и определить в нем метод **УЗНАТЬ\_БАЛАНС()**. Затем на основе этого класса создать два отдельных класса: **СРОЧНЫЙ\_ВКЛАД** и **ТЕКУЩИЙ\_СЧЁТ**. Поскольку они используют ту же логику для получения информации, что и класс **БАНКОВСКИЙ\_СЧЁТ**, то могут унаследовать от него метод **УЗНАТЬ\_БАЛАНС()**. Это позволяет программисту один раз на­писать общий код, который затем будет проще поддерживать. Как правило, производные классы не ограничены полями, свойствами и методами базового класса, а имеют более широкие возможности. В них можно определить присущие только им свойства и методы. Например, правила выполнения банковских операций требуют, чтобы при снятии денег с текущего счета на нем оставалась неко­торая минимально допустимая сумма, при снятии же денег со срочного вклада сохранения на сче­ту какой-то минимальной суммы не требуется. Поэтому каждый из производных классов может содержать собственное определение метода для обработки операции снятия денег со счета - метода **СНЯТЬ()**.

Чтобы с помощью VB создать производный класс, следует в его определении рядом с именем наследуемого класса указать ключевое слово **Inherits**.

**Пример 3.2-1.** **Изменить класс ОКРУЖНОСТЬ с заданным точкой центром, сделав его наследником класса ТОЧКА.**

Программный код производного класса **ОКРУЖНОСТЬ** представлен на рис. 3.2-1.

В производном классе **ОКРУЖНОСТЬ** явно определены закрытое поле **Frad**, три уже рассмотренных свойства **Rad, Centre, Len** и метод **Display()**. Код родительского класса **ТОЧКА** остался неизменным (рис.3.1-1). Он также содержит два закрытых поля, задающих координаты точки, и два открытых свойства **X** и **Y**, обеспечивающие доступ к этим полям. В классе **ТОЧКА** нет явного определения конструктора, он по умолчанию добавлен компилятором. Нет явного определения конструктора и в классе **ОКРУЖНОСТЬ**. Поэтому объекты класса **ОКРУЖНОСТЬ** можно создавать только с умалчиваемыми значениями полей.

Производный класс наследует все поля, свойства и методы родительского класса, **за исключением конструктора** – **конструктор базового класса не наследуется.** Поля базового класса непосредственно входят в число полей наследника, однако доступ к полям базового класса для методов, свойств и объектов производного класса разрешен не всегда. Закрытые (**Private**) члены базового класса недоступны наследнику.

Открытыми полями, свойствами и методами базового класса, так как они доступны дочернему классу, можно пользоваться как внутри класса – наследника, так и во внешнем мире, обращаясь к ним с помощью объектов класса – наследника. В примере класс **ТОЧКА** имеет два открытых свойства **X** и **Y**, в методе **Display()** производного класса **ОКРУЖНОСТЬ** выполняется непосредственное обращение к этим унаследованным свойствам.

|  |
| --- |
| **Option Strict On**  **Imports System.Math**  **Public Class ОКРУЖНОСТЬ**  **Inherits ТОЧКА** *'класс ОКРУЖНОСТЬ – наследник класса ТОЧКА*  *'=== Закрытое поле только для радиуса =======*  **Private Frad As Double** *' радиус окружности*  *'==== Свойства=========*  *'свойство для радиуса окружности*  **Public Property Rad() As Double**  **Get**  **Return Frad**  **End Get**  **Set(ByVal value As Double)**  **Frad = value**  **End Set**  **End Property**  *'свойство для центра окружности*  **Public Property Centre() As ТОЧКА**  **Get**  **Dim temp As ТОЧКА**  **temp.X = X : temp.Y = Y**  **Return temp**  **End Get**    **Set(ByVal value As ТОЧКА)**  **X = value.X**  **Y = value.Y**  **End Set**  **End Property**  *'свойство для значения длины окружности*  **Public ReadOnly Property Len() As Double**  **Get**  **Return 2 \* PI \* Frad**  **End Get**  **End Property**  *' ====Метод для вывода характеристик окружности=====*  *'собираем характеристики в одну строку s*  *' для вывода в текстовое поле*  **Public Sub Display(ByRef s As String)**  **s = "координаты центра: x="**  **s = s + CStr(X) + " y=" + CStr(Y) + vbNewLine**  **s = s + "радиус = " + CStr(Frad) + vbNewLine**  **s = s + "длина окружности = " + CStr(Len)**  **End Sub**  **End Class** |

*Рис. 3.2-1. Программный код класса* ***ОКРУЖНОСТЬ****,*

*наследника класса* ***ТОЧКА***

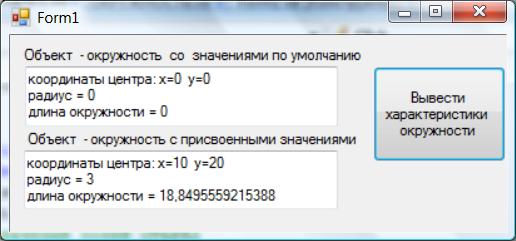
Особое внимание в нашем примере с наследованием следует уделить свойству **ОКРУЖНОСТЬ.Centre**. При агрегации и композиции класса **ТОЧКА** в класс **ОКРУЖНОСТЬ** значением этого свойства служит ссылка на непосредственно существующий объект класса **ТОЧКА**. В случае наследования в объекте класса **ОКРУЖНОСТЬ** объекта класса **ТОЧКА** нет – присутствуют только поля такого объекта и в классе **ОКРУЖНОСТЬ** доступны открытые свойства класса **ТОЧКА**. Поэтому для объявления в классе **ОКРУЖНОСТЬ** свойства **Centre** объект класса **ТОЧКА** приходится «реконструировать». В части **Get** этого свойства явно создается временный объект класса **ТОЧКА**, его полям через открытые свойства **X** и **Y** присваиваются значения полей, унаследованных классом **ОКРУЖНОСТЬ** от базового класса **ТОЧКА**. Ссылка на этот временный объект возвращается как значение свойства **ОКРУЖНОСТЬ.Centre**. В части **Set** свойства **ОКРУЖНОСТЬ.Centre** используются унаследованные свойства **X** и **Y** класса **ТОЧКА**.

Программный код событийной процедуры, демонстрирующей возможности производного класса приведен на рис. 3.2-2.

|  |
| --- |
| **Public Class Form1**  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button1.Click**  **Dim s As String = ""**  **Dim rim As New ОКРУЖНОСТЬ** *'Создание объекта класса*  **rim.Display(s)**  **TextBox1.Text = s**  *'Изменение(заполнение)значений полей объекта - наследника*  **rim.X = 10 : rim.Y = 20**  **rim.Rad = 3 : rim.Display(s)**  **TextBox2.Text = s**  **End Sub**  **End Class** |

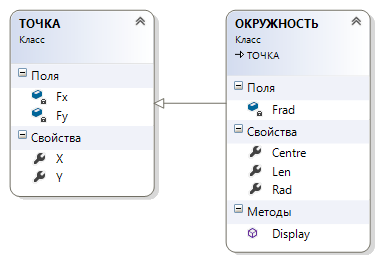
*Рис. 3.2-2. Программный код событийной процедуры   
при наследовании классов*

В событийной процедуре создан объект класса **ОКРУЖНОСТЬ**. Он ассоциирован со ссылкой **rim**, и с ее помощью осуществляется доступ как к свойствам и методам объекта класса **ОКРУЖНОСТЬ**, так и к свойствам объекта базового класса **ТОЧКА**. На форме присутствуют два текстовых поля. В первое текстовое поле выводятся характеристики объекта класса сразу после его создания (значения по умолчанию), а во второе – после изменения этих значений через свойства. Результаты работы программы приведены на   
рис. 3.2-3.



*Рис. 3.2-3. Результаты наследования классов:   
значения полей производного класса по умолчанию   
и изменение этих значений через свойства*

В языке UML предусмотрено специальное обозначение для отношения наследования классов, которое также называют обобщением, так как базовый класс – это более общая форма его производных классов. Два класса изображаются на Схеме отдельными прямоугольниками, которые соединены сплошной линией со стрелкой на конце. Стрелка направлена от класса-наследника на базовый класс, что подчеркивает тот факт, что производный класс ссылается на методы и поля базового класса, но при этом базовый класс не имеет доступа к наследнику (рис.3.2-4).



*Рис. 3.2-4. Схема наследования классов* ***ТОЧКА*** *и* ***ОКРУЖНОСТЬ***

***Примера 3.2-1***

#### 3.2.2. Доступность членов класса при наследовании

Очень важным в наследовании является знание того, как для объектов производного класса могут быть использованы члены базового класса, т.е. доступность членов класса при наследовании. Рассмотрим подробнее доступность членов класса при наследовании.

Как мы уже говорили, закрытые (**Private**) члены класса доступны только из своего класса, а для производного класса остаются закрытыми (недоступны). Открытые (**Public**) члены базового класса не имеют ограничений доступа, т.е. доступны для любой внешней процедуры и любого объекта программы, в том числе, естественно, и для объекта класса-наследника.

Иногда необходимо, чтобы члены базового класса были доступны для членов класса-наследника, но в то же время закрыты для всех других внешних классов и процедур, не принадлежащих классам-наследникам. В этом случае в базовом классе эти члены должны быть защищёнными, т.е. объявлены с модификатором **Protected**. Таким образом, если класс рассматривается вне наследования, то защищённые члены класса ничем не отличаются от закрытых членов класса. К защищённым членам класса нельзя обращаться извне объявления класса.

В производном классе обычно вводятся новые члены, определяющие новое поведение и дополнительные характеристики объектов производного класса. Для новых (не унаследованных) членов производных классов имена выбираются произвольно и иногда могут совпадать с именами членов базового класса. Для того, чтобы их различать, в производном классе используются уточненные имена:

|  |
| --- |
| **MyClass.***имя\_члена производного класса*  **MyBase.***имя\_члена базового класса* |

При внешних обращениях одноименные члены базового и производного классов различаются по типам объектов, для которых эти обращения выполнены.

**Пример 3.2-2.** **Описать базовый класс КРУГ с защищёнными членами и производный класс КОЛЬЦО.**

Программный код класса **КРУГ** представлен на рис.3.2-5.

|  |
| --- |
| **Option Strict On**  **Imports System.Math**  *'Базовый класс КРУГ*  **Public Class КРУГ**  *'Защищенные члены класса*  *'Поле - радиус круга*  **Protected Frad As Double**  *'Конструктор общего вида (с параметром)*  **Protected Sub New(ByVal ri As Double)**  **Frad = ri**  **End Sub**  *'Св-во для вычисления площади круга*  **Protected ReadOnly Property Area() As Double**  **Get**  **Return PI \* Frad ^ 2**  **End Get**  **End Property**  **End Class** |

*Рис. 3.2-5. Программный код базового класса* ***КРУГ****с защищёнными членами*

В классе **КРУГ** одно поле **Frad**, задающее значение радиуса круга, конструктор с параметром, который задает значение этого поля **Frad**, т.е. определяет радиус круга, и свойство для чтения **Area**, позволяющее получить значение площади круга. Все члены класса объявлены с модификатором **Protected**. При таком описании класс вне наследования ни к чему не годен. Невозможно создать объект класса **КРУГ** - его конструктор защищённый (**Protected**). Если убрать явно определённый конструктор, компилятор добавит открытый конструктор по умолчанию. Но и в этом случае пользы не видно – создав объект, нельзя будет обратиться к его полю или свойству.

Использование класса **КРУГ** в качестве базового для производного класса **КОЛЬЦО** показано на рис. 3.2-6.

|  |
| --- |
| **Option Strict On**  **Imports System.Math**  *'Производный класс КОЛЬЦО*  **Public Class КОЛЬЦО**  **Inherits КРУГ** *'наследник класса КРУГ*  *' ====================================================*  *'Закрытое поле - Радиус внутренней окружности*  *' Ключевое слово Shadows экранирует*  *'такое же поле базового класса*  **Private Shadows Frad As Double**  *' ==================================================*  *' Конструктор с параметрами*  **Public Sub New(ByVal Rout As Double, ByVal Rin As Double)**  *'Первым оператором в конструкторе потомка должен быть*  *'вызов конструктора базового класса КРУГ,*  *'в котором задается внешний радиус кольца(поле Frad)*  *' класса КРУГ через параметр Rout*  **MyBase.New(Rout)**  *'инициализация внутреннего радиуса кольца*  *'вторым параметром конструктора*  *'(заполнение поля Frad класса-наследника КОЛЬЦО)*  **Frad = Rin**  **End Sub**  *' ====================================================*  *'Св-во для вычисления площади кольца*  *' экранирует такое же свойство базового класса*  **Public Shadows ReadOnly Property Area() As Double**  **Get**  **Return MyBase.Area - PI \* Frad ^ 2**  **End Get**  **End Property**  *' ====Метод для вывода характеристик кольца=====*  *'собираем характеристики в одну строку s*  *' для вывода в текстовое поле*  **Public Sub Print(ByRef s As String)**  **s = "Кольцо: внешний радиус = "**  **s = s + CStr(MyBase.Frad) + vbNewLine**  **s = s + " Внутренний радиус = " + CStr(Frad) + vbNewLine**  **s = s + "Площадь кольца = " + CStr(Area)**  **End Sub**  **End Class** |

Рис. 3.2-6. Программный код класса **КОЛЬЦО**, производного от класса **КРУГ**

В производном классе **КОЛЬЦО** поле **Frad** с ключевым словом **Shadows** определяет значение внутренней окружности границы кольца. Радиус внешней границы определяет одноимённое поле **Frad**, унаследованное из базового класса. Оба поля вне объявления класса **КОЛЬЦО** недоступны.

Конструктор производного класса **КОЛЬЦО** объявлен явно, как открытый член класса. У этого конструктора два параметра, задающие значения радиусов границы кольца. В теле конструктора первым оператором обязательно должно быть обращение к конструктору базового класса **КРУГ** через уточненное имя **MyBase.New(Rout)**. Первый параметр **Rout** конструктора класса-наследника является аргументом при обращении к конструктору базового класса и будет задавать внешний радиус кольца. Второй параметр **Rin** определяет внутренний радиус и заполняет поле **Frad** класса-наследника.

Обратите внимание, что в объявлении поля **Frad** в производном классе **КОЛЬЦО** используется ключевое слово **Shadows**. В случае совпадения имени члена производного класса с именем какого-либо члена базового класса (как в нашем примере), член производного класса скрывает (экранирует, затеняет) соответствующее имя члена базового класса. Это ключевое слово **Shadows** как раз скрывает члена базового класса и подразумевается по умолчанию, т.е. необязательно. При отсутствии ключевого слова **Shadows** компиляция проходит успешно, но выдается предупреждение. Именно для того, чтобы удостоверить компилятор в преднамеренном совпадении имен, радиус внутренней окружности объявлен со словом **Shadows**. То же самое сделано и при объявлении в классе **КОЛЬЦО** свойства **Area**, позволяющего получить площадь кольца. Кроме того, класс **КОЛЬЦО** унаследовал свойство с тем же именем из базового класса. В части **Get** свойства **Area** из класса **КОЛЬЦО** выполнено явное обращение **MyBase.Area** к свойству базового класса **КРУГ**.

Открытый метод **Print()** позволяет вывести сведения об объекте класса **КОЛЬЦО**. Они выводятся как значения полей **MyBase.Frad**, **Frad** и свойства **Area**. Отметим, что принадлежность членов **Frad** и **Area** классу **КОЛЬЦО** можно подчеркнуть, если обратиться к ним следующим образом:

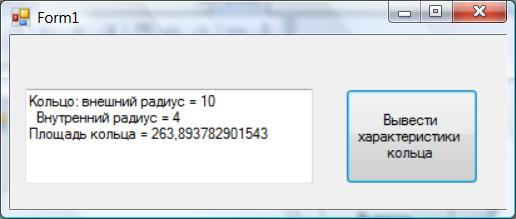
|  |
| --- |
| **s = s + " Внутренний радиус = " + CStr(MyClass.Frad)**  **s = s + "Площадь кольца = " + CStr(MyClass.Area)** |

Программный код событийной процедуры, иллюстрирующей возможности производного класса **КОЛЬЦО** приведен на рис. 3.2-7.

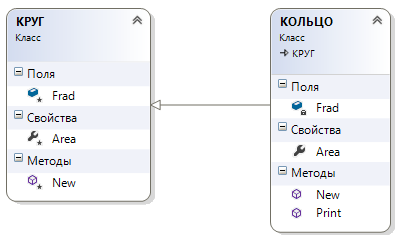
|  |
| --- |
| **Public Class Form1**  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button1.Click**  **Dim s As String = ""**  *'вызов конструктора класса-наследника с параметрами*  **Dim rim As New КОЛЬЦО(10, 4)**  **rim.Print(s)**  **TextBox1.Text = s**  **End Sub**  **End Class** |

*Рис. 3.2-7. Программный код событийной процедуры, которая   
создает объект производного класса* ***КОЛЬЦО*** *и вызывает его метод* ***Print()***

Результаты работы этой программы, и Схема классов представлены на рис. 3.2-8 и рис. 3.2-9.



*Рис. 3.2-8.Результаты работы метода* ***Print()*** *класса* ***КОЛЬЦО***



*Рис. 3.2-9. Схема классов: базовый класс* ***КРУГ*** *и производный класс* ***КОЛЬЦО***

#### 3.2.3. Конструкторы и методы при наследовании

Итак, следует обратить внимание на следующие важные аспекты, возникающие при наследовании классов:

* ***О конструкторах***. В приведенном примере наследования с классами **КРУГ** и **КОЛЬЦО** конструктор производного класса **КОЛЬЦО** явным образом обращается к конструктору базового класса **КРУГ** с помощью выражения **MyBase.New(Rout)**. Так как в нашем примере поле **Frad** базового класса **КРУГ** доступно для методов производного класса, то, казалось бы, можно попытаться записать определение конструктора класса **КОЛЬЦО** следующим образом:

|  |
| --- |
| **Public Sub New(ByVal Rout As Double, ByVal Rin As Double)**  **MyBase.Frad = Rout**  **Frad = Rin**  **End Sub** |

Внешне все выглядит правильно, но если не менять описание базового класса КРУГ, то компилятор выдаст сообщение об ошибке

|  |
| --- |
| **Первым оператором в Sub New должен быть вызов MyBase.New или MyClass.New, так как базовый класс "КРУГ" класса "КОЛЬЦО" не содержит доступную подпрограмму Sub New, которую можно вызывать без аргументов.** |

Такое сообщение компилятора обусловлено двумя правилами. Первое – если в определении класса присутствует объявление хотя бы одного конструктора, то конструктор без параметров (по умолчанию) автоматически в класс не добавляется. Второе – в отличие от других членов базового класса, конструкторы не наследуются. Конструктор базового класса необходимо явно вызывать в конструкторе производного класса. Если этого не сделать, то компилятор по умолчанию сам встроит вызов конструктора базового класса без параметров. Так как в классе КРУГ конструктор без параметров отсутствует, то компилятор выдает сообщение об ошибке. Именно по этой причине мы уже ранее   
(п. 2.2-1) упоминали о том, что правильный стиль программирования требует обязательного присутствия в классе конструктора без параметров в случае, если в нем уже есть конструктор с параметрами. Соблюдение этого правила необходимо, так как дает возможность классу-наследнику не заботиться о вызове конструктора базового класса, тем более, что наследник может и не иметь информации о параметрах, нужных конструктору базового класса.

* ***О методах****.* При наследовании в производном классе могут появляться новые методы, имена которых отличны от имен методов базового класса и никаких особенностей в их применении нет. Остановимся подробнее на одноименных методах базового и производного классов.

***Во-первых***, для методов возможна перегрузка (Overload). В этом случае одноименные методы имеют либо разный набор параметров (по их количеству), либо параметры имеют различный тип (говорят, что методы различаются сигнатурой). Такие методы принято отмечать ключевым словом **Overloads**, например, три перегружаемые процедуры для ввода данных разными способами могут иметь такие заголовки:

|  |
| --- |
| **Public Overloads Sub vvod()**  **. . .**  **End Sub**  **Public Overloads Sub vvod(ByVal mas() As String, \_**  **ByVal n As Integer, ByRef m As Integer)**  **. . .**  **End Sub**  **Public Overloads Sub vvod(ByVal mas() As Double, \_**  **ByVal n As Integer, ByRef m As Integer)**  **. . .**  **End Sub** |

Какая из перегружаемых процедур будет выполнена при вызове, зависит от количества и типа фактических параметров. Пример перегрузки конструкторов и методов мы рассмотрели в п.п. 2.2, 2.7 и также рассмотрим в п. 3.6. Заметим, что ключевое слово **Overloads** является необязательным и подразумевается по умолчанию при наличии в программе таких процедур, которые имеют одинаковые имена, но разный список формальных параметров. При наследовании никаких особенностей для перегружаемых методов не возникает.

***Во-вторых***, разрешено экранирование (скрытие, затенение) методом производного класса одноименного метода базового класса (сигнатуры методов полностью совпадают). При экранировании метода рекомендуется снабжать метод производного класса ключевым словом **Shadows**. При его отсутствии компиляция проходит успешно, но выдается предупреждение о возможном ошибочном скрытии метода базового класса. Если же требуется из какого-либо метода производного класса обратиться к скрытому методу базового класса, то используется уточненное имя:

|  |
| --- |
| **MyBase*.****имя\_метода\_базового\_класса* |

***В-третьих***, метод может быть объявлен виртуальным (с ключевым словом **Overridable**) и тогда при его переопределении (с ключевым словом **Overrides**) в производных классах возникает полиморфизм. Рассмотрим разницу между экранированием и переопределением методов.

#### 3.2.4. Виртуальные методы и полиморфизм

Для того, чтобы понять один из основных принципов ООП – полиморфизм, сначала рассмотрим разницу между экранированием и переопределением методов.

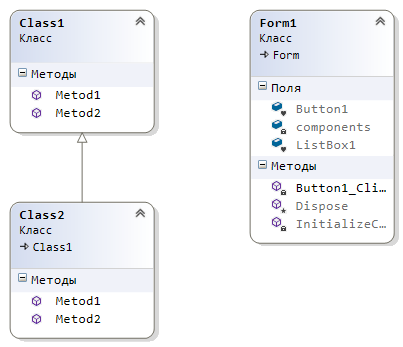
**Пример 3.2-3. Описать базовый класс Class1 с обычным и виртуальным методами и производный класс Class2 с экранирующим и переопределенным методами.**

Программный код базового и производного классов, а также событийной процедуры представлен на рис.3.2-10, а Схема классов на   
рис.3.2-11

|  |
| --- |
| *'Базовый класс*  **Public Class Class1**  *'обычный метод*  **Public Sub Metod1(ByRef s As String)**  **s = "Метод1 базового класса"**  **End Sub**  *'виртуальный метод*  **Public Overridable Sub Metod2(ByRef s As String)**  **s = "Виртуальный(переопределяемый) Метод2 базового класса"**  **End Sub**  **End Class**  *'Производный класс*  **Public Class Class2**  **Inherits Class1** *'Class2 - наследник класса Class1*  *'Экранирование метода базового класса*  **Public Shadows Sub Metod1(ByRef s As String)**  **s = "Метод1 производного класса"**  **End Sub**  *'Переопределение метода базового класса*  **Public Overrides Sub Metod2(ByRef s As String)**  **s = "Переопределённый Метод2 производного класса"**  **End Sub**  **End Class**  *'событийная процедура*  **Public Class Form1**  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button1.Click**  **Dim s As String = ""**  **Dim c2 As New Class2***'создание экземпляра класса-наследника*  **Dim c1 As Class1** *'объявление переменной базового класса*  *' без создания экземпляра*  **c1 = c2** *'присваивание переменной базового класса*  *'значения ссылки на экземпляр наследника*  **c1.Metod1(s) : ListBox1.Items.Add(s)**  **c2.Metod1(s) : ListBox1.Items.Add(s)**  **c1.Metod2(s) : ListBox1.Items.Add(s)**  **c2.Metod2(s) : ListBox1.Items.Add(s)**  **End Sub**  **End Class** |

*Рис. 3.2-10. Экранирование и переопределение   
методов базового класса в классе-наследнике*

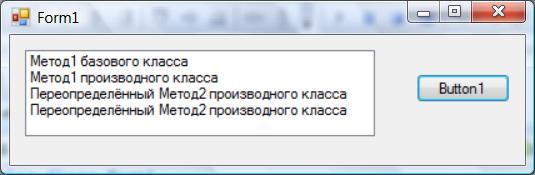
В примере объявляется два класса с методами, имеющими одинаковое название, причем в классе-наследнике один из методов скрывает, а другой переопределяет метод базового класса. После создания экземпляра наследника ссылка на него присваивается переменной, имеющей тип базового класса. Это всегда можно сделать, т.к. класс-наследник гарантированно не уменьшает набор членов базового класса, т.е. то, что доступно в базовом классе, гарантированно доступно и в производном.



*Рис. 3.2-11. Схема классов* ***Примера 3.2-3***

Результаты работы программы представлены на рис.3.2-12.

Таким образом, ссылке с типом базового класса можно присвоить значение ссылки на объект производного класса. Однако после такого присваивания ссылка не обеспечивает доступ к обычным (не виртуальным) методам производного класса, а вызываются методы базового класса. Это показывает первая строка результатов на рис. 3.2-12.



*Рис. 3.2-12. Результаты экранирования и переопределения   
методов базового класса в классе-наследнике*

Однако, как видно из третьей строки этого же рисунка, при работе с виртуальным методом вызывается именно переопределенный метод производного класса. Такие результаты объясняются механизмом вызова виртуальных (**Overridable**) методов. Чтобы лучше понять разницу между обычными и виртуальными методами, рассмотрим еще два примера.

**Пример 3.2-4. Описать базовый класс и два производных класса с не виртуальными методами и событийную процедуру с несколькими ссылками базового класса, адресующими объекты и базового, и производных классов.**

Программный код классов, а также текст событийной процедуры представлены на рис. 3.2-13.

|  |
| --- |
| *'Базовый класс*  **Public Class Class1**  **Protected x, y As Double** *'защищенные поля*  **Public Function print() As String** *'обычный метод*  **Dim s As String**  **s = "Базовый класс x=" + CStr(x) + " y=" + CStr(y)**  **Return s**  **End Function**  **End Class**  *' 1 наследник*  **Public Class Class2**  **Inherits Class1**  *'Конструктор 1-го наследника*  **Public Sub New(ByVal xi As Double, ByVal yi As Double)**  **x = xi**  **y = yi**  **End Sub**  *'Экранирование**метода базового класса*  **Public Shadows Function print() As String**  **Dim s As String**  **s = "Class2 x=" + CStr(x) + " y=" + CStr(y)**  **s = s + MyBase.print()**  **Return s**  **End Function**  **End Class**  *' 2 наследник*  **Public Class Class3**  **Inherits Class1**  *'Конструктор 2-го наследника*  **Public Sub New(ByVal xi As Double, ByVal yi As Double)**  **x = xi : y = yi**  **End Sub**  *'Экранирование метода базового класса*  **Public Shadows Function print() As String**  **Dim s As String**  **s = "Class3 x=" + CStr(x) + " y=" + CStr(y)**  **s = s + MyBase.print()**  **Return s**  **End Function**  **End Class** |

*Рис. 3.2-13. Экранирование обычного метода базового класса   
в классах-наследниках*

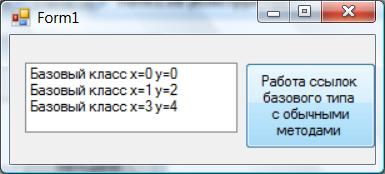
В каждом из производных классов есть свой метод **print()**, который экранирует одноимённый метод базового класса. В методе **print()** производного класса выполнено обращение к методу **print()** базового класса.

В событийной процедуре (рис.3.2-14) три ссылки **c1**, **c2** и **c3**, имеющие тип базового класса **Class1** ассоциированы с объектами разных классов. Затем с помощью этих ссылок выполнены обращения к методу **print()**.

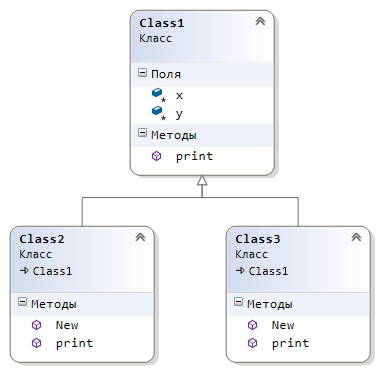
|  |
| --- |
| **Public Class Form1**  *'событийная процедура*  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button1.Click**  **Dim c1 As Class1 = New Class1**  **Dim c2 As Class1 = New Class2(1, 2)**  **Dim c3 As Class1 = New Class3(3, 4)**  **Dim s As String = ""**  **s = c1.print() : ListBox1.Items.Add(s)**  **s = c2.print() : ListBox1.Items.Add(s)**  **s = c3.print() : ListBox1.Items.Add(s)**  **End Sub**  **End Class** |

*Рис. 3.2-14. Программный код процедуры, в которой ссылкам типа   
базового класса присваиваются объекты производных классов*

Результаты работы этой программы представлены на рис. 3.2-15, а Схема классов на рис. 3.2-16.



*Рис. 3.2-15. Результаты работы ссылок   
базового типа с обычными методами*



*Рис. 3.2-16. Схема классов Примера 3.2-4*

Как видно из полученных результатов, присваивать ссылке с типом базового класса объект производного класса можно, но вызываются для него только методы и свойства, определённые в базовом классе. То есть, возможность доступа к членам класса определяется типом ссылки, а не типом объекта, на который она указывает. Это понятно: ведь для вызова любой процедуры надо знать ее адрес в памяти. Для обычных методов компилятор сразу записывает в машинный код нужный адрес, потому что он заранее известен – это так называемое *статическое* *(раннее)* связывание (на этапе *компиляции*), при выполнении программы этот адрес не меняется. При этом компилятор может руководствоваться только типом переменной, для которой вызывается метод. То, что в этой переменной в разные моменты времени могут находиться ссылки на объекты разных типов, компилятор учесть не может.

Следовательно, если мы хотим, чтобы вызываемые методы соответствовали типу объекта, необходимо отложить процесс связывания до этапа выполнения программы, точнее – до момента вызова метода, когда уже точно известно, на объект какого типа указывает ссылка. Такой механизм называется *динамическим (поздним)* связыванием и реализуется с помощью виртуальных методов, когда адрес вызываемой процедуры определяется при *выполнении* программы, когда уже определен тип объекта, с которым работает метод, т.е. тип фактического параметра. Таким образом, виртуальный метод – это метод базового класса, который могут переопределить классы-наследники так, что что конкретный адрес вызываемого метода определяется только при выполнении программы.

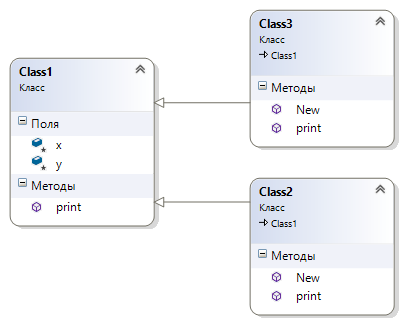
**Пример 3.2-5. Описать базовый класс с виртуальным методом и два производных класса, в которых переопределяется этот виртуальный метод и событийную процедуру с несколькими ссылками базового класса, адресующими объекты и базового, и производных классов.**

Напомним, для определения в базовом классе виртуального метода, в его заголовок нужно добавить модификатор **Overridable**. В производном классе для переопределения виртуального метода используется модификатор **Overrides**. Заметим, что виртуальный метод не может быть закрытым (**Private**), а переопределенный метод должен обладать таким же набором параметров, как и одноименный метод базового класса. В нашем примере по сравнению с предыдущим (на рис. 3.2-13) изменения минимальны. Заголовок метода в базовом классе примет вид:

|  |
| --- |
| *'Виртуальный метод*  **Public Overridable Function print() As String** |

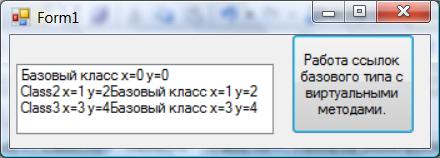
В каждом из производных классов заголовок переопределяющего метода примет вид:

|  |
| --- |
| *'Переопределенный метод*  **Public Overrides Function print() As String** |



*Рис. 3.2-17. Схема классов* ***Class1, Class2*** *и* ***Class3*** *с виртуальным и переопределёнными методами* ***print()***

Результат работы такой же программы, как на рис. 3.2-14 со ссылками **c1**, **c2** и **c3** показан на рис. 3.2-18.



*Рис. 3.2-18. Результаты работы ссылок базового типа   
с виртуальными и переопределёнными методами*

Здесь сначала выполнен вызов виртуального метода **print()** базового класса, а затем дважды выполнено обращение к методам **print()** производных классов. Как видно, теперь при выполнении программы вызывается метод, соответствующий фактическому типу вызвавшего его объекта.

Теперь можно легко объяснить результаты **Примера 3.2-3** на   
рис. 3.2-12. Так как переменной типа базового классабыла присвоена ссылка на объект производного класса, то фактическим объектом, вызывающим виртуальный метод **Metod2()** в обоих случаях, является объект производного класса **Class2**, что и отразилось в двух последних строках результата на рис. 3.2-12.

Именно с помощью виртуальных методов реализуется один из основных принципов ООП – *полиморфизм.*Это слово в переводе с греческого означает «много форм», что в данном случае значит «один вызов – много методов». Применение виртуальных методов обеспечивает гибкость и возможность расширения функциональности класса. Также виртуальные методы незаменимы и при передаче объектов в методы в качестве параметра. Формальным параметром метода описывается объект базового класса, а при вызове в него передается в качестве фактического параметра объект производного (наследующего) класса и по типу этого фактического параметра определяется какой конкретно метод будет реализован. При описании классов рекомендуется определять в качестве виртуальных те методы, которые в производных классах должны реализовываться по-другому. Если во всех классах иерархии метод будет выполняться одинаково, его лучше определить как обычный метод.

### 3.3. Абстрактные классы

При создании иерархии объектов для исключения повторяющегося кода часто бывает логично выделить их общие свойства в один родительский класс. При этом может оказаться, что создавать экземпляры такого класса не имеет смысла, потому что никакие реальные объекты им не соответствуют. Такие классы называют абстрактными. *Абстрактный* класс создается только как базовый и служит только для наследования (порождения потомков). Как правило, в нем задается набор виртуальных абстрактных методов, которые каждый из потомков будет реализовывать по-своему. Абстрактные классы предназначены для представления общих понятий, которые предполагается конкретизировать в производных классах.

**Пример 3.3-1. Создать программный проект для моделирования управляющих схем, построенных на логических элементах. Необходимо «собрать» заданную схему и построить ее таблицу истинности [11].**

Выполним объектно-ориентированный анализ. Все объекты, из которых состоит схема, – это логические элементы, однако они могут быть разными («НЕ», «И», «ИЛИ» и другие). Выделим общие атрибуты и методы всех логических элементов.

Ограничимся только элементами, у которых один или два входа. Среди всех элементов с двумя входами покажем только элементы «И» и «ИЛИ». Тогда иерархия классов может выглядеть так (рис.3.3-1):



*Рис. 3.3-1. Упрощённая иерархия классов логических элементов*

Определим класс **ЛОГИЧЕСКИЙ\_ЭЛЕМЕНТ (TLogElement)**. Обозначим его первый вход как поле **FIn1**, а выход (результат логической операции) – как поле **FRez**. Так как этот класс будет использоваться для логических элементов и с одним, и с двумя входами, пока не будем создавать в нем еще одно поле для второго входа.

Любой логический элемент должен уметь вычислять значение выхода по известным входам, для этого введем метод **calc()**. Мы должны объявить этот метод (ввести его в описание класса), поскольку он должен быть у любого логического элемента. С другой стороны, **невозможно** написать процедуру **calc()**, пока мы не знаем, какой именно логический элемент моделируется. Такой метод, который только объявляется, но не реализуется в классе, называется *абстрактным* и должен быть объявлен с ключевым словом **MustOverride**, т.е. метод не определен в данном классе и должен быть **переопределен** в наследующих классах (поэтому абстрактный метод по умолчанию является виртуальным). Класс, который содержит хотя бы один абстрактный метод, также является *абстрактным* и должен быть объявлен с модификатором **MustInherit**. Таким образом, объявление метода с ключевым словом **MustOverride** требует, чтобы класс был помечен ключевым словом **MustInherit** и автоматически запрещает создание экземпляров такого класса, класс можно использовать только для разработки классов - наследников. Чтобы класс-наследник не был абстрактным, он должен переопределить все абстрактные методы предка, в нашем случае метод **calc()**. Программный код абстрактного класса **ЛОГИЧЕСКИЙ\_ЭЛЕМЕНТ (TLogElement)** приведен на рис.3.3-2.

|  |
| --- |
| *'Абстрактный класс,нельзя создавать его экземпляры*  **Public MustInherit Class TLogElement**  *'Поля*  **Protected FIn1 As Boolean** *'1-ый вход*  **Protected FRez As Boolean** *'результат*  *'Абстрактный метод, к-рый д.б. переопределен в наследниках*  **MustOverride Sub calc()**  *' Свойства*  **Public Property In1() As Boolean**  **Get**  **Return FIn1**  **End Get**  **Set(ByVal value As Boolean)**  *'при установке(записи) нового значения в поле*  *'одновременно тут же пересчитывается поле рез-та*  **FIn1 = value**  **calc()**  **End Set**  **End Property**  **Public ReadOnly Property Rez() As Boolean**  **Get**  **Return FRez**  **End Get**  **End Property**  **End Class** |

*Рис. 3.3-2. Программный код абстрактного класса* **TLogElement**

В этом классе поля **FIn1** (для первого входа) и **FRez** (для выхода, т.е. результата логической операции) объявлены как защищенные   
(**Protected**), чтобы значения этих полей были доступны в наследниках. Для доступа к полю **FIn1** описано открытое свойство **In1**, в части **Set** которого (т.е. при присвоении полю **FIn1** нового значения), вызывается метод **calc()** для пересчета выходного значения (поля результата **FRez**). Для доступа к полю **FRez** предназначено свойство **Rez** – это свойство только для чтения (**ReadOnly**), так как значение этого поля должно вычисляться внутренним методом **calc()** класса и изменять его снаружи нельзя. Но поскольку сам метод **calc()** реализуется только в наследниках, то изменение значения этого поля должно быть доступно наследникам, и поэтому поле объявлено как **Protected**.

Теперь займемся классами – наследниками от **TLogElement**. Поскольку у нас единственный элемент с одним входом (**«НЕ»**), сделаем его наследником прямо от **TLogElement** (не будем вводить отдельный класс **ЛЭ\_С\_ОДНИМ\_ВХОДОМ)**. Программный код производного класса **TNot («НЕ»)** приведен на рис.3.3-3.

|  |
| --- |
| **Public Class TNot** *'класс для логического «НЕ»*  **Inherits TLogElement** *'наследник класса TLogElement*  *'переопределение абстрактного метода*  **Public Overrides Sub calc()**  **FRez = Not (FIn1)**  **End Sub**  **End Class** |

*Рис. 3.3-3. Программный код производного класса* **TNot**

В этом классе переопределен абстрактный метод базового класса **calc()** (вернее, запрограммирован, так как в базовом классе метод только объявлен и не имеет программного кода). Класс **TNot** уже не абстрактный, и теперь известно, что делать при вызове метода **calc()**, значит можно создавать и использовать объекты этого класса.

Остальные логические элементы нашей иерархии имеют два входа и будут наследниками класса **TLog2In** (**ЛЭ\_С\_ДВУМЯ\_ВХОДАМИ**). В этом классе, производном от класса **TLogElement,** объявляется еще одно защищенное поле **FIn2** для второго входа и открытое свойство **In2** для доступа к этому полю (рис.3.3-4).

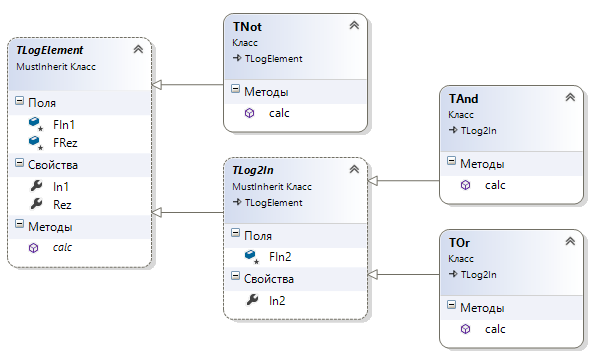
Класс **TLog2In** тоже абстрактный, так как он не переопределил метод **calc()**. Это сделают его наследники **TAnd** (**ЛЭ «И»**) и **TOr** (**ЛЭ «ИЛИ»**), которые определяют конкретные логические элементы (рис.3.3-5).

|  |
| --- |
| **Public MustInherit Class TLog2In**  *'класс для лог. эл-тов с 2-мя входами*  **Inherits TLogElement** *'наследник класса TLogElement*  **Protected FIn2 As Boolean** *'2-ой вход*  **Public Property In2() As Boolean**  **Get**  **Return FIn2**  **End Get**  *'при установке(записи) нового значения в поле*  *'одновременно тут же пересчитывается поле рез-та*  **Set(ByVal value As Boolean)**  **FIn2 = value**  **calc()**  **End Set**  **End Property**  **End Class** |

*Рис. 3.3-4. Программный код производного абстрактного класса* **TLog2In** *для элементов с двумя входами*

|  |
| --- |
| *' класс для лог. эл-та "И"*  **Public Class TAnd**  **Inherits TLog2In***'наследник класса TLog2In*  *'переопределение абстрактного метода*  **Public Overrides Sub calc()**  **FRez = FIn1 And FIn2**  **End Sub**  **End Class**  *' класс для лог. эл-та "ИЛИ"*  **Public Class TOr**  **Inherits TLog2In** *'наследник класса TLog2In*  *'переопределение абстрактного метода*  **Public Overrides Sub calc()**  **FRez = FIn1 Or FIn2**  **End Sub**  **End Class** |

*Рис. 3.3-5. Программный код производных классов* **TAnd** *и* **TOr**



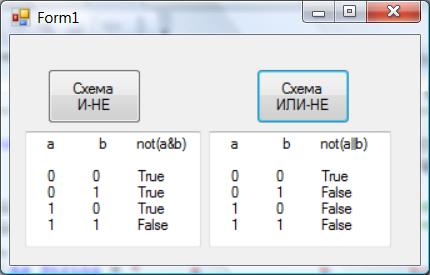
*Рис.3.3-6. Схема классов логических элементов*

Схема объявленных классов представлена на рис. 3.3-6.

Теперь можно создавать и использовать построенные логические элементы. Например, таблицу истинности для последовательного соединения элементов **«И»** и **«НЕ»** можно построить так, как показано в программном коде событийной процедуры нажатия на кнопку **Button1**, а таблицу истинности для последовательного соединения элементов **«ИЛИ»** и **«НЕ»** – как в событийной процедуре нажатия на кнопку **Button2** (рис. 3.3-7):

|  |
| --- |
| **Option Strict On**  **Public Class Form1**  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button1.Click**  **Dim elemNot As New TNot** *'создан объект - лог. эл-т "НЕ"*  **Dim elemAnd As New TAnd** *'создан объект - лог. эл-т "И"*  **Dim a, b As Integer**  *' строка для вывода в TextBox с заголовком*  **Dim z As String = " a b not(a&b) "**  **z = z & vbNewLine & vbNewLine**  **For a = 0 To 1**  **elemAnd.In1 = CBool(a)**  **For b = 0 To 1**  **elemAnd.In2 = CBool(b) : elemNot.In1 = elemAnd.Rez**  **z = z & Space(5) & (a) & Space(10) & (b) & \_**  **Space(10) & elemNot.Rez & vbNewLine**  **Next**  **Next**  **TextBox1.Text = z** *' вывод рез-та*  **End Sub**    **Private Sub Button2\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button2.Click**  **Dim elemNot As New TNot** *'создан объект - лог. эл-т "НЕ"*  **Dim elemOr As New TOr** *'создан объект - лог. эл-т "ИЛИ"*  **Dim a, b As Integer**  **Dim z As String = " a b not(a||b) "**  **z = z & vbNewLine & vbNewLine**  **For a = 0 To 1**  **elemOr.In1 = CBool(a)**  **For b = 0 To 1**  **elemOr.In2 = CBool(b) : elemNot.In1 = elemOr.Rez**  **z = z & Space(5) & (a) & Space(10) & (b) & \_**  **Space(10) & elemNot.Rez & vbNewLine**  **Next**  **Next**  **TextBox2.Text = z**  **End Sub**  **End Class** |

*Рис. 3.3-7. Программный код построения таблицы истинности   
при последовательном соединении логических элементов* ***И-НЕ*** *и* ***ИЛИ-НЕ***



*Рис. 3.3-8. Результаты построения таблицы истинности   
при последовательном соединении логических элементов* ***И-НЕ*** *и* ***ИЛИ-НЕ***

В событийной процедуре нажатия на **Button1** сначала создаются два объекта – логические элементы **elemNot** (элемент **«НЕ»** класса **TNot**) и **elemAnd** (элемент **«И»** класса **TAnd**). Затем в двойном цикле перебираются все возможные значения целочисленных переменных **a** и **b**, которые преобразуются к логическому типу и подаются на входы элемента **«И»**, а выход «**И»** – на вход элемента **«НЕ»**. Результаты такого соединения «склеиваются» в одну строку **z**, которая выводится в текстовое поле **TextBox1**. Аналогично получена таблица истинности для соединения **«ИЛИ»** и **«НЕ»** при нажатии на **Button2**. Результаты работы программы представлены на рис. 3.3-8.

Абстрактные классы используются при работе со структурами данных, предназначенными для хранения объектов одной иерархии, и в качестве параметров методов. Мы уже отметили, что нельзя создать объект абстрактного класса. Однако можно объявить ссылку с типом абстрактного класса. Такая ссылка может служить параметром метода. На место этого параметра при выполнении программы может передаваться объект любого производного класса. Это позволяет создавать *полиморфные* методы, работающие с объектом любого типа в пределах одной иерархии. Также очень сильным средством полиморфизма является возможность присваивать элементам массива с типом абстрактного класса ссылок на объекты любых классов – наследников абстрактного. Полиморфизм в различных формах является мощным и широко применяемым инструментом ООП. Рассмотрим пример использования абстрактного класса в полиморфных методах.

**Пример 3.3-2. Создать проект, который содержит абстрактный класс ФУНКЦИЯ** **с методом вычисления функции y=f(x) в заданной точке и два производных класса ЛИНИЯ (y=ax+b) и ПАРАБОЛА (y=ax2+bx+c) со своими методами вычисления значения в заданной точке. Создать массив n функций и вывести полную информацию о значении данных функций в заданной точке x.**

Программный код проекта **Примера** **3.3-2** приведен на рис. 3.3-9.

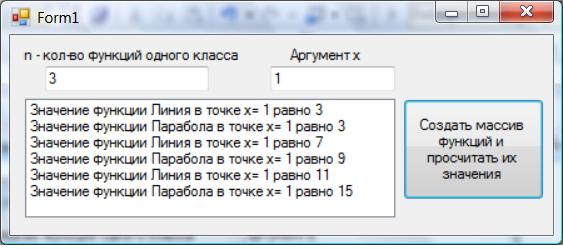
|  |
| --- |
| *'Абстрактный класс*  **Public MustInherit Class ФУНКЦИЯ**  *'Метод вычисления значения функции*  **Public MustOverride Function f() As Double**  *'Метод, возвращающий название функции*  **Public MustOverride Function GetName() As String**  *'Поля*  **Protected Fx, Fa, Fb As Double**  *'Свойства*  **Public Property x() As Double**  **Get**  **Return Fx**  **End Get**  **Set(ByVal value As Double)**  **Fx = value**  **End Set**  **End Property**  **Public Property a() As Double**  **Get**  **Return Fa**  **End Get**  **Set(ByVal value As Double)**  **Fa = value**  **End Set**  **End Property**  **Public Property b() As Double**  **Get**  **Return Fb**  **End Get**  **Set(ByVal value As Double)**  **Fb = value**  **End Set**  **End Property**  **End Class**  **Public Class ЛИНИЯ**  **Inherits ФУНКЦИЯ**  *'конструкторы*  **Public Sub New(ByVal x As Double, ByVal a As Double, \_**  **ByVal b As Double)**  **Fx = x : Fa = a : Fb = b**  **End Sub**  **Public Sub New()**  **End Sub**  *'переопределение абстрактных методов*  **Budlic Overrides Function GetName() As String**  **Return "Линия"**  **End Function**    **Public Overrides Function f() As Double**  **Return a \* x + b**  **End Function**  **End Class**  **Public Class ПАРАБОЛА**  **Inherits ФУНКЦИЯ**  *'Дополнительное поле*  **Protected Fc As Double**  *'конструкторы*  **Public Sub New(ByVal x As Double, ByVal a As Double, \_**  **ByVal b As Double, ByVal c As Double)**  **Fx = x : Fa = a : Fb = b : Fc = c**  **End Sub**    **Public Sub New()**  **End Sub**  **Public Property c() As Double**  **Get**  **Return Fc**  **End Get**  **Set(ByVal value As Double)**  **Fc = value**  **End Set**  **End Property**  *'переопределение абстрактных методов*  **Public Overrides Function f() As Double**  **Return a \* x ^ 2 + b \* x + c**  **End Function**  **Public Overrides Function GetName() As String**  **Return "Парабола"**  **End Function**  **End Class**  **Public Class Form1**  *'Метод вывода информации об объекте в ListBox,*  *'параметром является ссылка с типом абстрактного класса*  **Sub INFO(ByRef FUNC As ФУНКЦИЯ)**  **Dim z As String**  **z = "Значение функции " & FUNC.GetName() & \_**  **" в точке x= " & FUNC.x & " равно " & FUNC.f()**  **ListBox1.Items.Add(z)**  **End Sub**  *'Метод возвращает ссылку с типом абстрактного класса,*  *'параметром является массив ссылок с типом абстрактного класса*  **Function RecFunc(ByRef mas() As ФУНКЦИЯ, \_**  **ByVal m As Integer) As ФУНКЦИЯ**  **Return mas(m)**  **End Function**  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button1.Click**  **Dim n, i As Integer**  **n = CInt(TextBox1.Text)**  *'кол-во функций в массиве удвоено*  **n = n \* 2**  *'массив функций*  **Dim MAS(n - 1) As ФУНКЦИЯ**  **Dim x As Double** *'заданная точка*  **x = CDbl(TextBox2.Text)**  *'Цикл создания массива объектов-функций*  *'с инициализацией полей*  **For i = 0 To n - 1 Step 2**  **MAS(i) = New ЛИНИЯ(x, 1 + i, 2 + i)**  **MAS(i + 1) = New ПАРАБОЛА(x, 1 + i, 2 + i, i)**  **Next**  **Dim FF As ФУНКЦИЯ**  *'цикл вывода информации об элементах массива*  **For i = 0 To n - 1**  **FF = RecFunc(MAS, i)**  **INFO(FF)**  **Next**  **End Sub**  **End Class** |

*Рис. 3.3-9. Программный код проекта* ***Примера******3.3-2****абстрактного класса* ***ФУНКЦИЯ****,   
производных классов* ***ЛИНИЯ*** *и* ***ПАРАБОЛА*** *и класса формы*

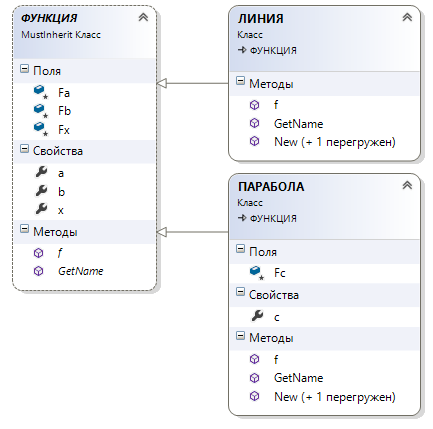
В классе формы определен метод **INFO** вывода информации о любом объекте классов **ЛИНИЯ** и **ПАРАБОЛА**, так как параметром этого метода является ссылка с типом базового абстрактного класса **ФУНКЦИЯ**. Также для демонстрации возможностей массивов ссылок с типами абстрактных классов описан метод **RecFunc**, параметрами которого являются такой массив ссылок и целое число **m**, определяющее индекс элемента этого массива. Этот метод возвращает значение элемента массива. В событийной процедуре определен массив ссылок с типом абстрактного класса. Значениями этих ссылок должны быть адреса объектов конкретных классов, реализующих базовый класс. Эти объекты создаются конструкторами в первом цикле и ссылки на них присваиваются элементам массива. Во втором цикле переменной **FF** с типом абстрактного класса **ФУНКЦИЯ** присваиваются значения, возвращаемые методом **RecFunc**, а затем **FF** используется в качестве фактического параметра метода **INFO**. На форме имеется два текстовых поля для ввода аргумента функций x и количества функций в массиве, а также элемент управления **ListBox**, в который выводится информация об элементах созданного массива. Результаты работы приведены на рис. 3.3-10, а Схема классов – на рис. 3.3-11.

Заметим, что вывод значений функций во втором цикле можно выполнить, не прибегая к методу **RecFunc**, который просто демонстрирует возможности работы с массивом ссылок, а непосредственно обращаясь к элементам массива. Тогда цикл вывода информации примет вид:

|  |
| --- |
| **For i = 0 To n - 1**  **INFO(MAS(i))**  **Next** |



*Рис. 3.3-10. Результаты работы проекта* ***Примера 3.3-2*** *с массивом ссылок абстрактного класса*



*Рис.3.3-11. Схема классов проекта* ***Примера 3.3-2***

Завершая рассмотрение методов при наследовании классов, еще раз напомним, что ***в производном классе метод по отношению к методу базового класса может быть новым, перегруженным, унаследованным, скрывающим (экранирующим) метод базового класса и переопределяющим (реализующим) виртуальный или абстрактный метод базового класса.***

### 3.4. Интерфейсные классы

***Интерфейсный класс*** или просто интерфейс является «крайним случаем» абстрактного класса. В нем задается набор абстрактных методов и свойств, которые должны быть реализованы в производных классах. Сам интерфейс не реализует никакие методы. Таким образом, интерфейс является логической конструкцией, которая описывает функциональность классов, не определяя способов её реализации, т.е. описывает, какие действия нужны для объектов классов, но не определяет, как эти действия должны выполняться. Иными словами, интерфейс определяет правила поведения объектов ещё не существующих классов, которые будут поддерживаться реализующими этот интерфейс классами. Основная идея использования интерфейса состоит в том, чтобы к объектам таких классов можно было обращаться одинаковым образом.

В отличие от классов, с помощью интерфейсов нельзя определять объекты.

На основе интерфейса объявляются новые классы, и при этом используется механизм наследования. Говорят, что класс, построенный на базе интерфейса, реализует данный интерфейс. Чтобы реализовать интерфейс, класс должен определить тело (реализацию) каждого метода, описанного в этом интерфейсе.

На базе одного интерфейса могут быть созданы несколько классов, и у каждого из этих классов будет набор всех средств, объявленных в интерфейсе. Однако каждый такой класс может по-своему определять эти средства. Таким образом, два класса могут реализовать один и тот же интерфейс различными способами, но каждый класс должен поддерживать один и тот же набор методов. Следовательно, программист, которому известно об интерфейсе, знает средства взаимодействия с объектами любого класса, реализовавшего этот интерфейс, и поэтому объекты разных классов, реализующих один интерфейс, могут обрабатываться одинаково, хотя по-разному реагируют на вызовы одного и того же метода. Это наряду с перегрузкой методов ещё один пример проявления полиморфизма.

В проекте VS интерфейс создается аналогично стандартному модулю и классу. То есть, чтобы создать в проекте новый интерфейс, необходимо выбрать команду ***Добавить Новый элемент…***или ***Компонент…*** элемента Главного меню **Проект**. В результате выполнения одной из этих команд откроется диалоговое окно **Добавление нового элемента …**, в котором следует выб­рать шаблон ***Интерфейс*** и указать имя интерфейса.

После нажатия на кнопку ***Добавить*** в окне **Редактор кода** по­явится новый интерфейс:

|  |
| --- |
| **Public Interface** **Interface1** *' Объявление Имени интерфейса*  **…**  **End Interface** |

**Пример 3.4-1. Создать интерфейс класса, который генерирует последовательность целых чисел для числовых рядов, и реализовать его в двух классах. [14].**

Дадим интерфейсу имя **ISeries**. Хотя префикс **I** необязателен, многие программисты его используют, чтобы отличать интерфейсы от классов. В этом интерфейсе предусмотрен метод **start()** для установки начального значения ряда (и восстановления начального состояния) и свойство только для чтения **GetNext**, которое позволит получить значение очередного члена ряда и настроить объект на следующий член (рис. 3.4-1).

|  |
| --- |
| *'Объявление интерфейса*  **Public Interface ISeries**  *'метод**для установки начального значения*  **Sub start()**  *' св-во для получения очередного члена ряда*  **ReadOnly Property GetNext() As Integer**  **End Interface** |

*Рис. 3.4-1.Программный код интерфейса* **ISeries** *для числовых рядов*

В интерфейсе методы и свойства неявно определены как открытые (**Public**), и для них нельзя явно указывать любые модификаторы доступа. Те функциональные возможности, которые приписаны членам интерфейса **ISeries**, при реализации в конкретных классах могут быть изменены. Но в этом случае нарушается общий принцип применения интерфейсов. Поэтому в примерах конкретных классов будем придерживаться описанных соглашений о ролях членов интерфейса **ISeries**.

Интерфейс **ISeries** можно реализовать для представления разных числовых радов. Например, классами, реализующими этот интерфейс, можно представить такие регулярные числовые последовательности:

|  |
| --- |
| 2,4,6,8,10, … - ряд положительных чётных чисел ai = ai-1 +2, где i>1, a1=2;  1,1,2,3,5,8,13, … - ряд Фибоначчи ai=ai-2 + ai-1 , где i>2, a1=1, a2=1;  1,3,4,7,11,18,29, … - ряд Лукаса ai=ai-2 + ai-1 , где i>2, a1=1, a2=3;  1,2,5,12,29,70 … - ряд Пелла ai=ai-2 + 2\*ai-1 , где i>2, a1=1, a2=2. |

Чтобы показать, что какой-то класс реализует интерфейс, следует при его определении ввести оператор

|  |
| --- |
| **Implements** *имя\_интерфейса* |

На рис. 3.4-2 на основе интерфейса **ISeries** определяется класс **TPELL**, представляющий ряд Пелла:

|  |
| --- |
| *'Объявление класса, реализующего интерфейс для ряда Пелла*  **Public Class TPELL**  **Implements ISeries** *'класс реализует интерфейс ISeries*  *'Добавлены поля для предыдущего и текущего членов ряда*  **Private pred, tek As Integer '**  *'Добавлен конструктор*  **Public Sub New()**  **start()**  **End Sub**  *'Реализация метода, объявленного в интерфейсе ISeries*  *'Метод задает начальное состояние*  **Public Sub start() Implements ISeries.start**  **pred = 1**  **tek = 0**  **End Sub**  *'Реализация свойства, объявленного в интерфейсе ISeries*  *'возвращает следующее значение после tek*  **Public ReadOnly Property GetNext() As Integer \_**  **Implements ISeries.GetNext**  **Get**  **Dim temp As Integer**  **temp = pred + 2 \* tek**  **pred = tek**  **tek = temp**  **Return tek**  **End Get**  **End Property**  *'метод вывода n членов ряда, начиная со следующего*  *'этот метод не объявлялся в интерфейсе, он добавлен*  **Public Function seriesPrint(ByVal n As Integer) As String**  **Dim s As String = ""**  **Dim i As Integer**  **For i = 1 To n**  **s = s & GetNext & Space(3)**  **Next**  **Return s**  **End Function**  **End Class** |

*Рис. 3.4-2. Программный код класса* **TPELL**,

*реализующий интерфейс* **ISeries** *при генерации чисел ряда Пелла*

Если класс реализует интерфейс, то он должен реализовать его полностью и не может, например, выбрать и реализовать только некоторые части интерфейса. Как видно из рисунка 3.4-2, класс **TPELL** реализует оба члена интерфейса: и метод **start(),** и свойство **GetNext**, и они обязаны быть объявлены открытыми (**Public**). Кроме реализации членов интерфейса **ISeries**, в классе **TPELL** объявляются собственные члены: два закрытых поля **pred** и **tek** для хранения значений предыдущего и текущего членов ряда, конструктор и метод **seriesPrint().** Он выводит значения нескольких членов ряда, следующих за текущим членом. Количество членов определяет формальный параметр **n** этого метода. После выполнения метода состояние ряда изменится – текущим членом станет последний выведенный член ряда.

Как мы уже говорили, интерфейс может быть реализован любым количеством классов. На рис. 3.4-3 определяется класс **TSIMPLE**, генерирующий последовательность простых чисел. Его реализация интерфейса совершенно отличается от реализации того же интерфейса классом **TPELL**. Подчеркнем ещё раз, что интерфейс никак не определяет реализацию методов, и каждый класс реализует их так, как ему нужно.

|  |
| --- |
| *'ряд простых чисел*  **Public Class TSIMPLE**  **Implements ISeries**  **Private tek As Integer** *'текущее простое число*  **Public Sub New()** *'конструктор*  **start()**  **End Sub**  *' Реализация свойства, объявленного в интерфейсе ISeries*  *'возвращает следующее после tek простое число*  **Public ReadOnly Property GetNext() As Integer \_**  **Implements ISeries.GetNext**  **Get**  *'флаг, найдено ли простое число*  **Dim flag As Integer = 0**  *'Цикл, пока не найдено простое число*  **Do**  **tek = tek + 1**  **If prost(tek) Then**  *'флаг=1 - простое число найдено*  **flag = 1**  **Return tek**  **End If**  **Loop While flag = 0**  **End Get**  **End Property**  *'Реализация метода, объявленного в интерфейсе ISeries*  *'Метод задает начальное состояние*  **Public Sub start() Implements ISeries.start**  **tek = 1**  **End Sub**  *'функция определяет, является ли целое число x простым*  *'этот метод не объявлялся в интерфейсе и добавлен*  **Private Function prost(ByVal x As Integer) As Boolean**  **Dim i As Integer**  *'перебор всех возможных делителей числа x*  **For i = 2 To CInt(Math.Sqrt(x))**  *'если найден делитель, то число не простое*  **If x Mod i = 0 Then**  **Return False**  **End If**  **Next**  **Return True**  **End Function**  *'метод вывода n членов ряда, начиная со следующего*  *'этот метод не объявлялся в интерфейсе*  **Public Function seriesPrint(ByVal n As Integer) As String**  **Dim s As String = ""**  **Dim i As Integer**  **For i = 1 To n**  **s = s & GetNext & Space(3)**  **Next**  **Return s**  **End Function**  **End Class** |

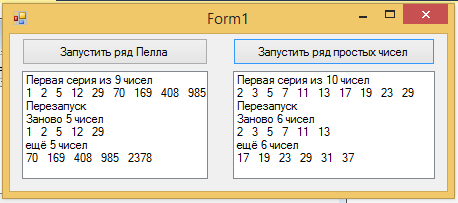
*Рис. 3.4-3. Программный код класса* **TSIMPLE,**

*реализующий интерфейс* **ISeries** *при генерации простых чисел*

Для иллюстрации работы двух описанных классов, использующих один интерфейс, создадим форму с двумя кнопками и двумя **ListBox**. Программный код событийных процедур (текст которых практически аналогичен) и результаты их работы приведены на рис.3.4-4 и рис. 3.4-5.

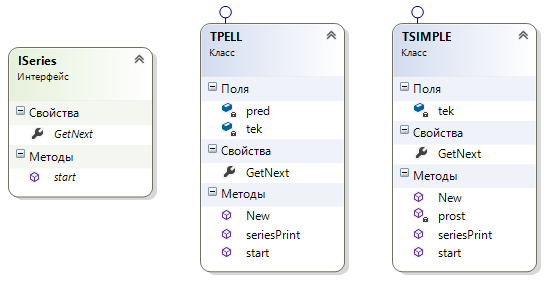
|  |
| --- |
| **Public Class Form1**  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button1.Click**  **Dim z As String = ""** *'строка для вывода*  **Dim p As New TPELL***'создание объекта класса для ряда Пелла*  **ListBox1.Items.Add("Первая серия из 9 чисел")**  **z = p.seriesPrint(9)**  **ListBox1.Items.Add(z)**  **ListBox1.Items.Add("Перезапуск")**  **p.start()**  **ListBox1.Items.Add("Заново 5 чисел")**  **z = p.seriesPrint(5)**  **ListBox1.Items.Add(z)**  **ListBox1.Items.Add("ещё 5 чисел")**  **z = p.seriesPrint(5)**  **ListBox1.Items.Add(z)**  **End Sub**  **Private Sub Button2\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button2.Click**  **Dim z As String = ""***'строка для вывода*  **Dim s As New TSIMPLE***'создание объекта для простых чисел*  **ListBox2.Items.Add("Первая серия из 10 чисел")**  **z = s.seriesPrint(10) : ListBox2.Items.Add(z)**  **ListBox2.Items.Add("Перезапуск")**  **s.start()**  **ListBox2.Items.Add("Заново 6 чисел")**  **z = s.seriesPrint(6) : ListBox2.Items.Add(z)**  **ListBox2.Items.Add("ещё 6 чисел")**  **z = s.seriesPrint(6) : ListBox2.Items.Add(z)**  **End Sub**  **End Class** |

*Рис. 3.4-4. Программный код событийных процедур для вывода чисел ряда Пелла и ряда простых чисел*



*Рис. 3.4-5. Результаты реализации интерфейса* **ISeries**

*классами* **TPELL** *и* **TSIMPLE**



*Рис. 3.4-6. Схема классов* **TPELL** *и* **TSIMPLE** *с интерфейсом* **ISeries**

Схема классов с интерфейсом представлена на рис. 3.4-6.

**Пример 3.4-2. Применить определённый в примере 3.4-1 интерфейс как тип и показать использование ссылочных переменных интерфейса.**

Несмотря на то, что нельзя создавать экземпляры (объекты) интерфейсов, но интерфейс, как и класс, является ссылочным типом. Подобно тому, как можно объявлять ссылку с типом абстрактного класса, разрешено объявлять ссылку с типом интерфейса. Такая ссылка может быть связана с объектом любого класса, который реализовал данный интерфейс. При вызове метода для объекта с помощью интерфейсной ссылки выполняется версия метода, которая реализована этим объектом. С помощью такой ссылки можно получить доступ ко всем членам класса, реализующим соответствующие члены интерфейса. Однако ссылка с типом интерфейса не позволяет получить доступ к членам класса, которые отсутствовали в интерфейсе, но были добавлены в класс.

На рис. 3.4-7 приведен программный код событийной процедуры, в которой одна и та же ссылочная переменная **g** типа интерфейса используется при вызове свойства **GetNext** как для объекта **p** класса **TPELL**, так и для объекта **s** класса **TSIMPLE**.

|  |
| --- |
| **Public Class Form1**  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button1.Click**  **Dim z As String = ""** *'строка для вывода*  **Dim p As New TPELL**  **Dim s As New TSIMPLE**  **Dim g As ISeries**  **For i As Integer = 1 To 6**  **g = p : z = g.GetNext**  **ListBox1.Items.Add("След. число ряда Пелла = " & z)**  **g = s : z = g.GetNext**  **ListBox1.Items.Add("След. простое число = " & z)**  **Next**  **End Sub**  **End Class** |

*Рис. 3.4-7. Использование интерфейсной ссылки*

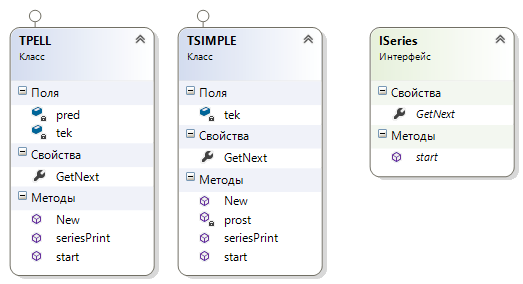
*для доступа к членам классов, реализующих интерфейс*

Ссылка с типом интерфейса может быть также использована как параметр метода, может определять тип возвращаемого методом значения и т.п. Чтобы продемонстрировать применимость интерфейсной ссылки в качестве параметра, определим в модуле формы процедуру – функцию **report()**, из тела которой выполняется обращение к реализации свойства **GetNext**. На рис. 3.4-8 приведен программный код этой функции, а также событийной процедуры нажатия на вторую кнопку, в которой иллюстрируется применение интерфейсной ссылки как параметра процедуры.

|  |
| --- |
| **Public Class Form1**  **Public Function report(ByVal g As ISeries) As String**  **Dim z As String**  **z = "След. число объекта " + g.GetType.ToString + "= "**  **z = z & g.GetNext**  **Return z**  **End Function**  **Private Sub Button2\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button2.Click**  **Dim z As String = ""** *'строка для вывода*  **Dim p As New TPELL**  **Dim s As New TSIMPLE**  **Dim g As ISeries**  **For i As Integer = 1 To 3**  **z = report(p)**  **ListBox2.Items.Add(z)**  **z = report(s)**  **ListBox2.Items.Add(z)**  **g = p :z = report(g)**  **ListBox2.Items.Add(z)**  **g = s : z = report(g)**  **ListBox2.Items.Add(z)**  **Next**  **End Sub**  **End Class** |

*Рис. 3.4-8. Использование интерфейсной ссылки   
как параметра процедуры*

**ЗАЧЕМ НИЖЕСЛЕДУЮЩАЯ СХЕМА?** Ведь код классов и интерфейса не менялся (поменяли только событийную процедуру) и оба рисунка 3.4-6 и 3.4-9 абсолютно одинаковы. Я думаю, рис 3.4-9 убрать и переименовать рисунок с результатами из 3.4-10 в 3.4-9

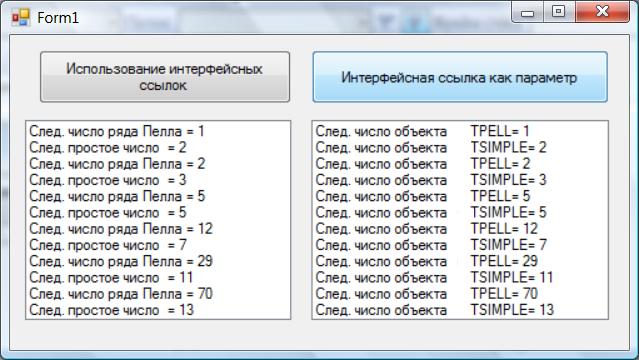


*Рис.3.4-9. Схема классов* *ПРИМЕРА 3.4-2* *с использованием   
интерфейсных ссылок*

Параметром функции **report()** является ссылка **g** с типом интерфейса **ISeries**. Выражение **g.GetType** в теле функции **report()** позволяет получить имя класса, ссылка на который используется в качестве фактического параметра при обращении к функции **report()**. Для того, чтобы это имя класса «приклеить» к строке **z** - результату функции, это имя приводится к строковому типу применением метода **ToString().** Оператор **g.GetNext** обеспечивает вызов той реализации свойства **GetNext**, которая соответствует типу аргумента.

Использование интерфейсной ссылки в качестве параметра позволяет применять функцию **report()** для обработки объектов любых классов, которые реализовали интерфейс **ISeries**. Код процедуры **Button2\_Click()** это иллюстрирует. В ней созданы два объекта классов **TPELL** и **TSIMPLE** и ассоциированные с ними ссылки **p** и **s**. Их использование в качестве фактических параметров функции **report()** приводит к вызовам свойств **GetNext** из соответствующих классов.

Результаты работы событийных процедур приведены на рис. 3.4-9.



*Рис.3.4-9. Результаты использования интерфейсных ссылок*

Применение в качестве параметров интерфейсных ссылок, так же, как и ссылок с типом базового класса (при наличии в нем виртуальных методов), обеспечивает позднее (динамическое) связывание – одно из главных проявлений полиморфизма.

Итак, когда при программировании нужно описать некую функциональность без реализации, возникает проблема выбора между созданием интерфейса и использованием абстрактного класса. Следует руководствоваться общим правилом. Если вы можете полностью описать, то, что должен делать класс, не уточняя, как он должен это делать, следует использовать интерфейс. Если же в описание класса необходимо вставить какие-то детали реализации, то используется абстрактный класс. И ещё. Если некий набор действий имеет смысл только для какой-то конкретной иерархии классов, реализующих эти действия разными способами, уместнее задать этот набор в виде виртуальных методов базового абстрактного класса иерархии. То, что работает в пределах иерархии одинаково, предпочтительно полностью определить в базовом классе. Интерфейсы же чаще используются для задания общих свойств и методов объектов различных иерархий.

В заключение сформулируем отличия интерфейса от базового класса:

* члены интерфейса по умолчанию имеют модификатор доступа **Public** и не могут иметь модификаторов, заданных явным образом;
* интерфейс не может содержать полей и обычных методов – все элементы интерфейса должны быть абстрактными;
* класс, который реализует интерфейс, должен определять все его элементы, в то время как потомок абстрактного класса может не переопределять часть абстрактных методов предка (в этом случае производный класс также будет абстрактным).

### 3.5. Задачи для самостоятельного решения

***Описать базовый класс (возможно, абстрактный или интерфейсный), в котором с помощью виртуальных или абстрактных методов и свойств задается основа для производных классов. Классы, как правило (и где возможно), должны содержать: закрытые поля, свойства для доступа к полям, конструкторы с параметрами и без параметров, методы. Для абстрактного класса определить, какие методы должны быть абстрактными, а какие обычными. Полную структуру классов и их взаимосвязь продумать самостоятельно.***

1. Создать абстрактный класс (или интерфейс) **ФИГУРА** с методами вычисления площади и периметра, а также методом, выводящим информацию о фигуре на экран.

Создать производные (или реализующие интерфейс) классы: **ПРЯМОУГОЛЬНИК**, **КРУГ**, **ТРЕУГОЛЬНИК** со своими методами вычисления площади и периметра.

Создать массив **n** фигур и вывести полную информацию о фигурах на экран.

1. Создать абстрактный класс (или интерфейс) **ИЗДАНИЕ** с методом, позволяющим **вывести** на экран информацию об издании, а также определить является ли данное издание искомым.

Создать производные (или реализующие интерфейс) классы: **КНИГА** (название, фамилия автора, год издания, издательство), **СТАТЬЯ** (название, фамилия автора, название журнала, его номер и год издания), **ЭЛЕКТРОННЫЙ\_РЕСУРС** (название, фамилия автора, ссылка, аннотация) со своими методами вывода информации на экран.

Создать каталог (массив) из **n** изданий, вывести полную информацию из каталога, а также организовать поиск изданий по фамилии автора.

1. Создать абстрактный класс (или интерфейс) **ТРАНСПОРТ** с методами, позволяющими вывести на экран информацию о транспортном средстве, а также определить грузоподъемность транспортного средства.

Создать производные (или реализующие интерфейс) классы: **АВТОМОБИЛЬ** (марка, номер, скорость, грузоподъемность), **МОТОЦИКЛ** (марка, номер, скорость, грузоподъемность, наличие коляски, при этом если коляска отсутствует, то грузоподъемность равна 0), **ГРУЗОВИК** (марка, номер, скорость, грузоподъемность, наличие прицепа, при этом если есть прицеп, то грузоподъемность увеличивается в два раза) со своими методами вывода информации на экран, и определения грузоподъемности.

Создать базу (массив) из **n** машин, вывести полную информацию из базы на экран, а также организовать поиск машин, удовлетворяющих требованиям грузоподъемности.

1. Создать абстрактный класс (или интерфейс) **ПЕРСОНА** с методами, позволяющими вывести на экран информацию о персоне, а также определить ее возраст (на момент текущей даты).

Создать производные (или реализующие интерфейс) классы: **АБИТУТИЕНТ** (фамилия, дата рождения, факультет), **СТУДЕНТ** (фамилия, дата рождения, факультет, курс), **ПРЕПОДАВАТЕЛЬ** (фамилия, дата рождения, факультет, должность, стаж), со своими методами вывода информации на экран, и определения возраста.

Создать базу (массив) из **n** персон, вывести полную информацию из базы на экран, а также организовать поиск персон, чей возраст попадает в заданный диапазон.

1. Создать абстрактный класс (или интерфейс) **ТОВАР** с методами, позволяющими вывести на экран информацию о товаре, а также определить, может ли купить товар покупатель, имеющий заданную сумму денег.

Создать производные (или реализующие интерфейс) классы:   
**ПРОДУКТ** (название, цена, дата производства, срок годности), **ПАРТИЯ** (название, цена за штуку, количество штук, дата производства, срок годности), **ТЕЛЕФОН** (название, цена) со своими методами вывода информации на экран, и определения соответствия заданной цене.

Создать базу (массив) из **n** товаров, вывести полную информацию из базы на экран, а также организовать поиск товара, который может купить покупатель, имеющий заданную сумму денег.

1. Создать абстрактный класс (или интерфейс) **ТОВАР** с методами, позволяющими вывести на экран информацию о товаре, а также определить, предназначен ли он для заданного возраста потребителя.

Создать производные (или реализующие интерфейс) классы: **ИГРУШКА** (название, цена, производитель, материал, возраст, на который рассчитана), **КНИГА** (название, автор, цена, издательство, возраст, на который рассчитана), **СПОРТИНВЕНТАРЬ** (название, цена, производитель, возраст, на который рассчитан), со своими методами вывода информации на экран, и определения соответствия возрасту потребителя.

Создать базу (массив) из **n** товаров, вывести полную информацию из базы на экран, а также организовать поиск товаров для потребителя в заданном возрастном диапазоне.

1. Создать абстрактный класс (или интерфейс) **ТЕЛЕФОННЫЙ\_СПРАВОЧНИК** с методами, позволяющими вывести на экран информацию о записях в телефонном справочнике, а также определить соответствие записи критерию поиска.

Создать производные (или реализующие интерфейс) классы: **ПЕРСОНА** (фамилия, адрес, номер телефона), **ОРГАНИЗАЦИЯ** (название, адрес, телефон, факс, контактное лицо), **ДРУГ** (фамилия, адрес, номер телефона, дата рождения) со своими методами вывода информации на экран, и определения соответствия заданной фамилии.

Создать базу (массив) из **n** записей, вывести полную информацию из базы на экран, а также организовать поиск в базе по фамилии.

1. Создать абстрактный класс (или интерфейс) **КЛИЕНТ** с методами, позволяющими вывести на экран информацию о клиентах банка, а также определить соответствие клиента критерию поиска.

Создать производные (или реализующие интерфейс) классы: **ВКЛАДЧИК** (фамилия, дата открытия вклада, размер вклада, процент по вкладу), **КРЕДИТОР** (фамилия, дата выдачи кредита, размер кредита, процент по кредиту, остаток долга), **ОРГАНИЗАЦИЯ** (название, дата открытия счета, номер счета, сумма на счету) со своими методами вывода информации на экран, и определения соответствия дате (открытия вклада, выдаче кредита, открытия счета).

Создать базу (массив) из **n** клиентов, вывести полную информацию из базы на экран, а также организовать поиск клиентов, начавших сотрудничать с банком в заданную дату.

1. Создать абстрактный класс (или интерфейс) **ПРОГРАММНОЕ\_ОБЕСПЕЧЕНИЕ** с методами, позволяющими вывести на экран информацию о программном обеспечении, а также определить соответствие возможности использования (на момент текущей даты).

Создать производные (или реализующие интерфейс) классы: **СВОБОДНОЕ** (название, производитель), **УСЛОВНО\_БЕСПЛАТНОЕ** (название, производитель, дата установки, срок бесплатного использования), **КОММЕРЧЕСКОЕ** (название, производитель, цена, дата установки, срок использования) со своими методами вывода информации на экран, и определения возможности использования на текущую дату.

Создать базу (массив) из **n** видов программного обеспечения, вывести полную информацию из базы на экран, а также организовать поиск программного обеспечения, которое допустимо использовать на текущую дату.

1. Создать абстрактный класс (или интерфейс) **ТРАНСПОРТ** с методами, позволяющими вывести на экран информацию о транспортном средстве, а также определить, находится ли транспортное средство в пределах заданных координат.

Создать производные (или реализующие интерфейс) классы: **САМОЛЁТ** (марка, максимальные скорость и высота, количество пассажиров, координаты), **АВТОМОБИЛЬ** (марка, номер, год выпуска, координаты), **КОРАБЛЬ** (название, координаты, скорость, количество пассажиров, порт приписки) своими методами вывода информации на экран, и определения присутствия транспортного средства в пределах заданных координат.

Создать массив из **n** транспортных средств, вывести полную информацию из базы на экран, а также организовать поиск транспортных средств, которые сейчас находятся в пределах заданных координат.

1. Создать абстрактный класс (или интерфейс) **ИГРУШКА** с методами, позволяющими вывести на экран информацию о товаре, а также определить соответствие игрушки критерию поиска.

Создать производные (или реализующие интерфейс) классы: **КУБИК** (цвет, цена, материал, размер ребра), **МЯЧ** (цена, цвет, диаметр, материал), **МАШИНКА** (название, цена, производитель, цвет), со своими методами вывода информации на экран, и определения соответствия заданному цвету.

Создать базу (массив) из **n** игрушек, вывести полную информацию из базы на экран, а также организовать поиск игрушек заданного цвета.

1. Создать абстрактный класс (или интерфейс) **ТЕЛО** с методами вычисления площади поверхности и объёма, а также методом, выводящим информацию о фигуре на экран.

Создать производные (или реализующие интерфейс) классы: **ПАРАЛЛЕЛЕПИПЕД, ШАР, ПИРАМИДА**  со своими методами вычисления площади и объёма.

Создать массив **n** фигур и вывести полную информацию о фигурах на экран.

1. Создать абстрактный класс (или интерфейс) **ТРЕУГОЛЬНИК**, заданный длинами двух сторон и угла между ними, с методами вычисления площади и периметра треугольника, а также методом, выводящим информацию о фигуре на экран.

Создать производные (или реализующие интерфейс) классы: **ПРЯМОУГОЛЬНЫЙ, РАВНОБЕДРЕННЫЙ, РАВНОСТОРОННИЙ** со своими методами вычисления площади и периметра.

Создать массив **n** треугольников и вывести полную информацию о треугольниках на экран.

1. Создать абстрактный класс (или интерфейс) **УРАВНЕНИЕ**, с методами вычисления корня уравнения и вывода результата на экран.

Создать производные (или реализующие интерфейс) классы: **ЛИНЕЙНОЕ, КВАДРАТНОЕ** со своими методами вычисления корней и вывода на экран.

Создать массив **n** уравнений и вывести полную информацию об уравнениях на экран.

1. Создать абстрактный класс (или интерфейс) **ВАЛЮТА**, с методами перевода денежной суммы в рубли и вывода на экран.

Создать производные (или реализующие интерфейс) классы: **ДОЛЛАР, ЕВРО** со своими методами перевода и вывода на экран.

Создать массив **n** валютных денежных сумм и вывести полную информацию о них на экран.

1. Создать абстрактный класс (или интерфейс) **ПРОГРЕССИЯ** с методами вычисления j-го элемента прогрессии, её суммы и методом, выводящим сумму на экран.

Создать производные (или реализующие интерфейс) классы: **АРИФМЕТИЧЕСКАЯ, ГЕОМЕТРИЧЕСКАЯ** со своими методами вычисления.

Создать массив **n** прогрессий и вывести сумму каждой из них экран.

1. Создать абстрактный класс (или интерфейс) **ТРОЙКА**, задаваемый тремя полями типа **UInteger** и с виртуальным методом увеличения на единицу.

Создать производные (или реализующие интерфейс) классы: **ДАТА** (год, месяц, день) и **ВРЕМЯ** (час, минута, секунда) со своими методами вычисления даты следующего дня и увеличения времени на секунду.

Создать массив из элементов базового класса, заполненный ссылками на производные классы и продемонстрировать использование всех разработанных элементов класса.

1. Создать абстрактный класс (или интерфейс) **ЦЕЛОЕ\_ЧИСЛО** с виртуальными методами - арифметическими операциями и методом вывода информации на экран. Число представляется массивом, каждый элемент которого – цифра.

Создать производные (или реализующие интерфейс) классы: **ДЕСЯТИЧНОЕ** и **ДВОИЧНОЕ** со своими методами - арифметическими операциями и вывода.

Создать массив из элементов базового класса, заполненный ссылками на производные классы и продемонстрировать использование всех разработанных элементов класса.

1. Создать базовый класс **ТОЧКА**, задаваемый координатами точки (целые числа в диапазоне [0;255]). На его основе создать классы **ЦВЕТНАЯ\_ТОЧКА** (имеет поле для цвета - целое число в диапазоне [0;255]) и **ЛИНИЯ**. На основе класса создать классы **ЦВЕТНАЯ\_ЛИНИЯ** и **МНОГОУГОЛЬНИК**. В классах описать следующие элементы:

* конструкторы с параметрами и конструкторы по умолчанию;
* свойства для установки и изменения значений всех координат, а также для изменения цвета и получения текущего цвета;
* для линий – методы изменения угла поворота линий относительно первой точки;
* для многоугольника – метод масштабирования.

Создать массив из элементов базового класса, заполненный ссылками на производные классы и продемонстрировать использование всех разработанных элементов класса.

1. Создать базовый класс **СТРОКА**, содержащий:

* поля для хранения символов строки и длины строки;
* конструктор без параметров;
* конструктор с параметром типа **String**;
* метод очистки строки (сделать строку пустой).

На его основе создать класс **ДЕСЯТИЧНАЯ\_СТРОКА**. Строки класса могут содержать только символы десятичных цифр и символы - и +, задающие знак числа, которые могут находиться только в первой позиции числа, причем символ + может отсутствовать, тогда число считается положительным. Если в составе инициализирующей строки встретятся любые другие символы, класс **ДЕСЯТИЧНАЯ\_СТРОКА** принимает нулевое значение. Содержимое данных строк рассматривается как целое число. В классе описать элементы:

* конструктор с параметром типа Integer;
* метод, реализующий арифметическую разность строк;
* метод проверки на «больше» (по значению);
* метод проверки на «меньше» (по значению).

Продемонстрировать использование всех разработанных элементов

класса.

### 3.6. Лабораторная работа по теме

### *«Отношение между классами.*

### *Включение, вложение и наследование классов»*

**Цель** данной лабораторной работы состоит в изучении основных понятий объектно-ориентированной технологии и получении практических навыков разработки объектной модели, в том числе определение: множества образующих классов (декомпозиция), отношений между классами, в том числе наследования, включения, и изучение абстрактных классов.

#### 3.6.1. Задание

1. ***Изучите основные отношения*** *между классами в VB (3.1 –3.4).*
2. ***Выберите вариант задания*** *из пункта 3.6.2.*
3. ***Разработайте концептуальную модель задачи****, то есть:*

* проведите объектно-ориентированную декомпозицию,определив множество образующих ее классов*;*
* определите для выделенных классов существенные в данной задаче интерфейсные члены класса*:*
* *атрибуты;*
* *операции (возможные действия), которые объекты могут выполнить для решения задачи.*

1. ***Разработайте объектную модель задачи логического уровня, определив необходимые связи между классами объектов, в том числе иерархию классов,*** *для чего:*

* определите базовый(е) класс(ы) и их структуру;
* определите классы – наследники;
* создайте схему (иерархию) наследования классов.

1. ***Проведите дальнейшую формализацию задания,*** *для чего:*

* определите перечень исходных данных;
* представьте смысловые данные, которые будут использоваться в ходе решения задачи в виде переменных или массивов, присвоив им соответствующие имена;
* определите размерности и типы используемых данных;
* приведите геометрическую иллюстрацию преобразования исходных данных в результат;
* при необходимости приведите расчетные формулы, которые будут использоваться для преобразования исходных данных в результаты.

1. ***Разработайте графический интерфейс пользователя – две формы:***

* разработайте форму, предназначенную для ввода исходных данных, как с клавиатуры, так и из текстового файла, и для отображения входных данных на форме;
* разработайте выходную форму, предназначенную для вывода результатов решения задачи на эту форму.

1. ***Создайте программный код для проекта, реализующий поставленную задачу и получите объектную модель на уровне реализации.***
2. ***Выполните проект и получите решение.***
3. ***Обоснуйте правильность полученных результатов.***

**3.6.2. Варианты индивидуальных заданий**

Для решения задачи данной лабораторной работы можно воспользоваться заданием из табл. 1.5-1 (Лабораторная работа **Темы 1**) или получить его у преподавателя.

#### 3.6.3. Содержание отчета

1. Тема и название работы.
2. Общее задание на разработку моделей и программного проекта, и вариант индивидуального задания.
3. Концептуальная модель предметной области задачи.
4. Объектная модель программного проекта на логическом уровне и ее геометрическая иллюстрация
5. Графический интерфейс пользователя:

* форма, предназначенная для ввода исходных данных;
* форма для вывода результатов решения задачи.

1. Программный проект на уровне реализации (приложения) с подробными комментариями.
   1. Содержание классов проектов и их элементы;
   2. Схемы алгоритмов функциональных задач;
   3. Программный код проектов с подробными комментариями;
2. Схема объектной модели на уровне реализации, построенная по программному коду.
3. Результаты выполнения приложения.
4. Обоснование правильности полученных результатов.

#### 3.6.4. Пример выполнения задания

1. ***Тема и название работы:***

Отношение между классами. Включение, вложение и наследование классов.

Формирование списка фамилий студентов, которым по итогам сессии необходимо повысить успеваемость до фактического среднего уровня.

1. ***Задание на разработку моделей и программного проекта, вариант индивидуального задания:***

В группе из **n** студентов каждым студентом получено в сессию по **m** оценок.

Разработать диаграмму классов, которая может быть использована в качестве объектной модели для решения задачи формирования списка фамилий студентов, которым по итогам сессии необходимо повысить успеваемость до фактического среднего уровня.

Формируемый список фамилий студентов должен быть представлен в двух видах: в порядке убывания среднего балла, полученного за сессию, а также по алфавиту. Исходные списки студентов и результатов сессии могут находиться как в текстовом файле, так и введены с клавиатуры.

1. ***Объектная модель программного проекта на концептуальном уровне:***
2. **Проведите объектно-ориентированную декомпозицию и определите множество образующих классов.**

Анализируя предметную область на концептуальном уровне, выделим три класса. Класс **СТУДЕНТ** содержит сведения об одном студенте (моделирует одного студента), класс **ГРУППА** будет моделировать группу из **n** студентов (т.е. содержать массив из **n** объектов класса **СТУДЕНТ** и общий средний балл всей группы). В классе   
**РЕЗУЛЬТАТ** будет содержаться полученный список (фамилии и средние оценки) тех студентов, у которых средняя оценка ниже общего среднего балла группы, т.е. которым надо повысить успеваемость. Объектная модель программного проекта на концептуальном уровне представлена на рис. 3.6-1.



*Рис. 3.6-1.* *Объектная модель программного проекта   
на концептуальном уровне*

1. **Определите для выделенных классов существенные в данной задаче интерфейсные члены класса:**

* **Входные атрибуты**:
  + **fam** – фамилия одного студента (**String**);
  + **m** – количество экзаменов у студента в сессию (**Integer**);
  + **oc(m-1)** – одномерный массив оценок одного студента (**Integer**);
  + **n** – количество студентов в одной группе (**Integer**);
  + **stud(n-1)** – одномерный массив, хранящий сведения обо всех **n** студентах группы (массив объектов класса **СТУДЕНТ**).
* **Рассчитываемые атрибуты**:
* **so** – средняя оценка каждого из **n** студентов одной группы (**Double**);
* **sr** - общий средний балл всей группы (**Double**);
* **Выходные атрибуты:**
* **k** – число студентов, для которых средняя оценка **so** меньше общего среднего балла всей группы **sr** (**Integer**);
* **f(k-1)** - одномерный массив фамилий искомых студентов   
  (**String**);
* **sb(k-1)** - одномерный массив их средних баллов (**Double**);
* **Операции:**
* ***ввод исходных данных*** (фамилий студентов и их оценок);
* ***нахождение средней оценки*** каждого студента и ***общего среднего балла*** группы;
* ***формирование списка студентов***, имеющих средний балл ниже общего среднего балла группы;
* ***сортировка сформированного списка*** студентов по убыванию среднего балла и по алфавиту;
* ***вывод результатов*** на форму.

1. **Объектная модель программного проекта на концептуальном уровне с учетом интерфейсных членов класса представлена на рис. 3.6-2.**



Рис. 3.6-2. *Объектная модель программного проекта   
на концептуальном уровне с учетом интерфейсных членов класса*

1. ***Объектная модель программного проекта на логическом уровне:***
   1. **Уточните состав классов объектной модели, ее логическую структуру и связи между классами.**

При анализе предметной области на логическом уровне, заметим, что классы **СТУДЕНТ** и **ГРУППА** находятся в отношении композиции, а класс **РЕЗУЛЬТАТ** является наследником класса **ГРУППА**. По условию задачи исходные данные могут как вводиться с клавиатуры, так и считываться из файла, поэтому классы **СТУДЕНТ** и **ГРУППА** будут содержать по две перегружаемые процедуры ввода данных (ввод с клавиатуры и из файла). Кроме того, для поставленной задачи результирующие списки выводятся и по убыванию среднего балла, и по алфавиту. Поэтому в классе **РЕЗУЛЬТАТ** предусмотрим абстрактный (виртуальный) метод сортировки, и, следовательно, сам класс объявим абстрактным базовым классом. Именно с помощью этого виртуального метода сортировки будет реализован один из основных принципов ООП **–** *полиморфизм*. Для реализации полиморфизма и вывода результирующего списка студентов по убыванию среднего балла объявим класс **СписПоУбыв**, который будет наследником класса **РЕЗУЛЬТАТ** и в котором будет переопределён (реализован) метод сортировки по убыванию. Аналогичным образом для формирования списка студентов по алфавиту объявим производный от абстрактного класса **РЕЗУЛЬТАТ** класс **СписПоАлф**, в котором будет переопределён метод сортировки по алфавиту.

****

*Рис. 3.6-3. Состав классов**объектной модели на логическом уровне*



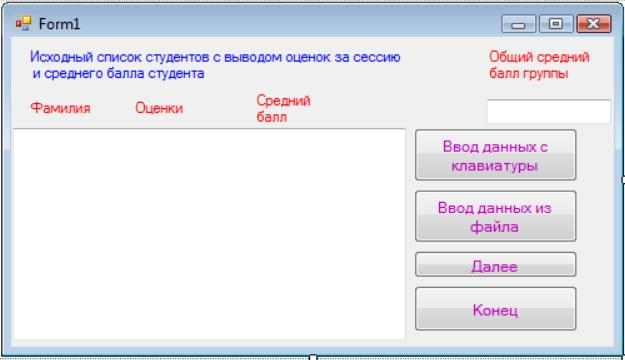
*Рис. 3.6-4.**Объектная модель задачи на логическом уровне*

Таким образом, при анализе предметной области на логическом уровне можно выделить 5 классов (рис. 3.6-3). Причем класс **ГРУППА** с исходными данными о студентах – базовый, и три производных класса с результатами, в том числе один абстрактный класс (рис. 3.6-4).

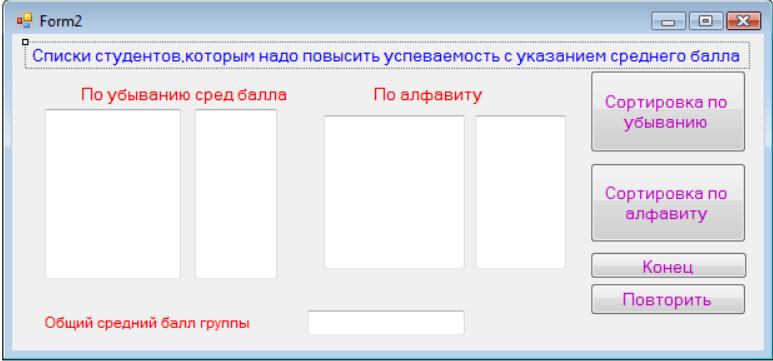
* 1. **Геометрическая иллюстрация была показана на рис. 2.7-4.**

1. ***Графический интерфейс пользователя:***

Формы интерфейса пользователя приведены на рис. 3.6-5 и 3.6-6.

**

*Рис. 3.6-5. Входная форма интерфейса пользователя*



*Рис. 3.6-6. Выходная форма интерфейса пользователя*

1. ***Программный проект на уровне реализации (приложения).*** 
   1. Содержание классов проектов и их элементы;

**Класс СТУДЕНТ:**

**Поля**:

* **Fm** – количество экзаменов в сессию (**Integer**);
* **Ffam** - фамилия студента (**String**);
* **Foc()** - одномерный целочисленный массив оценок;
* **Fso** - средняя оценка студента за сессию (**Double**).

В соответствии с основным принципом ООП – инкапсуляцией – объявим все поля для этого класса закрытыми (**Private**).   
Имена всех закрытых (внутренних) полей будем начинать с буквы **F** (от англ. field, поле).

**Свойства для чтения и записи полей класса:**

* **fam** – для чтения и записи фамилии студента;
* **m** – для чтения и записи количества экзаменов.

Для работы с внутренними полями определим в каждом классе методы – свойства, причем для свойств, которые допускают запись (установку) значения поля (т.е. имеют часть **Set**) предусмотрим в этой части проверку корректности устанавливаемых значений полей.

**Свойства только для записи полей класса:**

* **ocenka** - для записи одного значения в массив оценок студента (поля **Foc**) для контроля каждого записываемого значения оценки.

**Свойства только для чтения полей класса:**

* **oc** - для чтения целиком массива оценок студента **Foc**;
* **so** - для чтения средней оценки студента (значение поля **Fso** определяется методом **Sred()** и не может изменяться снаружи).

**Методы:**

* + - процедура – функция **Sred()** нахождения средней оценки студента;
    - две перегружаемые процедуры - подпрограммы **vvod()** для ввода данных с клавиатуры и из файла;
    - процедура – функция **vivod()** для вывода сведений о студенте («собирает» все сведения о студенте в одну строку).

Так как методы класса напрямую «видят» все поля своего класса и непосредственно работают с полями класса, то поля класса, как правило, не включаются в их список формальных параметров.

**Конструкторы для создания экземпляров (объектов) класса:**

* + - конструктор с параметром (число экзаменов), который инициализирует поле **Fm** через свойство **m** и выделяет необходимое количество памяти под массив оценок студента – поле **Foc** класса;
    - пустой конструктор по умолчанию (без параметров). Правильный стиль требует, чтобы среди конструкторов класса всегда присутствовал конструктор по умолчанию.

**Базовый класс ГРУППА в композиции с классом СТУДЕНТ:**

**Поля**:

* **Fn** – число студентов в группе (**Integer**);
* **Fstud()** - одномерный массив студентов (объектов класса   
  **СТУДЕНТ**);
* **Fsr** - общий средний балл за сессию всех студентов группы (**Double**).

В соответствии с основным принципом ООП – инкапсуляцией   
– и так как класс является базовым, т.е. предназначен, в том числе, для наследования, то объявим все поля базового класса защищёнными (**Protected**), чтобы они были доступны классам-наследникам.

**Свойства для чтения и записи полей класса:**

* **n** – для чтения и записи количества студентов в группе;
* **stud** – для чтения и записи сведений о студентах (массива объектов класса **СТУДЕНТ**).

Для работы с защищёнными полями определим в классе свойства, причем для свойств, которые допускают запись (установку) значения поля (т.е. имеют часть **Set**) предусмотрим в этой части проверку корректности устанавливаемых значений полей.

**Свойства только для чтения полей класса:**

* **sr** - для чтения общего среднего балла группы (значение поля **Fsr** определяется методом **Sred()** и не может изменяться снаружи).

**Методы:**

* процедура – функция **Sred(**) нахождения общего среднего балла в группе;
* две перегружаемые процедуры-подпрограммы **vvodGr()** для ввода данных о группе студентов с клавиатуры и из текстового файла;
* процедура - функция **vivod()** для вывода сведений о группе студентов («собирает» все сведения обо всех студентах группы в одну строку).

**Конструкторы для создания экземпляров (объектов) класса:**

* + - конструктор с параметром (число студентов в группе), который выделяет необходимое количество памяти под все массивы – поля класса и инициализирует поле **Fn** через свойство **n**;
    - пустой конструктор по умолчанию (без параметров). Правильный стиль требует, чтобы среди конструкторов класса всегда присутствовал конструктор по умолчанию.

**Производный абстрактный класс РЕЗУЛЬТАТ, являющийся наследником класса ГРУППА:**

**Поля (защищённые):**

* **Fk** - количество студентов, которым надо повысить успеваемость, т.е. имеющих средний балл ниже общего среднего балла группы (**Integer**);
* **Ff()** - одномерный строковый массив их фамилий;
* **Fsb()** – одномерный вещественный массив их средних оценок.

**Свойства только для чтения полей класса:**

* **k** - для чтения количества студентов, которым надо повысить успеваемость;
* **f** - для чтения их фамилий;
* **sb** - для чтения их средних оценок.

Значения этих полей не могут изменяться вне класса, а вычисляются методом **FormirSpisok().**

***Методы:***

* процедура **–** подпрограмма **FormirSpisok()** формирования списка студентов, имеющих средний балл ниже общего среднего балла группы (формирования полей класса **Fk**, **Ff** и **Fsb**) из полей, объявленных в базовом классе **ГРУППА** и с помощью свойств класса **СТУДЕНТ**;
* перегружаемые процедуры **VivodMas()** вывода полей класса **Ff** и **Fsb** - одномерных массивов разного типа (для **String** и для **Double**);
* абстрактный виртуальный метод – **MustOverride** **Sub sort()**- процедура сортировки списка студентов. Способ сортировки (по убыванию среднего балла или по алфавиту) должен быть переопределен в двух классах-наследниках этого класса   
  **СписПоУбыв** и **СписПоАлф**.

**Конструкторы класса:**

* конструктор с параметром вызывает конструктор базового класса **ГРУППА** и выделяет необходимое количество памяти под массивы – поля класса;
* пустой конструктор по умолчанию (без параметров) «на всякий случай», правильный стиль требует, чтобы среди конструкторов класса всегда присутствовал конструктор по умолчанию.

**Производный класс СписПоУбыв, являющийся наследником**

**Класса РЕЗУЛЬТАТ:**

**Методы**:

* переопределенный (**Overrides**) метод базового класса **sort()**- процедура сортировки по убыванию массива средних баллов студентов, имеющих средний балл ниже общего среднего балла группы с одновременной перестановкой их фамилий;

**Конструкторы класса:**

* конструктор с параметром необходим для вызова конструктора базового класса;
* пустой конструктор по умолчанию (без параметров) «на всякий случай», правильный стиль требует, чтобы среди конструкторов класса всегда присутствовал конструктор по умолчанию.

**Производный класс СписПоАлф, являющийся наследником**

**класса РЕЗУЛЬТАТ:**

**Методы**:

* переопределенный (**Overrides**) метод базового класса **sort()–** процедура сортировки по алфавиту массива фамилий студентов, имеющих средний балл ниже общего среднего балла группы с одновременной перестановкой их средних баллов;

**Конструкторы класса:**

* конструктор с параметром необходим для вызова конструктора базового класса;
* пустой конструктор по умолчанию (без параметров) «на всякий случай», правильный стиль требует, чтобы среди конструкторов класса всегда присутствовал конструктор по умолчанию.

**6.2) Схемы алгоритмов функциональных задач:**

Схемы алгоритмов функциональных задач реализуются по указа-

нию преподавателя.

**6.3) Программный код проекта приведен на рис 3.6-7.**

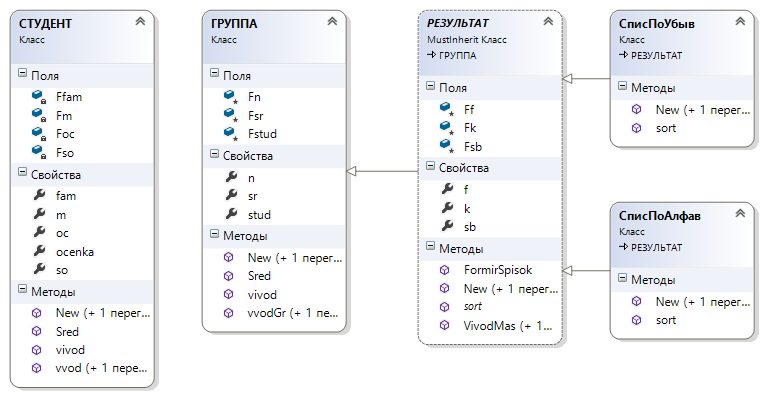
Для реализации полиморфизма на 1 форме создадим два объекта **SpUb** и **SpAlf** для двух классов – наследников базового класса **РЕЗУЛЬТАТ** с абсолютно одинаковыми значениями полей. На второй форме создаем ссылку с типом базового класса и по очереди ассоциируем ее со ссылками объектов – наследников (т.е. присваиваем ссылке с типом базового класса значение ссылки с типом производного класса). Так как метод сортировки списка «нерадивых» студентов (т.е. результирующих массивов) объявлен в базовом классе как абстрактный виртуальный метод, то способ сортировки определяется именно фактическим значением ссылки с типом базового класса, т.е. реализуется «позднее связывание».

|  |  |
| --- | --- |
| |  | | --- | | **Option Strict On**  *'Класс СТУДЕНТ*  **Public Class СТУДЕНТ**  *'====ПОЛЯ (все закрыты)====*  **Private Ffam As String** *'фамилия*  **Private Fm As Integer** *'кол-во экзаменов в сессию*  **Private Foc() As Integer** *'массив оценок за сессию*  **Private Fso As Double** *'средний балл за сессию*  *'====КОНСТРУКТОРЫ=============*  *' Конструктор класса получая кол-во экзаменов,*  *'заполняет поле Fm через св-во и выделяет память под Foc()*  **Public Sub New(ByVal kol As Integer)**  **m = kol** *'заполнение поля кол-ва экзаменов через св-во*  **ReDim Foc(kol - 1)**  **End Sub**  *'Пустой конструктор по умолчанию*  **Public Sub New()**  **End Sub**  *'====СВОЙСТВА========*  *'Свойство для фамилии*  **Public Property fam() As String**  **Get**  **Return Ffam**  **End Get**  **Set(ByVal value As String)**  **Ffam = value**  **End Set**  **End Property**  *'св-во для поля Fm с контролем при изменении значения поля*  **Public Property m() As Integer**  **Get**  **Return Fm**  **End Get**  **Set(ByVal value As Integer)**  **If value > 0 Then**  **Fm = value**  **Else**  **MsgBox("Ошибка при вводе кол-ва экзаменов")**  **Fm = 4**  **End If**  **End Set**  **End Property**  *'св-во только для записи одного значения*  *'массива оценок студента (поля Foc) с*  *'контролем записываемого значения оценки*  **Public WriteOnly Property ocenka(ByVal i As Integer) As Integer**  **Set(ByVal value As Integer)**  **If value > 0 And value <= 5 Then**  **Foc(i) = value**  **Else**  **MsgBox("Ошибка при вводе оценки!")**  **Foc(i) = 0**  **End If**  **End Set**  **End Property**  *'св-во только для чтения целиком*  *'массива оценок студента (поля Foc)*  **Public ReadOnly Property oc() As Integer()**  **Get**  **Return Foc**  **End Get**  **End Property**  *'свойство только для чтения поля средней оценки*  *'т.к. значение этого поля определяется вызовом*  *'метода Sred из метода vvod класса при вводе*  *'данных о студенте и не может изменяться снаружи*  **Public ReadOnly Property so() As Double**  **Get**  **Return Fso**  **End Get**  **End Property**  *'===============МЕТОДЫ==============*  *'подсчет среднего балла студента*  **Public Function Sred() As Double**  **Dim s As Integer = 0**  **For i As Integer = 0 To Fm - 1**  **s = s + Foc(i)**  **Next**  **Return s / Fm**  **End Function**  *'Перегружаемая процедура ввода данных о студенте с клавиатуры*  **Public Overloads Sub vvod()**  **Ffam = InputBox("Фамилия")**  *'присвоение значений массиву оценок - полю Foc()*  *'производится через вызов свойства ocenka для*  *'контроля вводимого значения одной оценки*  **For j As Integer = 0 To Fm - 1**  **ocenka(j) = CInt(InputBox("Оценка"))**  **Next j**  *'после ввода всех оценок студента сразу*  *'подсчитывается его средняя оценка*  **Fso = Sred()**  **End Sub**  *'Перегружаемая процедура ввода данных о студенте из файла*  *'через буферный массив-одну строку файла*  **Public Overloads Sub vvod(ByVal buf() As String)**  *'Первый элемент этого массива-фамилия,следующие-оценк***и**  **Ffam = buf(0)**  *'присвоение значений массиву оценок - полю Foc()*  *'производится через вызов свойства ocenka для*  *'контроля вводимого значения одной оценки*  **For j = 0 To Fm - 1**  **ocenka(j) = CInt(buf(j + 1))**  **Next**  *'после ввода всех оценок студента сразу*  *' подсчитывается его средняя оценка*  **Fso = Sred()**  **End Sub**  *'Метод вывода сведений о студенте*  *'возвращает строку со сведениями*  **Public Function vivod() As String**  **Dim z As String = ""**  **Dim i, L As Integer**  *'L - кол-во букв в фамилии для форматного вывода*  **L = Ffam.Length**  **z = Ffam & Space(15 - L) & "Оценки: "**  **For i = 0 To Fm - 1**  **z = z & CStr(Foc(i)) & Space(3)**  **Next**  **z = z & " Сред. оценка "**  **z = z & Format(Fso, "0.0000")**  **Return z**  **End Function**  **End Class**  *'Базовый класс ГРУППА в композиции с классом СТУДЕНТ*  **Public Class ГРУППА**  *'ПОЛЯ защищённые, чтобы обеспечить их доступность*  *'классам-наследникам*  **Protected Fn As Integer** *'кол-во студентов в группе*  **Protected Fstud() As СТУДЕНТ** *'массив студентов*  **Protected Fsr As Double** *' Общий сред. балл группы за сессию*  *'=============КОНСТРУКТОРЫ================================*  *' Конструктор класса получая кол-во студентов,*  *'заполняет поле Fn через св-во n и выделяет память под Fstud()*  **Public Sub New(ByVal KolCt As Integer)**  **n = KolCt**  **ReDim Fstud(n - 1)**  **End Sub**  *'Пустой конструктор по умолчанию*  **Public Sub New()**  **End Sub**  *'===============СВОЙСТВА===================*  *'св-во для поля Fn с контролем при изменении значения поля*  **Public Property n() As Integer**  **Get**  **Return Fn**  **End Get**  **Set(ByVal value As Integer)**  **If value > 0 Then**  **Fn = value**  **Else**  **MsgBox("Ошибка при вводе кол-ва студентов")**  **Fn = 20**  **End If**  **End Set**  **End Property**  *'св-во для поля Fstud (массива объектов класса СТУДЕНТ)*  **Public Property stud() As СТУДЕНТ()**  **Get**  **Return Fstud**  **End Get**  **Set(ByVal value As СТУДЕНТ())**  **Fstud = value**  **End Set**  **End Property**  *'св-во только для чтения для поля Fsr*  **Public ReadOnly Property sr() As Double**  **Get**  **Return Fsr**  **End Get**  **End Property**  *' ==========МЕТОДЫ=================*  *' подсчет среднего балла всей группы*  **Public Function Sred() As Double**  **Fsr = 0**  **For i As Integer = 0 To Fn - 1**  **Fsr = Fsr + Fstud(i).so**  **Next**  **Fsr = Fsr / Fn**  **Return Fsr**  **End Function**  *'Перегружаемая процедура ввода группы студентов из файла*  **Public Overloads Sub vvodGr(ByVal mas() As String, \_**  **ByVal n As Integer)**  **Dim i, ke As Integer**  **Dim line As String**  **Dim buf() As String**  **For i = 0 To n - 1**  **line = mas(i)**  *'Удаление из строки возможных лишних пробелов*  *'(цикл замены двух пробелов одним)*  **Do While line.IndexOf(Space(2)) >= 0**  **line = line.Replace(Space(2), Space(1))**  **Loop**  **If mas(i) = "" Then**  **n = n - 1**  **MsgBox("В файле есть пустая строка")**  **Exit For**  **Else**  *'Ф-ция Split разбивает строку line на массив*  *'подстрок buf() по символам пробела*  **buf = Split(line, Space(1))**  *'длина массива buf() равна кол-ву экзаменов за*  *'вычетом фамилии студента*  **ke = buf.Length - 1**  *'Для заполнения полей класса ГРУППА необходимо*  *' создать ссылку на экземпляр класса СТУДЕНТ*  **Dim st As СТУДЕНТ**  **st = New СТУДЕНТ(ke)**  *'вызов метода класса СТУДЕНТ для ввода*  *'сведений об одном студенте через*  *'буферный массив*  **st.vvod(buf)**  *'созданный экземпляр класса СТУДЕНТ*  *'присваивается полю класса ГРУППА*  **stud(i) = st**  **End If**  **Next**  *'подсчет среднего балла всей группы*  **Fsr = Sred()**  **End Sub**  *'Перегружаемая процедура ввода группы студентов с клавиатуры*  **Public Overloads Sub vvodGr(ByVal ke As Integer)**  **For i As Integer = 0 To Fn - 1**  *'Для заполнения полей класса ГРУППА необходимо*  *' создать ссылку на экземпляр класс СТУДЕНТ*  **Dim st As СТУДЕНТ**  **st = New СТУДЕНТ(ke)**  *'вызов метода класса СТУДЕНТ для*  *'ввода сведений об одном студенте*  **st.vvod()**  *'созданный экземпляр класса СТУДЕНТ*  *'присваивается полю класса ГРУППА*  **stud(i) = st**  **Next**  *'подсчет среднего балла всей группы*  **Fsr = Sred()**  **End Sub**  *'Метод для вывода сведений о группе студентов*  *'все сведения "собираются" в одну строку*  *'для последующего вывода в TextBox*  **Public Function vivod() As String**  **Dim z As String = ""**  **Dim i As Integer**  **Dim st As СТУДЕНТ**  **For i = 0 To Fn - 1**  **st = stud(i)**  *'сведения о каждом студенте "приклеиваются"*  *'к строке-результату z*  *'вызовом метода vivod класса СТУДЕНТ*  **z = z & st.vivod**  **z = z & vbNewLine**  **Next i**  **Return z**  **End Function**  **End Class**  *'=====================================================*  *'абстрактный класс,производный от класса ГРУППА*  **Public MustInherit Class РЕЗУЛЬТАТ**  **Inherits ГРУППА**  *'ПОЛЯ защищённые, чтобы обеспечить их доступность*  *'классам-наследникам*  **Protected Fk As Integer** *'кол-во студ, к-рым надо*  *'повысить успеваемость*  **Protected Ff() As String** *'массив их фамилий*  **Protected Fsb() As Double** *'массив их средних баллов*  *'абстрактный виртуальный метод сортировки, к-рый*  *'должен быть переопределен в наследниках*  *'именно из-за наличия этого абстрактного метода*  *'(т.е. не определенного) класс абстрактный*  **MustOverride Sub sort()**  *'конструктор с параметром n - общего кол-ва студентов в группе*  **Public Sub New(ByVal n As Integer)**  **MyBase.new(n)** *'вызов конструктора базового класса ГРУППА*  **ReDim Ff(n - 1), Fsb(n - 1)**  **End Sub**  *'конструктор по умолчанию*  **Public Sub New()**  **End Sub**  *'Все свойства только для чтения, так как значения полей*  *'определяются методом FormirSpisok самого класса*  *' и не могут изменяться снаружи*  **Public ReadOnly Property k() As Integer**  **Get**  **Return Fk**  **End Get**  **End Property**  **Public ReadOnly Property f() As String()**  **Get**  **Return Ff**  **End Get**  **End Property**  **Public ReadOnly Property sb() As Double()**  **Get**  **Return Fsb**  **End Get**  **End Property**  *'Перегружаемые процедуры вывода данных-массивов*  **Public Overloads Sub VivodMas(ByVal a() As String, \_**  **ByRef z As String)**  **Dim i As Integer**  **z = ""**  **For i = 0 To UBound(a)**  **z = z + a(i) + vbNewLine**  **Next**  **End Sub**  **Public Overloads Sub VivodMas(ByVal a() As Double, \_**  **ByRef z As String)**  **Dim i As Integer**  **z = ""**  **For i = 0 To UBound(a)**  **z = z + Format(a(i), "0.0000") + vbNewLine**  **Next**  **End Sub**  *' Формирование списка студентов группы*  *' имеющих средний балл ниже общего среднего*  **Public Sub FormirSpisok()**  **Dim i As Integer**  **Fk = 0**  **For i = 0 To Fn - 1**  **If Fstud(i).so < sr Then**  **Ff(Fk) = Fstud(i).fam**  **Fsb(Fk) = Fstud(i).so**  **Fk = Fk + 1**  **End If**  **Next**  **ReDim Preserve Ff(Fk - 1), Fsb(Fk - 1)**  **End Sub**  **End Class**  *'==========================================================*  *'1 класс-наследник абстрактного класса РЕЗУЛЬТАТ*  **Public Class СписПоУбыв**  **Inherits РЕЗУЛЬТАТ**  *'Конструкторы*  **Public Sub New(ByVal n As Integer)**  **MyBase.New(n)**  **End Sub**  **Public Sub New()**  **End Sub**  *'переопределенный метод базового класса в классе-наследнике*  *'сортировка по убыванию среднего балла студента*  **Public Overrides Sub sort()**  **Dim i, j As Integer, temp As String, t As Double**  **For i = 0 To Fk - 2**  **For j = i + 1 To Fk - 1**  **If Fsb(i) < Fsb(j) Then**  **t = Fsb(i) : Fsb(i) = Fsb(j) : Fsb(j) = t**  **temp = Ff(i) : Ff(i) = f(j) : Ff(j) = temp**  **End If**  **Next**  **Next**  **End Sub**  **End Class**  *'==========================================================*  *'2 класс-наследник абстрактного класса РЕЗУЛЬТАТ*  **Public Class СписПоАлфав**  **Inherits РЕЗУЛЬТАТ**  *'Конструкторы*  **Public Sub New(ByVal n As Integer)**  **MyBase.New(n)**  **End Sub**  **Public Sub New()**  **End Sub**  *'переопределенный метод базового класса в классе-наследнике*  *'сортировка студентов по алфавиту*  **Public Overrides Sub sort()**  **Dim i, j As Integer, temp As String, t As Double**  **For i = 0 To Fk - 2**  **For j = i + 1 To Fk - 1**  **If Ff(i) > Ff(j) Then**  **temp = Ff(i) : Ff(i) = f(j) : Ff(j) = temp**  **t = Fsb(i) : Fsb(i) = Fsb(j) : Fsb(j) = t**  **End If**  **Next**  **Next**  **End Sub**  **End Class**  *'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *'1 форма*  *'на этой форме вводятся и выводятся исходные данные о студентах*  **Option Strict On**  **Public Class Form1**  *'объявление двух глобальных переменных(не экземпляров!)*  *'для двух классов-наследников необходимо,*  *'т.к. они должны быть доступны для двух событийных процедур*  *'(кнопки ввода из файла и ввода с клавиатуры)*  *'и после создания объекты этих классов*  *'должны быть доступны и на второй форме*  **Public SpAlf As СписПоАлфав**  **Public SpUb As СписПоУбыв**  *'Процедура чтения сведений о студентах из текстового файла*  *'в одну строку*  **Sub vvodFile(ByVal filename As String, ByRef S As String)**  *' Открываем существующий файл и загружаем его текстовое*  *'содержимое целиком в одну строку S.*  **Try**  **S = My.Computer.FileSystem.ReadAllText(filename)**  **Catch ex As Exception**  **MsgBox("ошибка при чтении файла")**  **End Try**  **End Sub**  *'кнопка для ввода с клавиатуры*  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button1.Click**  *'если выбрана кнопка для ввода данных с клавиатуры,*  *'делаем недоступной 2-ю кнопку для ввода из файла*  **Button2.Enabled = False**  **Dim kolSt, ke As Integer**  **Dim s As String = ""**  **kolSt = CInt(InputBox("Введите количество студентов"))**  **ke = CInt(InputBox("Введите количество экзаменов"))**  *'создание экземпляров двух классов через конструкторы*  **SpUb = New СписПоУбыв(kolSt)**  **SpAlf = New СписПоАлфав(kolSt)**  *'создаём два объекта разных классов*  *'с одинаковыми значениями полей:*  *'вызываем метод ввода группы студентов*  *'с клавиатуры для одного из объектов*  **SpUb.vvodGr(ke)**  *'заполняем поле Fstud() массива студентов*  *'объекта 2-го класса, копируя значение*  *'этого поля из 1-го объекта через свойство*  **SpAlf.stud = SpUb.stud**  *'заполнить поле Fsr среднего балла группы в объекте*  *'SpAlf,также копируя его значение из 1-го объекта SpUb*  *'нельзя, т.к. св-во для этого поля только для чтения*  *'поэтому вызываем метод подсчета среднего балла группы*  **SpAlf.Sred()**  *'так как объекты двух классов одинаковы*  *'вызываем метод вывода сведений о группе*  *'один раз для любого из объектов*  **s = SpUb.vivod()**  **TextBox1.Text = s**  *'вывод общего среднего балла группы*  **TextBox2.Text = CStr(SpUb.sr)**  *'Кнопку3 делаем доступной после ввода данных*  *'и расчета средних баллов*  **Button3.Enabled = True**  **End Sub**  *'кнопка для ввода из файла*  **Private Sub Button2\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button2.Click**  *'если выбрана кнопка для ввода данных из файла,*  *'делаем недоступной 1-ю кнопку для ввода с клавиатуры*  **Button1.Enabled = False**  **Dim s As String = ""**  **Dim kolSt As Integer**  *'Вызов проц-ры чтения сведений о студентах из текстового*  *'файла целиком в одну строку s*  **vvodFile("myfile.txt", s)**  **Dim mas() As String**  *'Ф-ция Split разбивает строку s на массив подстрок mas()*  *'по символам конца строки vbNewLine*  **mas = Split(s, vbNewLine)**  **kolSt = mas.Length** *'кол-во эл-тов получ. массива равно*  *'кол-ву студентов*  *'создание экземпляров двух классов через конструкторы*  **SpUb = New СписПоУбыв(kolSt)**  **SpAlf = New СписПоАлфав(kolSt)**  *'создаём два объекта разных классов*  *'с одинаковыми значениями полей*  *'для заполнения полей в двух классах*  *'вызываем процедуру ввода группы из файла*  *'через массив строк файла два раза*  **SpUb.vvodGr(mas, kolSt)**  **SpAlf.vvodGr(mas, kolSt)**  *'так как объекты двух классов одинаковы*  *'вызываем метод вывода сведений о группе*  *'один раз для любого из объектов*  **s = SpUb.vivod()**  **TextBox1.Text = s**  *'вывод общего среднего балла группы*  **TextBox2.Text = CStr(SpUb.sr)**  *'Кнопку3 делаем доступной после ввода данных*  *'и расчета средних баллов*  **Button3.Enabled = True**  **End Sub**  *'Кнопка "Показать решение"*  **Private Sub Button3\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button3.Click**  **Form2.Show()**  **End Sub**  **End Class**  *'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *'2 форма*  *'на этой форме выводятся списки студентов,*  *'которым нужно повысить успеваемость*  **Option Strict On**  **Public Class Form2**  *'код событийных процедур кнопок Button1 и Button2*  *'практически одинаков и отличается только тем,*  *'ссылка на какой объект-наследник присваивается*  *'переменной Res базового класса*  *'===============================*  *' КНОПКА для сортировки по убыванию*  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button1.Click**  **Dim z As String = ""**  *'объявление переменной(не экземпляра!)базового абстрактного*  *'класса нужно, чтобы присвоить ей ссылку на экземпляр*  *'класса-наследника,обеспечив полиморфизм*  **Dim Res As РЕЗУЛЬТАТ**  *'присвоение переменной базового класса*  *'(т.е. ссылке с типом базового класса)*  *' значения ссылки на объект 1-го класса-наследника*  *'(в котором список по убыванию)*  **Res = Form1.SpUb**  *'вызов метода формирования списка "нерадивых" студентов*  **Res.FormirSpisok()**  *'Способ сортировки будет определяться значением ссылки*  *'базового класса, т.е. по убыванию средней оценки*  **Res.sort()**  *'вывод результирующих полей-массивов(фамилий и сред. оценки)*  **Res.VivodMas(Res.f, z)**  **TextBox1.Text = z**  **Res.VivodMas(Res.sb, z)**  **TextBox2.Text = z**  *'вывод общего среднего балла группы для наглядности*  **TextBox5.Text = Format(Res.sr, "0.0000")**  **End Sub**  *' КНОПКА для сортировки по алфавиту*  **Private Sub Button2\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button2.Click**  **Dim z As String = ""**  *'объявление переменной(не экземпляра!)базового абстрактного*  *'класса нужно, чтобы присвоить ей ссылку на экземпляр*  *'класса-наследника,обеспечив полиморфизм*  **Dim Res As РЕЗУЛЬТАТ**  *'присвоение переменной базового класса*  *'(т.е. ссылке с типом базового класса)*  *' значения ссылки на объект 2-го класса-наследника*  *'(в котором список по алфавиту)*  **Res = Form1.SpAlf**  *'вызов метода формирования списка "нерадивых" студентов*  **Res.FormirSpisok()**  *'Способ сортировки будет определяться значением ссылки*  *'базового класса, т.е. по алфавиту*  **Res.sort()**  *'вывод результирующих полей-массивов(фамилий и сред. оценки)*  **Res.VivodMas(Res.f, z)**  **TextBox3.Text = z**  **Res.VivodMas(Res.sb, z)**  **TextBox4.Text = z**  *'вывод общего среднего балла группы для наглядност***и**  **TextBox5.Text = Format(Res.sr, "0.0000")**  **End Sub**  *'Кнопка "Конец"*  **Private Sub Button3\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button3.Click**  **End**  **End Sub**  *' Кнопка для повторения*  **Private Sub Button4\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button4.Click**  **Me.Close()**  **Form1.TextBox1.Text = "" : Form1.TextBox2.Text = ""**  **Form1.TextBox3.Text = "" : Form1.TextBox4.Text = ""**  **Form1.Button1.Enabled = True**  **Form1.Button2.Enabled = True**  **Form1.Button3.Enabled = False**  **Form1.Show()**  **End Sub**  **End Class** | |

*Рис. 3.6-7. Программный код проекта*

1. ***Схема объектной модели на уровне реализации, построенная по программному коду.***

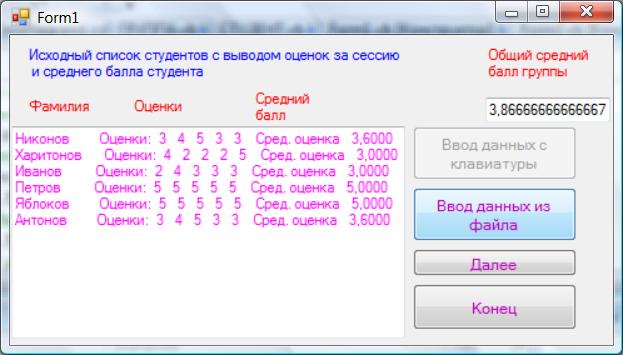
Схема классов уровня реализации здесь приведена в качестве иллюстрации. Эта схема (рис. 3.6-8) была получена по готовому программному коду проекта VS. Чтобы получить иерархию классов, необходимо в окне **Обозреватель решений** выделить *имя проекта*, а затем выбрать мышью иконку ***Перейти к схеме классов***, в результате чего откроется окно **ClassDiagram1.cd** (см. Приложение 2).



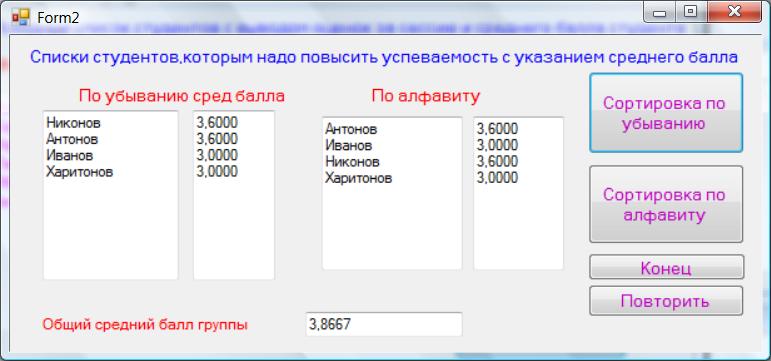
*Рис. 3.6-8. Схема классов уровня реализации*

1. ***Результаты выполнения приложения.***

Результаты выполнения проекта приведены на рис. 3.6-9 и рис. 3.6-10.



*Рис. 3.6-9. Результаты решения задачи при вводе данных   
из файла на Form1*



*Рис. 3.6-10. Результаты решения задачи при выводе данных на Form2*

1. ***Обоснуйте правильность работы проекта.***

Обоснование правильности работы программы приведите самостоятельно

### 3.7. Контрольные вопросы

1. Для чего используется наследование?
2. Опишите синтаксис производного класса. Какие спецификаторы доступа применяются в иерархиях?
3. Как вызвать метод базового класса из производного класса?
4. Опишите порядок вызова конструкторов базовых классов при работе конструктора производного класса.
5. Какие ключевые слова используется при переопределении методов базового класса в производном?
6. Опишите механизмы раннего и позднего связывания.
7. Какой метод называется виртуальным методом? Для чего применяются виртуальные методы?
8. Чем отличаются обычные методы от методов виртуальных классов?
9. Для чего используются абстрактные классы?
10. Назовите альтернативы наследованию классов.
11. Какие отношения между классами Вы знаете?
12. Что такое композиция классов?
13. Чем отличается композиция классов от агрегации?
14. Что наследует производный класс от базового класса?
15. Какие члены базового класса доступны производному классу?
16. Какие члены класса имеют модификатор доступа **Protected**? Зачем они нужны?
17. Что такое перегрузка методов?
18. Что такое экранирование методов?
19. Что означает переопределение методов?
20. Чем отличается экранирование от переопределения? Какие ключевые слова используются при этом?
21. Как проявляется полиморфизм?
22. Что такое интерфейс (интерфейсные классы)?
23. Какой класс может считаться классом, реализующим интерфейс?
24. Что общего, и какие отличия абстрактных и интерфейсных классов?