## Тема 2. Объектная модель, средства описания и использования классов в приложениях на Visual Basic

2.1. Объектная модель и ее составные части

2.2. Средства VB создания приложений с классами

[2.3. [Две роли классов в ООП и типы данных](#_5.3.1.__Две)](file:///F:\2015УчП-Раз-5-Инф-Шакин-П+Л\КНИГА-4\Тема-05-00-Огл-Пред-Введ%20ПРОВ-23_11_2014.docx#_5.3.2.__Средства)

[2.4. Взаимодействие, взаимное различие и сходство форм, модулей и классов](file:///F:\2015УчП-Раз-5-Инф-Шакин-П+Л\КНИГА-4\Тема-05-00-Огл-Пред-Введ%20ПРОВ-23_11_2014.docx#_5.3.3._Взаимодействие,_взаимное)

[2.5. Пример разработки программных приложений с использованием классов](file:///F:\2015УчП-Раз-5-Инф-Шакин-П+Л\КНИГА-4\Тема-05-00-Огл-Пред-Введ%20ПРОВ-23_11_2014.docx#_5.3.4._Создание_объектной)

[2.6. 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#_5.3.6.__Задачи)

2.7. Лабораторная работа по теме «Объектная модель, средства описания и

использования классов в приложениях на Visual Basic»

2.8. Контрольные вопросы

### 2.1. Объектная модель и ее составные части

.

Человек привык существовать в мире, который состоит из объектов.

***Объект*** – это *понятие, абстракция* или любая *сущность* с четко очерченными границами детализации, которая обладает *состоянием,* *поведением* и имеет смысл в контексте рассматриваемой прикладной проблемы (предметной области). ***Объектно-ориентированная технология (ООТ)*** – это иной подход к решению задач по разработкеПО, чем, например, ***технология структурного программирования***. У разработчиков ПО средствами ООТ масса преимуществ, причем это относится не только к созданию более эффективного программного кода, но и к ***модификации и расширению возможностей*** уже имеющихся программных систем.

Необходимо отметить, что собственно программирование, прежде всего, подразумевает правильное и эффективное использование средств и механизмов конкретных языков программирования, а проектирование, основное внимание уделяет правильному и эффективному структурированию программных систем.

Объектно-ориентированное проектирование базируется на двух важнейших аспектах:

* на объектно-ориентированной декомпозиции;
* на многообразии приемов представления моделей, отражающих логическую (классы и объекты) и физическую (модули и процессы) структуру системы, а также ее статические и динамические аспекты.

Для объектно-ориентированного стиля концептуальная база – это объектная модель, и этот стиль характеризуется четырьмя основными понятиями, о которых уже упоминали во введении и вкратце повторим сейчас: абстрагирование, инкапсуляция, модульность, иерархия.

***Абстракция*** – выделяет существенные характеристики некоторого объекта предметной области, отличающие его от всех других видов объектов и, таким образом, четко определяет его концептуальные границы с точки зрения наблюдателя, пользователя и предметной области.

Абстрагирование позволяет концентрировать внимание на некоторых особенностях объекта и отделить самые существенные особенности поведения элементов предметной области от несущественных с точки зрения решаемой задачи и предметной области. Выбор правильного набора абстракций (объектов, классов объектов) для заданной предметной области представляет собой одну из главных задач объектно-ориентированного анализа.

***Инкапсуляция*** – это механизм, который объединяет данные и методы обработки этих данных в единое целое и защищает их от внешнего вмешательства или неправильного использования, скрывая детали реализации того или иного фрагмента программы. Этот фрагмент может быть доступен только через свой внешний интерфейс.

***Модульность*** – разработанные объекты, представляющие из себя инкапсулированную абстракцию с четко определенным интерфейсом, сохраняются в отдельном программном модуле (модуле класса). Не путать со стандартным модулем (модулем пользователя), который мы рассматривали в п. 1.2.

***Иерархия –*** объекты подвергаются иерархической классификации сверху вниз - разделению на группы (классы объектов), объединенные общими признаками. Например, яблоко сорта **«АНТОНОВКА»** можно отнести к классу «**ЯБЛОКО**», который в свою очередь, является частью класса «**ФРУКТЫ**», входящего в состав класса «**ЕДА**». Класс «**ЕДА**» обладает некоторыми характеристиками (съедобный, питательный и т.д.), которые также применимы к его подклассу «**ФРУКТЫ**». Кроме этих характеристик, класс **«ФРУКТЫ»** имеет дополнительные характеристики (сочный, сладкий, и т.д.), которые отличают его от других видов еды. Класс «**ЯБЛОКО**» определяет дополнительные характеристики (нетропический, растет на дереве и т.д.). Яблоко сорта **«АНТОНОВКА»**, в свою очередь, наследует все характеристики предшествующих классов иерархии и имеет свои уникальные. При использовании наследования для объекта нужно определять только те характеристики, которые делают этот объект уникальным внутри его класса, а основные атрибуты он может унаследовать от родительского класса. Если не использовать наследование, то для каждого объекта необходимо явно определить все его характеристики. Таким образом, благодаря механизму наследования объект может быть конкретным экземпляром более общего случая. Т.е. иерархия создается на основе наследования и представляется в виде дерева, где более общие объекты (родительские, базовые) располагаются ближе к корню, а более специализированные (производные, наследники, потомки) – на ветвях и листьях. Подробно механизм наследования будет рассмотрен в **Теме 3**.

Итак, объекты реального мира (предметной области) характеризуются состоянием, которое описывается данными, их еще называют характеристиками или атрибутами. Так, например, атрибутами автомобиля являются максимальная скорость, мощность двигателя, цвет кузова и т. д. Атрибутами усилителя являются частотный диапазон, выходная мощность, коэффициент нелинейных искажений, уровень шума.

Помимо атрибутов, объекты обладают поведением – некоторыми функциональными возможностями, которые называют еще операциями. Так, автомобиль может ездить, корабль – плавать, компьютер – производить вычисления.

В ООП объект – это экземпляр некоторого класса объектов или просто класса. Так, автомобиль Audi 6 является экземпляром класса автомобилей данной объектной модели. Таким образом, класс – это абстрактное понятие.

Объектно-ориентированная технология [5] базируется на так называемой объектной модели. Объектная модель описывает структуру объектов, составляющих программную систему, их атрибуты, операции и взаимосвязи с другими объектами. В объектной модели должны быть отражены те понятия и объекты реального мира, которые важны для разрабатываемой программной системы. Цель разработки объектной модели – описать предметную область с помощью объектов, составляющих в совокупности проектируемую программную систему, а также выявить и указать различные зависимости между объектами.

Для того чтобы построить объектную модель, необходимо проанализировать предметную область и:

1. выделить взаимодействующие объекты (сущности), с помощью которых можно описать поведение моделируемой системы на необходимом уровне детализации и их взаимодействие;
2. определить атрибуты выделенных объектов, существенные для данной задачи;
3. описать возможные операции объектов, которые объекты могут выполнить;
4. определить функциональные зависимости между классами.

Основным понятиям и сущностям предметной области в объектной модели ставятся в соответствие классы и их члены, а основными средствами описания и моделирования программных проектов является UML (унифицированный язык моделирования).

**Унифицированный язык моделирования** (**UML**) можно условно называть графическим языком, предназначенным для моделирования компьютерных программ, т.е. для наглядной визуальной интерпретации программ высокоуровневой организации. Весь UML состоит из четырех частей, описывающих различные аспекты системы: ста­тические, динамические, организационные и относящиеся к окружению. Наиболее важным средством UML является набор различных диаграмм. Диаграммы классов иллюстрируют отношения между классами, диаграммы объектов – между отдельными объектами, диаграммы связей отражают связь объектов во времени и т.д. Таким образом, с помощью UML можно легко разработать и иллюстрировать структуру программы.

Необходимо отметить, что на практике определение основных понятий предметной области и представление их в виде классов является нетривиальной задачей и зависит от квалификации разработчиков.

Возьмем за основу пример из [11] и на его базе построим концептуальную модель предметной области ***движения автомобилей на шоссе и объектную модель на логическом уровне***. Эти модели построим средствами **Диаграмм** UML-моделирования в конструкторе классов Visual Studio (Приложение 1). Эти Диаграммы можно построить и с помощью графической программы Visio.

**Пример 2.1-1. Построить концептуальную модель и логическую объектную модель движения автомобилей на шоссе для определения, достаточна ли его пропускная способность.**

Проведем анализ заданной предметной области.

1. Проанализируем движение автомобилей на шоссе:

В этой задаче можно выделить такие объекты, как **ДОРОГА** и двигающаяся по ней **МАШИНЫ**. По дороге может двигаться несколько **МАШИН**, причем все они с точки зрения нашей задачи имеют одинаковые атрибуты. Поэтому нет смысла описывать каждую машину отдельно: достаточно один раз определить их общие черты, а потом просто сказать, что все машины ими обладают. В ООП, как было сказано ранее, для этой цели существует специальное понятие – класс объектов, т.е. объектов, имеющих одинаковый набор атрибутов, структуру и поведение. В нашей задаче можно ввести две сущности (класса) – **TROAD (ДОРОГА)** и **TCAR (МАШИНА)**, причем по условию **ДОРОГА** одна, а **МАШИН** может быть много. Для упрощения будем считать, что:

* все **МАШИНЫ** одинаковы;
* все **МАШИНЫ** движутся слева направо с постоянной скоростью (при этом скорости разных **МАШИН** могут быть различны;
* по каждой полосе **ДОРОГИ** едет только одна **МАШИНА**, и можно не учитывать обгон и переход на другую полосу **ДОРОГИ**;
* если **МАШИНА** выходит за правую границу **ДОРОГИ**, вместо нее слева на той же полосе появляется новая **МАШИНА**.

1. Будем рассматривать только прямой отрезок **ДОРОГИ**, тогда класс **TROAD** имеет две важных для нашей задачи характеристики: **длину** (**Len**) и **число полос** (**Width**) движения. Эти атрибуты определяют состояние **ДОРОГИ**. Поведение (операции) дороги может заключаться в том, что число полос движения может меняться, например, из-за ремонта, но в нашей простейшей задаче объект **ДОРОГА** не будет изменяться. Итак, в нашей модели **ДОРОГИ** два атрибута.
2. Теперь рассмотрим объекты класса **TCAR**. Их важнейшие характеристики – **координаты** и **скорость** движения. За координаты **МАШИНЫ** примем расстояние **X** от левого края рассматриваемого участка шоссе и номер полосы **Y** (натуральное число). Скорость автомобиля **V** в нашей модели – неотрицательное число.
3. Что касается поведения **МАШИНЫ**, то в нашей модели она может выполнять всего одну операцию – ехать в заданном направлении, т.е. имеет операцию **Двигаться (Move)**.

Таким упрощенным способом (рис. 2.1-1) мы построили только модели отдельных объектов (точнее – классов). Чтобы построить модель предметной области (всей системы), нужно знать, как эти объекты взаимодействуют. Предположим, что объект **МАШИНА** должен уметь определить, что закончился рассматриваемый участок дороги. Для этого машина должна постоянно обращаться к объекту **ДОРОГА**, запрашивая длину дороги (это показывает стрелка на рис. 2.1-2).

В нашей модели **ДОРОГА** пока ни одной операции.



*Рис. 2.1-1. Диаграмма концептуальной модели*   
*предметной области* **ДОРОГА-МАШИНА**

Диаграммы, приведенные на рис. 2.1-1 и рис. 2.1-2 показывают:

* характеристики класса объектов;
* операции, которые они могут выполнять;
* характер обмена данными между объектами.



*Рис. 2.1-2. Диаграмма объектной модели*   
*предметной области* **ДОРОГА-МАШИНА** *на логическом уровне*

В этой концептуальной модели пока ничего не говорится о том, как построены объекты и как именно они будут выполнять операции. В то же время согласно принципам ООП, ни один объект не должен зависеть от внутреннего устройства и алгоритмов работы других объектов. Важно только, чтобы они соблюдали интерфейс связей между объектами и внешней средой – правила, описывающие взаимодействия «своих» объектов с остальными.

Таким образом, ***объектно-ориентированный анализ (ООА)*** направлен на создание моделей реальной предметной области на основе объектно-ориентированного мировоззрения – это методология, при которой требования к системе воспринимаются с точки зрения классов и объектов, выявленных в предметной области. По результатам ООА формируются модели, на которых основывается ***объектно-ориентированное проектирование (ООПр)***, которое, в свою очередь, создает фундамент для окончательной реализации программной системы, используя методологию и средства ***объектно-ориентированного программирования.***

Ключевым средством, необходимым для реализации объектной модели на языке программирования VB, как и других объектно-ориентированных языков, является ***класс***.

***Классы,*** являясь основными единицами построения программных проектов на этапе программирования, описывают объекты с общим набором характеристик (атрибутов) и действий (операций).

Далее рассмотрим средства определения и использования в приложениях классов и объектов.

### 2.2. Средства VB создания приложений с классами

#### 2.2.1. Средства определения пользовательских классов

Для определения класса в программном коде используется оператор Class**.** Простейшее описание класса следующее:

|  |
| --- |
| Class*имя\_класса*  *Тело\_класса*  End Class |

Оператор Class объявляет имя класса. В *Теле\_класса* могут быть объявлены ***константы***, ***поля***, ***свойства***, ***методы***, ***конструкторы***, ***события, делегаты и другие классы***, которыеобобщенно называют ***членами класса***:

* поля – соответствуют атрибутам объекта и содержат данные класса, т.е. переменные, принадлежащие классу или объекту класса (принадлеж­ность к классу или экземпляру класса характерна не только для полей, но и для методов, событий и свойств);
* свойства – синтаксическая надстройка, позволяющая осуществлять вызов функции, синтаксически аналогичный чтению/записи переменной (например, можно объявить свойство Возраст, и при попытке записи в него отрицательного значения выдавать ошибку). Таким образом, свойство – это способ доступа к внутреннему состоянию объекта класса, имитирующий обращение к закрытому полю класса;
* методы – процедуры класса, соответствуют операциям;
* события – синтаксическая надстройка, поддерживаемая компилятором и средой VB, которая позволяет вызывать методы других объектов, подписавшихся на данное событие (например, подписавшись на событие Нажатие объекта Кнопка, подписавшийся объект каждый раз при нажатии кнопки будет получать уведомление в виде вызова метода). Таким образом, события определяют уведомления, которые может генерировать класс (будут рассмотрены в Теме 4).

Основу любого класса составляют его ***поля и методы***, а также специальные методы – ***конструкторы***.

В проекте VB новый класс создается аналогично стандартному модулю. То есть, чтобы создать в проекте новый класс, необходимо щелкнуть на кнопке ***Добавление нового элемента*** стандартной панели инструментов или на команде ***Добавить Новый элемент…*** элемента Главного меню   
**Проект**. Также можно использовать команду ***Добавить Класс…*** элемента Главного меню **Проект*.*** В результате выполнения одной из этих команд откроется диалоговое окно **Добавление нового элемента …**, в котором следует выб­рать шаблон ***Class*** и указать имя класса.

После нажатия на кнопку ***Добавить*** в окне **Редактор кода** по­явится заготовка описания нового класса:

|  |
| --- |
| **Public Class Class1**  **…**  **End Class** |

Класс проекта, созданный первым, по умолчанию имеет имя **Class1,** а файл, где находится программный код этого класса, имеет имя **Class1.vb**. Это имя можно изменить в окне **Добавление нового элемента** или, сделав щелчок пра­вой кнопкой мыши на ***модуле класса*** в окне **Обозреватель решений,** ввести новое ***имя класса***.

**Пример 2.2-1. Создать программный проект для модели, имитирующей движение автомобилей на шоссе, рассмотренной в Примере 2.1-1 и представленной на рис. 2.1-2.**

Самый простой класс **TROAD**, который моделирует **ДОРОГУ**. Объекты этого класса имеют две характеристики: **длину,** которая может быть вещественным числом и **ширину** – количество полос, целое число. Для хранения значений этих характеристик используются поля **Len** и **Width**, принадлежащие классу **TROAD**.

|  |
| --- |
| *'Oбъявление класса ДОРОГА*  **Public Class** **TROAD**  *'Открытые поля(данные) класса*  **Public** **Len** **As Double** *'длина дороги*  **Public** **Width** **As Integer** *'ширина(кол-во полос)*  **End Class** |

Это описание нового типа данных – класса **TROAD** (условимся здесь начинать имя класса с буквы **T** – это сокращение от слова type), сообщение компилятору, что в программе, возможно, будут использоваться объекты этого типа. При этом в памяти не создается ни одного объекта. Это описание – как чертеж, по которому в нужный момент можно построить сколько угодно таких объектов. Для работы с объектом класса **TROAD** в программе нужно объявить соответствующую переменную – имя объекта.

|  |
| --- |
| **Dim road As** **TROAD** |

Однако и это еще не объект, а ссылка (про ссылочный тип данных подробно поговорим в п. 2.3), то есть переменная, в которой можно сохранить адрес любого объекта класса **TROAD**. Чтобы создать сам объект в памяти, нужно вызвать специальный метод **New**, который называется ***конструктором***. Адрес нового объекта записывается в переменную **road**:

|  |
| --- |
| **road = New TROAD** |

***Конструктор*** – это метод **New** класса, который вызывается для создания объекта этого класса. Он добавляется ко всем классам по умолчанию, если не описан явно, и при его вызове все числовые переменные (поля) объекта заполняются нулями. Значения полей можно изменить с помощью присваивания и точечной нотации. Таким образом, программный код событийной процедуры, которая создает объект класса **TROAD (ДОРОГА)** и задаёт значения его полям,будет следующим:

|  |
| --- |
| **Private Sub** **Button1\_Click(ByVal** **sender** **As** **System.Object, \_**  **ByVal** **e** **As** **System.EventArgs) \_**  **Handles** **Button1.Click**  **Dim** **road** **As** **TROAD**  **road =** **New** **TROAD** *'вызов конструктора по умолчанию*  **road.Len = 60** '*присваивание полю класса значения*  **road.Width = 3**  **End Sub** |

Начальные значения полей можно задавать прямо при создании объекта. Для этого нужно добавить в описание класса новый конструктор явным образом, которому будут передаваться два параметра – начальные значения **длины Len** и **ширины** **Width** дороги.

Т.о. если программист не определяет конструктор класса, то, как мы уже говорили, к классу автоматически добавляется конструктор по умолчанию – конструктор без параметров. Однако если программист сам создает один или несколько конструкторов с параметрами, то автоматического добавления конструктора по умолчанию (без параметров) не происходит и поэтому конструктор без параметров необходимо написать в коде класса явным образом, если это необходимо.

В коде нового конструктора с параметрами можно проверить правильность переданных параметров, чтобы по ошибке длина и ширина дороги не оказались нулевыми или отрицательными. При вызове этого конструктора создается объект, в котором значения полей уже инициализированы.

|  |
| --- |
| **road = New TROAD(60, 3)** *'вызов конструктора с параметрами* |

Таким образом, в классе **TROAD** имеется два конструктора (два метода с одним именем **New**). Так реализован один из основных принципов объектно-ориентированного программирования – **полиморфизм**, представленный в данном случае **перегрузкой** методов (когда в классе методы имеют **одно** имя и **разные** параметры). Как мы увидим в дальнейшем,   
**п**о**лиформизм** может быть представлен и другими средствами**.**

В VS используется основанная на **UML** (унифицированном языке моделирования), но немного модифицированная система графического представления Схем классов на основе существующего программного кода (**Приложение 2**). Поэтому на рис. 2.2-1 помимо программного кода класса **TROAD**, также приведено изображение Схемы класса **TROAD.** Чтобы его получить, необходимо в окне **Обозреватель Решений** выделить имя проекта правой кнопкой мыши и во всплывающем контекстном меню выбрать команду ***Перейти к схеме классов*** *(подробнее см. в* **Приложении 2***).*

|  |  |
| --- | --- |
| *'Oписание класса ДОРОГА*  **Public Class** **TROAD**   |  | | --- | |  |   *'Открытые поля(данные) класса*  **Public** **Len** **As Double** *'длина дороги*  **Public** **Width** **As Integer** *'ширина(кол-во полос)*  *'Конструктор с параметрами*  **Public Sub New**(**ByVal** **Len0** **As Double**, **\_**  **ByVal** **W0** **As Integer**)  **If** **Len0 > 0** **Then**  **Len = Len0**  **Else**  **Len = 100**  **End If**  **If** **W0 > 0** **Then**  **Width = W0**  **Else**  **Width = 1**  **End If**  **End Sub**  *'Пустой конструктор по умолчанию*  **Public Sub New()**  **End Sub**  **End Class** |

*Рис. 2.2-1. Программный код и схема класса* **TROAD**   
*с двумя конструкторами*

Теперь опишем класс **TCAR (МАШИНА)**. Объекты этого класса имеют три атрибута (поля), описывающих состояние объекта, и одну операцию (метод) **Двигаться()** – процедуру **Move()**, которая описывает поведение объекта. **Координата** – **X** и **скорость** – **V** это вещественные значения, а **номер полосы** – **Y** целое. Так как объекты – **МАШИНЫ** должны обращаться к объекту **ДОРОГА** (чтобы узнать её **длину**), в область данных класса включено дополнительное поле **R**. Это, разумеется, не значит, что в состав машины входит дорога. Напомним, что это только ссылка и сразу после создания нужно записать в нее адрес заранее созданного объекта **ДОРОГА**. Эту привязку удобно сделать прямо в конструкторе, при создании объекта. Заодно в конструкторе определяется **полоса** движения и **скорость**, а начальная **координата X** автоматически устанавливается в ноль. В методе **Move()** нужно вычислить новую **координату –** X машины, и, если она находится за пределами дороги, установить ее в ноль (машина появляется слева на той же полосе). Изменение координаты при равномерном движении описывается формулой **X = X0 + V·Δt**, где **X0** и **X** - начальная и конечная координаты, **V** - **скорость**, а **Δt** – время движения. Любое моделирование физических процессов на компьютере происходит в дискретном времени, с некоторым интервалом дискретизации. Для простоты можно измерять время в этих интервалах, а за скорость **V** принять расстояние, проходимое машиной за один интервал.

Программный код класса **TCAR** с методом **Move()**, который описывает изменение движения машины за один интервал времени (**Δt=1**), представлен на рис. 2.2-2.

|  |  |
| --- | --- |
| *'Oписание класса МАШИНА*  **Public Class TCAR**  *'Поля*  **Public X As Double** *'координата машины на дороге*  **Public Y As Integer** *'номер полосы,по к-рой едет машина* **Public V As Double** *'скорость*  *'ссылка на конкретную дорогу(уже созданный объект класса TRoad)-*  *'это дополнительное поле для программирования метода Move()*  **Public R As TROAD**  *'==================================================*  *' Конструктор с параметрами*  **Public Sub New(ByVal R0 As TROAD, \_**  **ByVal y0 As Integer, ByVal v0 As Double)**  **R = R0 : Y = y0 : V = v0**  **End Sub**   |  | | --- | |  |   *' Пустой конструктор по умолчанию*  **Public Sub New()**  **End Sub**  *'=================================*  *' Метод класса*  **Public Sub Move()**  **X = X + V**  **If X > R.Len Then**  **X = 0**  **End If**  **End Sub**  **End Class** |

*Рис. 2.2-2. Программный код и схема класса* **TCAR**   
*с двумя конструкторами*

Как видим, наша объектная модель уровня реализации имеет некоторые отличия от модели концептуального и логического уровней (в классе **TCAR** появилось дополнительное поле **R** - ссылка на объект класса **TROAD** и конструкторы).

В событийной процедуре объявляем массив объектов класса **TCAR – МАШИН**, причем количество элементов этого массива **n** равно значению поля **ширина** **Width (количество полос)** уже созданного объекта **road** класса **TROAD (ДОРОГА)**.

|  |
| --- |
| **n = road.Width**  **Dim cars(n - 1) As TCAR** |

Это еще не объекты, а ссылки, т.е. переменные, в которые можно записать адреса объектов класса **TCAR**. Сами объекты создаются конструктором, при вызове которого задаются три параметра: адрес объекта **road** класса **TROAD,** номер полосы и скорость.

|  |
| --- |
| **For i = 0 To n - 1**  **cars(i) = New TCAR(road, i+1, 2 \* (i+1))**  **Next** |

В приведенном примере машина на полосе с номером **i+1** идет со скоростью **2(i+1)** единиц за один интервал моделирования (к индексу **i** прибавляется 1, т.к. элементы массива **cars(i)** нумеруются от нуля, а номера полос дороги считаются от единицы).

Сам цикл моделирования движения машин по дороге прост: несколько раз (например, 100 раз) на каждом шаге вызывается метод **Move()**  для каждой машины.

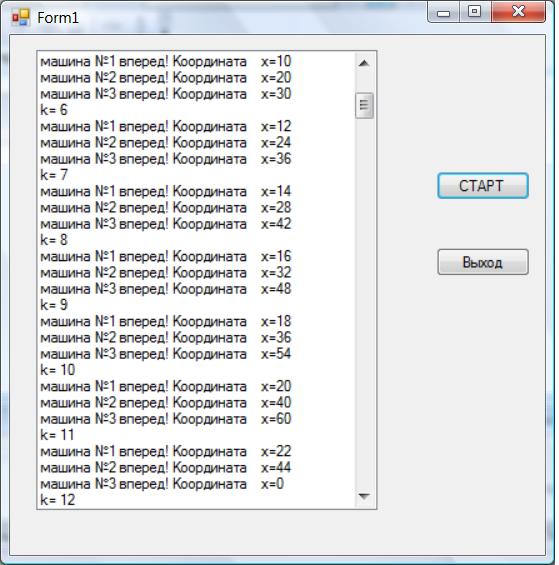
|  |
| --- |
| **For k As Integer = 1 To 100**  **For i = 0 To n - 1**  **cars(i).Move()**  **Next**  **Next** |

Полностью событийная процедура, отвечающая рассмотренной объектной модели, представлена на рис. 2.2-3.

|  |
| --- |
| **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button1.Click**  **Dim n, i, k As Integer**  **Dim s As String = ""** *'строка для записи в ListBox*  **Dim road As TROAD** *'объявление объекта road класса ROAD*  **road = New TROAD(60, 3)***'создание объекта road*  **n = road.Width** *'кол-во машин равно ширине дороги*  **Dim cars(n - 1) As TCAR***'объявление массива машин*  **For i = 0 To n – 1** *'создание объектов - машин*  **cars(i) = New TCAR(road, i + 1, 2 \* (i + 1))**  **Next**  **For k = 1 To 100** *'цикл моделирования движения*  **ListBox1.Items.Add("k= " + CStr(k))**  **For i = 0 To n - 1**  **cars(i).Move()**  **s = "машина №" + CStr(i + 1) + "вперед! Координата x="**  **s = s + CStr(cars(i).X)**  **ListBox1.Items.Add(s)**  **Next**  **Next**  **End Sub** |

*Рис. 2.2-3. Код событийной процедуры объектной модели* **Примера .2.2-1**

На форме имеется элемент управления **ListBox**, в который для иллюстрации выводится результат очередного шага цикла моделирования с указанием текущей координаты каждой машины (рис. 2.2-4).



*Рис. 2.2-4. Результаты работы программы* **Примера 2.2-1**

Итак, мы рассмотрели простейший проект, в котором в качестве членов классов использовали поля, методы и специальные методы – конструкторы. Повторим их главные особенности.

**Поля класса** синтаксически являются обычными переменными языка. Их описание удовлетворяет обычным правилам объявления переменных, а содержательно поля задают представление той самой абстракции данных, которую реализует класс. Поля определяют характеристикикласса**.** Напомним, что когда создается новый объект класса, то этот объект представляет собой набор полей класса. Два объекта одного класса **имеют один и тот же набор полей, но разные значения, хранимые в этих полях.** В рассмотренной выше задаче при создании конструктором объектов – **МАШИН**, их поля получали различные значения скорости и номера полосы. Или, например, все объекты класса **Person** могут иметь поле, характеризующее рост персоны, но один объект может быть высокого роста, другой – низкого, а третий – среднего роста.

**Методы класса** синтаксически являются обычными процедурами языка. Их описание удовлетворяет обычным правилам объявления процедур. Содержательно методы определяют ту самую абстракцию операций, которую реализует класс. Методы описывают операции, доступные объектам класса и определяют действия, выполняемые с экземпляром класса при обработке данных. Методы класса, точно так же, как и конструкторы, могут быть перегружаемыми (т.е. иметь одинаковые имена, но разную сигнатуру). Пример перегружаемых (**Overload**) методов рассмотрен в лабораторной работе (п.2.7) и далее в **Теме 3** (п.3.2, 3.6). Два объекта одного класса имеют один и тот же набор методов, точно так же, как и одинаковый набор полей.

**Конструкторы класса** синтаксически являются процедурами с именем **New** и используются для создания объектов (экземпляров) класса. Конструктор может применяться с присваиванием полю начального значения (инициализацией) или без него. При этом объект класса может создаваться одновременно с его объявлением, которое имеет вид:

|  |
| --- |
| **Dim** *ИмяПеременной* **As New** *ИмяКласса*  *'или*  **Dim** *ИмяПеременной* **= New** *ИмяКласса* |

Во втором способе (как показано в Примере 2.2-1 на рис.2.2-3 для объекта **road**) описание переменной не связано с созданием объекта и имеет вид:

|  |
| --- |
| **Dim** *ИмяПеременной* **As** *ИмяКласса*  *'например*  **Dim** **road** **As** **TROAD** |

А сам объект создается позже в процессе выполнения программы путем выполнения присваивания:

|  |
| --- |
| *ИмяПеременной* = **New** *ИмяКласса*  *'например*  **road** = **New** **TROAD** |

Таким образом, в приведенном примере процесс создания объекта **road** класса **TROAD** происходит в три этапа:

* Первым делом для объекта **road** создаётся ссылка, пока висячая со значением **Nothing**.
* Затем в динамической памяти создается объект – структура данных с полями, определяемыми классом **TROAD**. Поля объекта инициализируются значениями по умолчанию: ссылочные поля – значением **Nothing**, значимые – нулями. Эту работу выполняет конструктор по умолчанию (без параметров), если был вызван именно он. Если вызван конструктор с параметрами, то начинает выполняться тело этого конструктора. Как правило, при этом происходит инициализация отдельных полей класса значениями, переданными конструктору.
* На заключительном этапе ссылка связывается с созданным объектом.

Процесс создания объектов становится сложнее, когда речь идет об объектах, являющихся потомками некоторого класса. В этом случае, прежде чем создать сам объект, нужно вызвать конструктор, создающий родительский объект. Но об этом мы еще поговорим при изучении наследования.

У класса может быть несколько конструкторов – это типичная практика, – отличающихся сигнатурой (т.е. отличающихся количеством и типом параметров). Зачем классу нужно несколько конструкторов? Дело в том, что в зависимости от контекста и создаваемого объекта может требоваться различная инициализация его полей. Подчеркнем еще раз, правильный стиль требует, чтобы среди конструкторов класса **всегда** присутствовал конструктор по умолчанию – конструктор без параметров. Смысл этого требования станет ясен позже, при изучении наследования классов (Тема 3). Иногда тело этого конструктора по умолчанию пусто, поскольку все поля объявлены с инициализацией и конструктору нет смысла изменять эти значения.

И последнее. Если задача создания объектов с помощью конструкторов полностью возлагается на программиста, то задача удаления объектов, после того, как они стали ненужными, в VS .NET снята с программиста и возложена на соответствующий инструментарий VS – ***сборщик мусора***.

#### 2.2.2. Средства определения свойств классов

Рассмотрим теперь **Свойства (Properties)** класса. В языке VB принято, как и в других объектных языках, поля класса объявлять закрытыми, а нужную стратегию доступа к закрытым полям организовать через специальные средства. Для эффективности этого процесса и введены специальные члены класса – **свойства**. Во время построения объектной модели   
**ДОРОГА-МАШИНА** мы выделили отдельные объекты, которые для обмена данными друг с другом используют интерфейс между классами – внешние (открытые) поля и методы с модификатором доступа **Public**. Однако, все внутренние данные и детали внутреннего устройства объекта должны быть скрыты от «внешнего мира» (модификатором доступа **Private**). Скрытие внутреннего устройства объектов называется **инкапсуляцией** и является одним из основных принципов ООП. Такой подход позволяет обезопасить внутренние данные (поля) объекта от изменений (возможно, разрушительных) со стороны других объектов; проверять данные, поступающие от других объектов на корректность, повышая тем самым надежность программного кода; передавать внутреннюю структуру и код объекта любым способом, не меняя его внешние характеристики (интерфейс), и при этом никакой переделки других объектов не требуется.

Итак, поля классов рекомендуется определять, как закрытые, а для обеспечения достаточно полного интерфейса класса вводить нужное количество открытых методов. Объектно-ориентированный подход к программированию традиционно рекомендует вводить в класс специальные средства, позволяющие получать значения закрытых полей, и позволяющие желаемым способом задавать их значения. **Свойства** (**Property)** представляют собой специальную синтаксическую конструкцию, предназначенную для организации ограниченного доступа к полям класса. Программный код такого доступа (т.е. программный код свойства) представляет собой блоки операторов, которые выполняются при получении (**Get**) значения закрытого поля или установке и изменения (**Set**) этого значения. Если отсутствует часть **Set**, свойство доступно только для чтения (**ReadOnly**), если отсутствует часть **Get**, свойство доступно только для записи (**WriteOnly**), т.е. может отсутствовать либо часть **Set**, либо часть **Get**, но не обе части одновременно.

**Пример 2.2-2. Изменить в Примере 2.2-1 программный код классов** **TROAD и TCAR**, **объявив поля классов закрытыми (Private) и ввести в классы свойства для работы с закрытыми полями**.

Имена закрытых полей будем начинать с буквы **F** (от англ. Field – поле), а имена **Свойств** для работы с этими полями будут совпадать с именем поля без начальной буквы **F**. На рис. 2.2-5 приведен код измененного класса **TROAD**.

|  |  |
| --- | --- |
| *'Oписание класса ДОРОГА*  **Public Class TROAD**  *'Закрытые поля(данные) класса*  **Private FLen As Double** *'длина дороги*  **Private FWidth As Integer** *'ширина(кол-во полос)*  *' Конструктор с параметрами*  **Public Sub New(ByVal Len0 As Double, ByVal W0 As Integer)**  **Len = Len0 : Width = W0**  **End Sub**    **Public Sub New()**  *'Пустой конструктор по умолчанию*  **End Sub**  *'св-во для чтения и записи значения длины дороги*  **Public Property Len() As Double**  **Get** *'часть для чтения*  **Return FLen**  **End Get**  **Set(ByVal value As Double)** *'часть для записи*  **If value > 0 Then**  **FLen = value**  **Else**  **MsgBox("Ошибка!Длина дороги больше нуля")**  **FLen = 100**   |  | | --- | |  |   **End If**  **End Set**  **End Property**    *'св-во для чтения и записи FWidth*  **Public Property Width() As Integer**  **Get**  **Return FWidth**  **End Get**  **Set(ByVal value As Integer)**  **If value > 0 Then**  **FWidth = value**  **Else**  **MsgBox("Ошибка!Число полос дороги больше нуля")**  **FWidth = 1**  **End If**  **End Set**  **End Property**  **End Class** |

*Рис. 2.2-5 Программный код и схема класса* **TROAD**   
 *с закрытыми полями и свойствами*

Итак, в классе **TROAD** имеются два свойства: **Len()** и **Width()**. Каждое из них состоит из двух частей: часть **Get** – функция для получения значения закрытого поля, куда мы пишем код, показывающий значение поля, и часть **Set** – процедура всегда с параметром **value** (тип которого совпадает с типом поля, для работы с которым предназначено свойство), куда записывается код, позволяющий изменять значение поля извне. При этом в нашем случае здесь проводится проверка на допустимость вводимого значения поля и выводится сообщение об ошибке в случае попытки ввести некорректное значение и поэтому изменен код конструктора с параметрами.

При рассмотрении объекта **МАШИНА** учтем, что как правило, другие объекты не могут непосредственно менять скорость и местоположение **МАШИНЫ**, но могут получить информацию о ней – узнать значения этих полей (скорости и местоположения). Поэтому в классе **TCAR** дляработы с закрытыми теперь полями класса введены свойства «**только для чтения**», а изменения значений закрытых полей производятся только методами **New** и **Move()** самого класса, код которых немного исправлен, чтобы обеспечить эти изменения. Код измененного класса **TCAR** приведен на рис. 2.2-6.

|  |
| --- |
| *'Oписание класса МАШИНА*  **Public Class TCAR**  *'Закрытые поля(данные) класса*  **Private FX As Double** *'координата машины на дороге*  **Private FY As Integer** *'номер полосы,по к-рой едет машина*  **Private FV As Double** *'скорость*  *'ссылка на конкретную дорогу(уже созданный объект класса TRoad)-*  *'это доп. поле для программирования метода Move()*  **Private R As TROAD**  *'===================================*  *' Конструктор с параметрами*  **Public Sub New(ByVal R0 As TROAD, \_**  **ByVal y0 As Integer, ByVal v0 As Double)**  **R = R0 : FY = y0 : FV = v0**  **End Sub**    **Public Sub New()**  *' Пустой конструктор по умолчанию*  **End Sub**  *'свойства закрытых полей класса (только для чтения)*  *'т.к. значения полей не должны изменяться снаружи,*  *'а изменяются только методами внутри класса*  *'св-во для чтения значения координаты машины*  **Public ReadOnly Property X() As Double**  **Get**  **Return FX**  **End Get**  **End Property**  *'св-во для чтения значения полосы,*  *'по которой едет машина*  **Public ReadOnly Property Y() As Integer**  **Get**  **Return FY**  **End Get**  **End Property**  *'св-во для чтения знач. скорости машины*  **Public ReadOnly Property V() As Double**  **Get**  **Return FV**  **End Get**  **End Property**  *'===================================*  *'Открытый метод класса*  **Public Sub Move()**  **FX = FX + FV**  **If FX > R.Len Then FX = 0**  **End Sub**  **End Class** |

*Рис. 2.2-6. Измененный программный код и схема класса* **TCAR**

В событийной процедуре, для иллюстрации работы свойств, воспользуемся при создании объекта **road** класса **TROAD** конструктором без параметров, и введем значения закрытых полей через свойства, используя локальные переменные **w** и **n**.

Программный код представлен на рис. 2.2-7.

|  |
| --- |
| **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button1.Click**  **Dim n, i, k As Integer**  **Dim s As String = ""**  **Dim road As TROAD** *'объявление объекта road класса TROAD*  **road = New TROAD** *'создание объекта конструктором без пар-ров*  **Dim w As Double** *'переменная для ввода длины дороги*  **w = CDbl(InputBox("введите длину дороги"))**  **road.Len = w** *'ввод значения закрытого поля через св-во*  **n = CInt(InputBox("введите число полос дороги"))**  **road.Width = n** *'ввод значения закрытого поля через св-во*  **Dim cars(n - 1) As TCAR** *' объявление массива машин*  **For i = 0 To n – 1** *'создание объектов - машин*  **cars(i) = New TCAR(road, i + 1, 2 \* (i + 1))**  **Next**  *'цикл моделирования*  **For k = 1 To 100**  **ListBox1.Items.Add("k= " + CStr(k))**  **For i = 0 To n - 1**  **cars(i).Move()**  **s = "машина №" + CStr(i + 1) + "вперед! Координата x="**  *'узнаем значение закрытого поля через св-во*  **s = s + CStr(cars(i).X)**  **ListBox1.Items.Add(s)**  **Next**  **Next**  **End Sub** |

*Рис. 2.2-7. Программный код событийной процедуры* **Примера 2.2-2**

Итак, **Свойство** – это член класса, который обеспечивает доступ к внутреннему состоянию объекта (закрытым полям класса). Как видно из программного кода на рис. 2.2-7, с точки зрения внешнего пользователя, свойства синтаксически не отличаются от полей класса. Однако между свойствами и полями есть принципиальное различие – в объекте отсутствует ассоциированный со свойством участок памяти. Такой участок памяти выделяется для каждого поля класса, а свойство управляет доступом к полю (получает либо задает его значение), но само не создает это поле, и поле должно быть описано независимо от свойства. Для обращения к свойству используется его имя с применением точечной нотации. То есть надо записать имя объекта, а затем имя свойства. Например, **road.Len = w*.***

Здесь для присваивания значения закрытому полю **FLen** мы обращаемся к свойству **Len**, вернее, его части **Set** через параметр **value**, который всегда имеет тот же тип, что и тип поля. Для того, чтобы узнать значение закрытого поля также используется обращение к свойству (части **Get**), которая возвращает значение того же типа, что и тип поля (и тип свойства).

Например, **w = road.Len** или, как в примере на рис 2.2-7:

|  |
| --- |
| *'узнаем значение закрытого поля FX через св-во X*  **s = s + CStr(cars(i).X)** |

Итак, как правило, свойство связано с закрытым полем класса. Свойства позволяют выполнить вызов функции, синтаксически похожий на присваивание значения переменной, или же чтение значения переменной. При работе со свойствами объекта часто необходимо выбрать нужную стратегию доступа к полю класса. Перечислим пять наиболее употребительных стратегий:

* чтение, запись (**Read, Write**);
* чтение, запись при первом обращении (**Read, Write-once**);
* только чтение (**ReadOnly**);
* только запись (**WriteOnly**);

**Пример 2.2-3**. **Рассмотрим класс Person, у которого пять полей: fam, status, salary, age, health, характеризующих фамилию, статус, зарплату, возраст и здоровье персоны. Для каждого из этих полей может быть разумной своя стратегия доступа. Возраст доступен для чтения и записи, фамилию можно задать только один раз, статус можно только читать, зарплата недоступна для чтения, а здоровье закрыто для доступа, только специальные методы класса могут сообщать некоторую информацию о здоровье персоны.**

В этом примере (рис. 2.2-8) показано, как на VB можно обеспечить эти стратегии доступа к закрытым полям класса, при этом в классе используется перечисление **Enum**, которое представляет собой набор связанных констант, т.е. изменять их значения нельзя, а для доступа к конкретной константе используется точечная нотация:

|  |
| --- |
| *'Класс, задающий общие свойства и поведение личности*  **Public Class Person**  **Public Enum Status1** *'перечисление всех возможных значений*  **ребенок** *'объекта Status1*  **школьник**  **студент**  **работник**  **пенсионер**  **End Enum**  *'поля класса(все закрыты)*  **Dim fam As String = ""** *'фамилия*  **Dim health As String = ""** *'здоровье*  **Dim age As Integer = 0** *'возраст*  **Dim salary As Integer = 0** *'зарплата*  **Dim status As Status1 = Status1.работник** *'статус*  *'свойство со стратегией Write-once*  **Public Property Fam1() As String**  **Set(ByVal value As String)**  **If (fam = "") Then fam = value**  **End Set**  **Get**  **Return fam**  **End Get**  **End Property**  *'стратегия: ReadOnly(Только чтение)*  **Public ReadOnly Property GetStatus() As Status1**  **Get**  **Return status**  **End Get**  **End Property**  *'стратегия: Read,Write (Чтение, запись)*  **Public Property Age1() As Integer**  **Set(ByVal value As Integer)**  **age = value**  **If (age < 7) Then**  **status = "ребенок"**  **ElseIf (age < 17) Then**  **status = "школьник"**  **ElseIf (age < 22) Then**  **status = "студент"**  **ElseIf (age < 60) Then**  **status = "работник"**  **Else**  **status = "пенсионер"**  **End If**  **End Set**  **Get**  **Return age**  **End Get**  **End Property**  *'стратегия: WriteOnly (Только запись)*  **Public WriteOnly Property Salary1() As Integer**  **Set(ByVal value As Integer)**  **salary = value**  **End Set**  **End Property**  **End Class** |

*Рис. 2.2-8 Программный код класса* ***Person*** *с различной стратегией свойств*

Ещё раз подчеркнём важность свойств класса. Часто возникает вопрос. Нельзя ли просто создать общие перемен­ные, значения которых программист может непосредственно получать и изменять? Ответ содер­жится в одном из важнейших понятий технологии ООП – инкапсуляции данных, которая означает, что внешние объекты не имеют права прямого доступа к внутренним переменным класса. При работе с такими переменными клиент должен использовать только четко определенные свойства, предоставляемые ему экземпляром класса. Преимущества этого способа представления данных состоят в следующем:

* предотвращается несанкционированный доступ к данным;
* гарантируется целостность и корректность данных путем контроля ошибок;
* создаются свойства, доступные только для чтения и только для записи;
* пользователь лишается возможности вносить изменения в выполняемую программу.

#### 2.2.3. Области видимости членов класса. Статические и динамические члены класса.

Каждый член класса, а также сам класс могут иметь так называемые модификаторы доступа, которые указывают область их видимости. Значения модификаторов могут быть следующими:

* **Public** – открытый класс или член класса, доступ к которому разрешен из любого места программного кода;
* **Private** – класс или член класса, доступный только из контекста, в котором он объявлен, и во всех вложенных контекстах; т.е. если, например, свойство объявлено с модификатором **Private**, то оно доступно только из того же самого класса и из вложенных в него;
* **Friend** – класс или член класса, доступный только внутри той же сборки, в которой объявлен (в VB сборка обычно соответствует всей программе, поэтому данный модификатор можно воспринимать как указание видимости только в пределах программы);
* **Protected** – член, доступный только из самого класса и из наследующих клас­сов (данный модификатор применим только к членам классов, но не к самому классу);
* **Protected Friend** – объединение областей видимости **Protected** и **Friend.**

Обычно различают области видимости четырёх уровней (в порядке увеличения охвата):

* блок внутри процедуры или функции;
* процедура или функция;
* модуль;
* проект.

Длясистематизации знаний об областях видимости переменных, констант и процедур внутри программного модуля и из других модулей рассмотрим пример.

**Пример 2.2-4. Анализ области видимости.**

Рассмотрим программный код модуля класса (рис. 2.2-9).

|  |
| --- |
| **Public Class Class1**  **Dim C As Integer**  **Public A As Integer**  **Private В As Integer**  **Const M As Integer = 3**  **Public Const К As Integer = 1**  **Private Const L As Integer = 2**  **Private Sub Проц()**  **Dim C1 As Integer**  **End Sub**  **Sub Проц1( )**  **Const L1 As Integer = 4**  **End Sub**  **Public Function Функц() As String**  **Dim D As Integer**  **Return "Привет"**  **End Function**    **End Class** |

*Рис. 2.2-9. Программный код класса   
с членами различной области видимости*

Рассмотрим области видимости переменных, констант и процедур модуля класса, начиная с самой узкой и кончая самой широкой**:**

* Локальные переменные и константы.

Внутри процедур и функций переменные могут быть объявлены только при помощи слова **Dim**, а константы – только при помощи **Const**. Это блочная и локальная области видимости и знакомые локальные переменные и константы. Они видны только в той процедуре, функции или блоке, в которых объявлены. В примере это **С1**,**L1** и **D**.

Вне процедур и функций могут быть объявлены переменные, константы, процедуры и модули. Они могут быть видны или только в том модуле, где они объявлены, или во всем проекте, или в неограниченной области в зависимости от того, с каким модификатором было сделано объявление.

* Область видимости – модуль.

Слова **Dim** или **Private** для переменной и слова **Const** или **Private Const** для константы делают их видимыми только в своем модуле. Их область видимости - весь этот модуль, включая все процедуры, функции. Это модульные переменные и константы – **C, B, M, L**.

Таким образом, для объявления модульных переменных и констант можно обойтись и без модификатора **Private** (достаточно оператора **Dim**). Любой другой элемент (процедура, модуль), чтобы быть видимым аналогичным образом только в своем модуле, объявляется с модификатором **Private** (процедура **Проц()**).

* Область видимости – проект.

Если необходимо, чтобы элемент VB был виден во всем проекте, но не дальше, объявляем его модификатором **Friend.** Часто употребляют вместо **Friend** более привычный модификатор **Public**

* Видимость по умолчанию.

Если совсем убрать модификатор доступа к классам и модулям, то будет иметь доступ **Friend** или **Public**.

Если совсем убрать модификатор доступа к процедуре, то она будет иметь доступ **Public**.

Переменные и другие программные элементы разных модулей или разных процедур могут иметь одинаковые имена. Как в этом случае VB определяет, какой из ви­димых одноименных элементов имеется в виду в каждом конкретном случае? Здесь вступает в действие эффект ***затенения (экранирования)*:** из нескольких одноименных элементов всегда имеется в виду более локальный, т.е. тот, чья зона видимости меньше. Это значит, что локальные элементы имеют предпочтение перед модульными элементами, а модульные элементы – перед глобальными. Мы также будем рассматривать эффект экранирования при изучении наследования в **Теме 3** (п.3.2).

Вернемся к классам и членам класса. Независимо от модификатора доступа все члены класса доступны для всех остальных членов данного класса. Поля класса доступны только для методов класса, если они имеют модификатор доступа **Private,** который можно опускать. Такие поля считаются ***закрытыми***.

Понятно, что класс, у которого все методы закрыты, абсурден, поскольку никто не смог бы вызвать ни один из его методов. Как правило, у класса есть ***открытые методы***, задающие ***интерфейс класса*** (не путать с интерфейсами **Interface**, которые будем изучать в **Теме 3**), и ***закрытые методы***. ***Интерфейс класса*** – это лицо класса, именно он определяет, чем класс интересен своим клиентам, что он может делать, какие сервисы предоставляет клиентам. Закрытые методы составляют важную часть класса, позволяя клиентам класса не вникать во многие детали реализации класса. Эти методы клиентам класса недоступны, они о них могут ничего не знать, и, самое главное, изменения в закрытых методах класса никак не отражаются на клиентах класса при условии корректной работы открытых методов класса.

О ***защищённых*** членах класса (с модификатором **Protected**) мы будем подробно говорить при рассмотрении наследования классов в **Теме3**.

Наряду с модификаторами доступа, регламентирующими видимость, члены класса могут содержать ***модификаторы, устанавливающие их принадлежность к классу или к экземпляру класса***. Члены класса, принадлежащие всему классу, называют ***разделяемыми* (Shared)** или ***статическими*** членами. Разделяемыми (статическими) членами класса являются свойства, поля и методы, которые разделяются всеми объектами класса, то есть являются общими для всех объектов этого класса. При объявлении такого члена класса перед его именем указывается модификатор **Shared**. Члены класса, которые объявлены без этого модификатора, называются ***экземплярными* (Instance)** или ***динамическими*** и принадлежат конкретному экземпляру класса, а не всему классу. Чтобы понять разницу между статическими и экземлярными методами и полями, необходимо более подробно рассмотреть механизм вызова методов и обращений к полям.

Объявление класса фактически задает последовательность расположения полей в памяти и способы вызова методов класса. При создании экземпляра класса (конкретного объекта) происходит выделение памяти согласно структуре полей класса. При вызове экземплярного метода в качестве неявного параметра ему передается информация об экземпляре класса, для которого вызван этот метод (в VBэта неявная ссылка обозначается ключевым словом **Me**, а ссылка на класс обозначается словом **MyClass**). При вызове статического метода такой информации не передается, по­этому статический метод может быть вызван и при отсутствии какого-либо экземпляра класса. То есть при вызове статической процедуры не нужно создавать объект (экземпляр) класса. Отсюда сразу вытекает ограничение, накладываемое на статические методы. Статический метод не может обращаться к нестатическим методам и полям своего класса без указания конкретного экземпляра.

При обращении к полю экземпляра класса доступ производится по смещению относительно начала положения экземпляра класса в памяти. В отличие от экземплярного поля, статическое поле не требует конкретного объекта, поскольку оно создается и инициализируется всего один раз. Статические поля аналогичны глобальным переменным.

Напомним, что классы библиотеки .NET Frameworkиспользовались двояко: без создания объектов (статические) и с созданием объекта (экземплярные). В первом случае, т.е. не создавая объект класса, чтобы воспользоваться возможностями класса, необходимо было написать имя класса, точку и имя необходимого статического свойства или метода этого класса. Так использовались математические функции класса **System.Math**. Во втором случае, чтобы вызвать экземлярный метод класса, необходимо сначала создать объект – экземпляр класса, а уж затем перед именем необходимого экземлярного метода писать имя объекта класса. Так использовались потоковый ввод/вывод и классы **StringBuilder** и **Array [1]**.

**Пример 2.2-5. Описать класс, члены которого имеют различные модификаторы доступа и принадлежности.**

Простой класс, иллюстрирующий работу со статическими и экземплярными методами, а также полями различной области видимости приведен на рис. 2.2-10.

|  |
| --- |
| **Public Class SampleBase**  *'набор полей с различными модификаторами*  **Private privateVar As Integer**  **Public publicVar As Integer**  **Friend friendVar As Integer**  **Protected protVar As Integer**  *'Открытый экземплярный метод класса*  **Public Sub SomeMethod()**  *'Внутри класса доступны все его пол****я***  **privateVar = 5**  **End Sub**  *'Закрытый экземплярный метод класса*  **Private Function SomeOtherMethod() As Integer**  *'Открытые поля доступны всегда*  **Return publicVar**  **End Function**  *'Открытое экземплярное свойство*  **Public Property SomeProperty( ) As Integer**  **Get**  **Return privateVar**  **End Get**  **Set(ByVal Value As Integer)**  **privateVar = Value**  **End Set**  **End Property**  *'Открытый статический метод класса*  **Public Shared Sub SharedMethod()**  **…**  **End Sub**  **End Class**  **. . .**  *'Точка входа в программу*  **Private Sub Button1\_Click( ... )**  **Dim a As New SampleBase**  **a.friendVar = 5** *'Допустимо: обращение из*  *'той же программы*  **a.publicVar = 6** *'Допустимо: поле открыто*  **a.privateVar = 6** *'Ошибка:поле класса закрыто*  **а.SomeProperty = 5**  **SampleBase.SharedMethod()***'обращение к статическому методу:*  *'вместо имени объекта указывается имя класса*  **End Sub**  **. . .** |

*Рис. 2.2-10. Программный код класса   
с полями и методами различных модификаторов доступа*

В данном примере описывается класс **SampleBase**. В классе **SampleBase** объявлен набор полей, методов и одно свой­ство. При попытке откомпилировать этот код будут выдано сообщение:

|  |
| --- |
| **'sampleVBProject.SampleBase.privateVar' is not accessible in this context because it is 'Private'.** |

Это показывает, что закрытые (**Private**) члены не доступны вне контекста класса. Обратите внимание на имя переменной в сообщении об ошибке:

|  |
| --- |
| **sampleVBProject.SampleBase.privateVar** |

Оно представляет собой полное имя члена класса, которое состоит из пространства имен (**sampleVBProject**), имени класса (**SampleBase),** имени члена **(privateVar).**

### 2.3. Две роли классов в ООП и типы данных

ЯзыкпрограммированияVBявляется объектно-ориентированным языком. Это означает, что все функциональные части приложений рассматриваются как классы или объекты, содер­жащие в себе некоторые характеристики, способные выполнять определенные операции и генерировать события. На первый взгляд классы ООП и связанные с ними характеристики и операции имеют много общего с процедурным подходом и модульным строением программ. Ключевое различие заключается в следующем: *класс представляет собой шаблон для создания объектов, состояние которых изменяется со временем*.

При изучении базовых средств VB, так или иначе мы сталкивались с классами. В первую очередь это были классы, предоставленные пользователю VS .NETв виде библиотек классов .NET Framework**.** Например, класс математических функций – **System.Math**, классы **System.Array**, **System.IO,** форма и элементы управления– **System.Form** и другие.

Любую программную систему, построенную в объектном стиле, можно рассматривать как совокупность классов, объединенных в проекты и решения. Очень важно обратить внимание на то, что у класса две различные роли: ***модуля*** и ***типа данных***.

***Во-первы****х*, **класс – это модуль**, **архитектурная единица** построения программной системы. ***Модульность построения*** – основное свойство программных систем. В **ООП** программная система, строящаяся по модульному принципу, состоит из классов, являющихся основным видом модуля. Модуль может не представлять собой содержательную единицу, его размер и содержание определяется архитектурными соображениями, а не семантическими. Ничто не мешает построить монолитную систему, состоящую из одного модуля (модуля пользователя), она может решать ту же задачу, что и система, состоящая из многих модулей. Особенности построения и использования в VB стандартного модуля (модуля пользователя) были рассмотрены в **Теме 1** и еще будут рассмотрены в примерах п. 2.4

***Вторая роль класса*** не менее важна. **Класс – это тип данных**, задающий реализацию некоторой абстракции данных, характерной для задачи, в интересах которой создается программная система. Как мы уже рассмотрели в примерах п. 2.2, каждый объект в VB определяется с помощью класса. Объекты представляют собой экземпляры классов; после того как класс определен, можно создать любое количество объектов этого класса (т.е. этого пользовательского типа). Таким образом, в этом случае класс – это шаблон, который определяет форму объекта. Он задает как данные (поля), так и программный код процедур (методов), которые оперируют этими данными. Здесь важно понимать, что класс – это логическая абстракция. В памяти не существует физического представления класса (за исключением статических классов) до тех пор, пока не создан объект этого класса. То есть, класс является ссылочным пользовательским типом данных.

Для лучшего понимания рассматриваемого ранее и в следующих Темах 3 и 4 материала, напомним, что типы данных *по способу хранения элементов* разделяются на два вида: на ***типы – значения*** и ***ссылочные типы***. Элементы **типов-значений** или ***значимых типов*** представляют собой просто последовательности битов в памяти, которую выделяет компилятор в стеке, т.е. переменные значимых типов хранят свои значения непосредственно и представляют в программе конкретные данные. Величина **ссылочного типа** хранит не сами данные, а ссылку (указатель) на них (адрес, по которому они расположены), причем сами данные хранятся не в стеке, а в динамической области памяти, называемой «кучей».

Основное ***отличие типов значений от типов-ссылок*** состоит в следующем. Каждой переменной значимого типа принадлежит её собственная копия данных, и поэтому операции с одной переменной не влияют на значения других переменных. Несколько переменных ссылочного типа могут быть одновременно соотнесены с одним и тем же объектом. Поэтому операции, выполняемые с одной из этих переменных, могут изменять объект, на который в этот момент ссылаются другие переменные (с типом ссылок). Величины значимого типа равны, если равны их значения, а величины ссылочного типа равны, если они ссылаются на одни и те же данные. На рис. 2.3-1 b и c равны, но a не равно b даже при одинаковых значениях.

|  |
| --- |
| Куча  значение  значение |
| Стек  значение  ссылка  ссылка  ссылка  значение  xy ab c |

*Рис. 2.3-1. Хранение в памяти величин значимого и ссылочного типов*

К ***типам – значениям*** относятся:

* все числовые типы данных;
* типы **Boolean**, **Char** и **Date**;
* все структуры, даже если их члены являются ссылочными типами;
* перечисления, поскольку их базовый тип всегда является **SByte, Short, Integer, Long, Byte, UShort, UInteger** или **ULong**.

К ***ссылочным*** типам относятся:

* тип **String**;
* все массивы, даже если их члены являются типами значений;
* типы классов;
* делегаты (будут рассмотрены в Теме 4).

Итак, объектно-ориентированная разработка программной системы основана на стиле, называемом ***проектированием от данных***, которое сводится к поиску подходящих для данной задачи абстракций данных. Каждая из них реализуется в виде класса, которые и становятся модулями – архитектурными единицами построения системы и, в то же время, в основе класса лежит абстрактный тип данных.

В хорошо спроектированной объектно-ориентированной системе каждый класс играет обе роли, так что каждый модуль проектируемой программной системы имеет вполне определенную смысловую нагрузку. Типичная ошибка – рассматривать класс только как архитектурную единицу, объединяя в одном классе разнородные поля и функции, после чего становится неясным, какой же тип данных задает этот класс. Напомним, что для этой цели в VB может использоваться ***модуль пользователя (стандартный модуль)***.

Класс можно рассматривать как «интеллектуальный» пользовательский тип данных, обладающий расширенными возможностями. Например, перед изменением своих внутренних данных класс может проверить полномочия стороны, от которой поступил вызов. С этой точки зрения объект пред­ставляет собой переменную пользовательского типа, с которой ассоциированы функции обраще­ния к данным и их проверка. Главный принцип работы с данными класса – замена прямого досту­па к данным вызовом вспомогательных процедур.

### 2.4. Взаимодействие, взаимное различие и сходство форм, модулей и классов

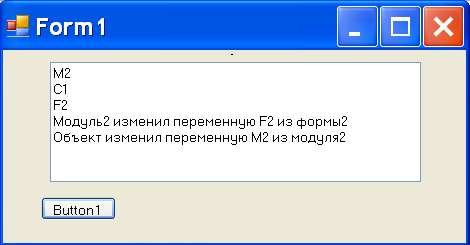
Рассмотрим взаимодействие, взаимное различие и сходство форм, модулей и классов на примере.

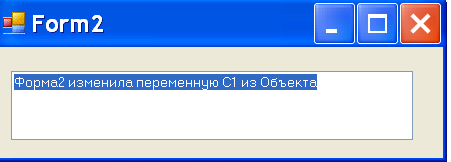
**Пример 2.4-1. Создать проект из двух форм, двух модулей и класса.**

Рассмотрим программный код проекта, состоящего из двух стандартных модулей, двух форм и одного класса, представленный на рис. 2.4-1, а также результат его работы на рис. 2.4-2.

|  |
| --- |
| *'Первый стандартный модуль объявляет (и создает конструктором!)*  *'общедоступные переменные (они будут «видны» из любого места*  *'проекта)* – *объекты типа Form2 и Класс*  **Module Module1**  **Public Форма2 As New Form2**  **Public Объект As New Класс**  **End Module**  *'Второй стандартный модуль объявляет общедоступную строковую*  *'переменную М2 и процедуру, которая обращается к объявленному*  *'и созданному в первом модуле объекту второй формы Форма2*  **Module Module2**  **Public M2 As String = "M2"**  **Public Sub Процедура\_модуля2()**  **Форма2.F2 = "Модуль 2 изменил переменную F2 из формы 2"**  **End Sub**  **End Module**  *'Класс объявляет общедоступную строковую переменную – поле С1*  *'и процедуру, которая обращается к строке М2 из второго модуля*  **Public Class Класс**  **Public C1 As String = "C1"**  **Public Sub Процедура\_класса( )**  **M2 = "Объект изменил переменную M2 из модуля M2"**  **End Sub**  **End Class**  *'Во второй форме объявляется общедоступная строковая перем. F2*  *'и процедура, которая обращается к объявленному*  *'и созданному в первом модуле объекту Объект класса Класс,*  *'изменяет значение его поля С1 и выводит его в свой элемент*  *'управления - текстовое поле TextBox1*  **Public Class Form2**  **Public F2 As String = "F2"**  **Public Sub Процедура\_формы2()**  **Объект.C1 = \_**  **"Форма2 изменила переменную С1 из Объекта" & vbNewLine**  **TextBox1.Text = Объект.C1**  **End Sub**  **End Class**  *'В событийной процедуре первой формы иллюстрируется обращение*  *'к объектам стандартного модуля (М2), класса (С1) и второй*  *'формы (F2), значения которых «склеиваются» в строковую*  *'переменную SS до и после вызова процедур, которая выводится*  *' в текстовое поле TextBox1*  **Public Class Form1**  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button1.Click**  **Dim SS As String = ""**  **SS = SS & M2 & vbNewLine : SS = SS & Объект.C1 & vbNewLine**  **SS = SS & Форма2.F2 & vbNewLine : Процедура\_модуля2( )**  **SS = SS & Форма2.F2 & vbNewLine**  **Объект.Процедура\_класса()**  **SS = SS & M2 & vbNewLine**  **TextBox1.Text = SS** *'вывод*  **Форма2.Процедура\_формы2()***'вызов процедуры из 2-й формы*  **Форма2.Show()***'показать 2-ю форму*  **End Sub**  **End Class** |

*Рис. 2.4-1*. *Программный код проекта из двух стандартных модулей,   
двух форм и класса*





*Рис. 2.4-2 Результаты работы проекта из двух модулей,   
двух форм и класса*

Рассмотрим сходство и различия трех модулей, а именно **стандартного модуля** **Module2**, **модуля формы Form2** и **модуля класса Класс**. Очевидно, что их программный код почти одинаков. Код всех этих трех модулей содержит объявление переменной и объявление процедуры. И переменная, и процедура объявлены как **Public**, чтобы и той, и другой можно было пользоваться из любого модуляпроекта.

Когда форма, модуль и класс могут начать работу?

Если форма – **стартовый объект**, VB сам, без всякого вмешательства загружает ее и отображает на экране.

Стандартные модули тоже безо всяких забот программиста готовы к работе и использованию сразу после запуска проекта.

Это значит, что форма **Form1**и оба модуля готовы к работе и использованию сразу после запуска проекта.

Что же касается класса и нестартовой формы, то для того, чтобы они пришли в рабочее состояние, надо создать их экземпляры (объекты). Создание этих объектов с помощью конструктора по умолчанию **New** происходит в первом стандартном модуле. Напомним, если необходимо использовать нестатические (экземплярные) поля и методы класса (а в нашем классе **Класс** поле и метод именно такие, т.к. они объявлены без модификатора **Shared**) то, чтобы класс заработал, нам нужно из него создать объект. Что и делает следующий оператор:

|  |
| --- |
| **Public Объект As New Класс** |

Форма тоже класс, только он может визуально отображаться на экране. А раз так, то, чтобы она заработала в проекте, из нее тоже должен быть создан объект и показан на экране.

Если форма - стартовый объект, эти заботы берет на себя VB.

Если форма - нестартовый объект, программист должен сам создать ее как объект из ее класса, а затем показать на экране. Это можно реализовать с помощью двух операторов: оператора, который создает форму, как объект класса **Form2** (в нашем проекте этот оператор записан в первом стандартном модуле):

|  |
| --- |
| **Public Форма2 As New Form2** |

и оператора, показывающего форму:

|  |
| --- |
| **Форма2.Show()** |

Строки стандартного модуля, объявляющие переменные, выполняются автоматически.

Поэтому объекты **Объект** и **Форма2*,*** объявленные в модуле **Module1**, будут созданы сразу же в начале работы проекта, так как стандартный модуль, в отличие от класса и формы (если она не стартовая, конечно) готов к работе сразу после запуска проекта.

Когда программист в режиме проектирования работает над формой **Form2**, добавляет в нее элементы управления, пишет код, он создает и видит перед собой класс (не объект) **Form2**. Когда же запускается проект и выполняется оператор:

|  |
| --- |
| **Public Форма2 As New Form2** |

то из этого класса порождается объект **Форма2**, как экземпляр класса **Form2**. Форма (объект, а не класс) порождена, и можно пользоваться ее переменными и процедурами. Но на экране она не видна. Чтобы ее увидеть и пользоваться ее элементами управления, необходимо выполнить оператор:

|  |
| --- |
| **Форма2.Show()** |

Аналогично, когда в режиме проектирования пишется код модуля класса, то создается класс, а когда запускается проект и выполняется оператор:

|  |
| --- |
| **Public Объект As New Класс** |

то порождается из этого класса объект **Объект** как экземпляр класса **Клас**с.

В чем же отличие **формы** от **класса**? В том, что создатели **VB** для удобства разработчиков снабдили формы механизмом визуализации. Это значит, что как в ре­жиме проектирования, так и в режиме работы форма на экране видима. Специальные, скрытые от программиста методы рисуют и перерисовывают как форму, так и все ее эле­менты управления.

Класс же лишен механизма визуализации. Поэтому объекты, порожденные из него, невидимы. У них не может быть ни кнопок, ни меток, ни других элементов управления.

Еще разнапомним ***правила обращения к переменным и процедурам***. Ниже под модулем будем по­нимать или форму, или класс, или стандартный модуль.

Если переменная или процедура объявлены с помощью слова **Public**, то они видны и доступны не только из своего модуля, но и из других модулей проекта.

Грамматически обращение к переменным и процедурам разных модулей происходит следующим образом:

* для обращения к переменной или процедуре, объявленной в стандартном модуле, дос­таточно написать имя переменной или процедуры без указания имени модуля;
* для обращения к переменной или процедуре, объявленной в классе, нужно перед их именем написать имя объекта этого класса с точкой, а если переменная или процедура являются статическими (разделяемыми) членами класса, – то перед их именем пишется имя класса с точкой;
* для обращения к переменной или процедуре, объявленной в форме, нужно перед их именем писать имя экземпляра класса этой формы с точкой.

Перед именем переменной или процедуры, объявленной в стандартном модуле, имя модуля писать не запрещено. Это бывает даже необходимо, когда в нескольких стан­дартных модулях объявлены переменные или процедуры с одинаковыми именами. Ина­че их не различишь.

Все вышесказанное касается случая, когда необходимо обратиться к переменной или про­цедуре, объявленной в чужом модуле. Когда же нужно обратиться к переменной или процедуре, объявленной в своем модуле, все просто: пишется имя перемен­ной или процедуры. Иногда, правда, бывает удобно или даже необходимо указать имя хозяина или **Me**. Это не возбраняется (правда, в случае стандартного модуля слово **Me** писать нельзя).

**Пример 2.4-2. Способы обращения к собственной перемен­ной.**

Рассмотрим код, представленный на рис. 2.4-3, показывающий разные способы обращения к собственной перемен­ной из формы, стандартного модуля и из класса, дающие один и тот же результат (код стандартного модуля **Module1** такой же, как в **Примере 2.4-1**, и в нем создаются **Форма2** и **Объект**).

|  |
| --- |
| *'Во второй форме в 3-х текст. полях один результат- значение F2*  **Public Class Form2**  **Public F2 As String = "F2"**    **Public Sub Процедура\_формы2()**  **TextBox1.Text = F2**  **TextBox2.Text = Me.F2**  **TextBox3.Text = Форма2.F2**  **End Sub**  **End Class**  *'Во втором модуле две строки SM1 и SM2 получат одно значение M2*  **Module Module2**  **Dim SM1, SM2 As String**  **Public M2 As String = "M2"**  **Public Sub Процедура\_модуля2()**  **SM1 = M2 : SM2 = Module2.M2**  **End Sub**  **End Module**  *'В классе три строки SС1, SC2 и SC3 получат значение строки C1*  **Public Class Класс**  **Dim SC1, SC2, SC3 As String**  **Public C1 As String = "C1"**  **Public Sub Процедура\_класса()**  **SC1 = C1 : SC2 = Me.C1 : SC3 = Oбъект.C1**  **End Sub**  **End Class** |

*Рис. 2.4-3 Способы обращения к собственной переменной*

Из текста процедуры **Button1\_Click(…**) первой формы **Примера**   
2.4-1 (рис. 2.4-1) становится также ясно, что с переменными и процедурами формы можно работать еще до того, как форма появилась на экране.

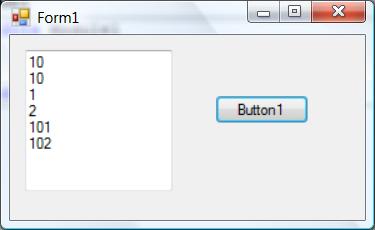
**Пример 2.4-3. Создать несколько объектов из одного класса*.***

Для иллюстрации работы с двумя объектами одного класса создать проект из формы, модуля и класса. На форму поместить кнопку, текстовое поле и ввести программный код, приведенный на рис. 2.4-4.

|  |
| --- |
| **Public Class Form1**  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button1.Click**  *'В строку SS для последующего вывода в TextBox1 «склеиваются»*  *'значения переменной* С *(поля класса), которые она имеет*  *'в двух разных объектах класса*  **Dim SS As String = ""**  *'В SS запишутся начальные значения полей Объекта1 и Объекта2*  **SS =****SS & CStr(Объект1.С) & vbNewLine**  **SS = SS & CStr(Объект2.С) & vbNewLine**  *'Изменение значений полей Объекта1 и Объекта2*  **Объект1.С = 1 : Объект2.С = 2**  **SS = SS & CStr(Объект1.С) & vbNewLine**  **SS = SS & CStr(Объект2.С) & vbNewLine**  *'вызов метода класса, который тоже изменяет*  *'поле* С *в Объекте1 и Объекте2*  **Объект1.Процедура()**  **SS = SS & CStr(Объект1.С) & vbNewLine**  **SS = SS & CStr(Объект2.С) & vbNewLine**  **TextBox1.Text = SS**  **End Sub**  **End Class**  *'В модуле создаются два объекта класса Класс*  **Module Module1**  **Public Объект1 As New Класс**  **Public Объект2 As New Класс**  **End Module**  **Public Class Класс**  **Public С As Integer = 10** *'начальное значение поля*  *'Процедура-метод класса изменяет значение поля в объектах класса*  **Public Sub Процедура()**  **Объект1.С = 101 : Объект2.С = 102**  **End Sub**  **End Class** |

*Рис. 2.4-4. Создание нескольких объектов из одного класса*

Результат представлен на рис. 2.4-5.



*Рис. 2.4-5. Результаты создания нескольких объектов одного класса*

Стандартный модуль **Module1** конструктором **New** создает из класса **Класс**два объекта: **Объект1**и **Объект2***.* При создании они абсолютно одинаковы и размещаются рядом в оперативной памя­ти. В каждом объекте выделяется память под пе­ременную (поле класса) **С**.

В программном коде класса переменная **С**инициализируется значением 10, а это значит, что она станет такой в обоих объектах, что и подтверждают результаты выполнения первых двух строк процедуры **Button1\_Click(…)**.

После выполнения следующих двух строк процедуры объекты перестают быть близнецами. Теперь в ячейке **С**у первого объекта единица, а у второго – двойка.

Объект может и сам изменять значения переменных, как в себе, так и у своего «брата». Это видно из кода процедуры в классе, вызов которой из событийной процедуры и приводит к этим изменениям. Таким образом, родившись одина­ковыми, близнецы в процессе жизни могут изменяться.

**Пример 2.4-4. Создать несколько объектов-форм из одного класса формы.**

Создадим проект из двух форм и модуля. На первую форму поместим кнопку, на вторую форму - 3 кнопки. Покажем, что из одного класса формы можно порождать несколько объек­тов-форм, причем эти объекты ведут себя так же, как и объекты, порождаемые из моду­ля класса в предыдущем примере, т.е. сначала одинаковы, а затем могут различаться. Это наглядно показывается при работе с цветом фона формы. Введем программный код, приведенный на рис. 2.4-6.

|  |
| --- |
| *'Модуль создает два объекта второй формы*  **Module Module1**  **Public Форма1 As New Form2**  **Public Форма2 As New Form2**  **End Module**  *'код Form1 показывает оба объекта (экземпляра) второй формы*  **Public Class Form1**  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button1.Click**  **Форма1.Show()**  **Фopмa2.Show()**  **End Sub**  **End Class**  *'Форма 2*  **Public Class Form2**  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button1.Click**  **Me.BackColor = Color.Blue**  **End Sub**  **Private Sub Button2\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) ­**  **Handles Button2.Click**  **Me.BackColor = Color.Red**  **End Sub**  **Private Sub Button3\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button3.Click**  **Фopмa1.Text = "Форма1"**  **Форма1.BackColor = Color.Black**  **Форма2.Техt = "Форма2"**  **Форма2.ВасkСоlоr = Color.White**  **End Sub**  **End Class** |

*Рис. 2.4-6. Создание нескольких объектов форм из одного класса формы*

При запуске проекта и нажатии на кнопку первой формы на экране будут отображены две совершенно одинаковые формы-близнеца с тремя кнопками каждая. Даже заголовок у них одинаковый – **Form2**. При щелчках по первым двум кнопкам близнецов каждый близнец красит себя, то в синий, то в красный цвет. Только себя. Потому чтов коде используется **Me**. Таким образом, близнецы очень скоро перестают быть похожими. Третья кнопка показывает, что каждая из форм может изменять не только себя, но и сестру.

Сходство и различие классов и стандартных модулей.

Основное различие между классами и модулями состоит в том, что классы могут быть инициализированы как объекты, а модули – нет. Классы являются объектно-ориентированными, а модули – нет. Пользователь может создать один или несколько экземпляров класса. Но существует только одна копия данных стандартного модуля, поэтому при изменении одной частью программы общей переменной в стандартном модуле при последующем чтении этой переменной любая другая часть программы получает измененное значение. Напротив, объект данных класса существует отдельно для каждого экземпляра объекта класса. В отличие от стандартных модулей, классы могут реализовывать интерфейсы, которые мы будем рассматривать в **Теме 4**.

Классы и модули используют различные области действия для своих членов. Члены, определенные внутри класса, находятся в определенном экземпляре класса и существуют только во время существования объекта. Для доступа к членам класса за пределами класса необходимо использовать полные имена в формате **Объект.Член**. С другой стороны, члены, объявленные в модуле, общедоступны по умолчанию, и доступ к ним может осуществить любой код, имеющий доступ к модулю. Это означает, что переменные в стандартном модуле являются фактически глобальными переменными, так как они видимы из любой части проекта и существуют во время выполнения программы.

Однако если член класса является статическим (когда к нему применяется модификатор **Shared**), он связан с самим классом непосредственно, а не через экземпляр класса. Непосредственный доступ к члену осуществляется с помощью имени класса. Тем же способом осуществляется доступ **к членам модуля.**

### 2.5. Пример разработки программных приложений с использованием классов

***Объект*** - ***это синтез данных (в частности, переменных и констант) и действий (процедур), которые эти данные обрабатывают***.

Объекты в программировании напоминают объекты реального мира, которые в некотором смысле представляют синтез данных и действий.

Почти все приложения Windows, и сама операционная система Windows создана на основе объектного подхода. Типичные примеры объектов в Windows – это окна, кнопки и пр. Все они характеризуются, с одной стороны, данными (размер, цвет и пр.), а с другой стороны, действиями, которые определяют их поведение.

**Пример 2.5-1. Создание и использование массива объектов одного класса.**

Пусть вы директор только что родившегося садового товарищества, в нем всего несколько участков. Однако вы уже решили использо­вать компьютер для решения повседневных задач.

Надо создать класс **УЧАСТОК***,* в котором собрать все ***данные****,* характеризующие участок (например, размеры), и ***действия***для вычисления нужных ве­личин, касающихся участка (например, его площади, или в нашей задаче будем вычислять, сколько краски понадобится для покраски забора на каждом участке).

Предполагается, что на форме имеется 6 текстовых по­лей (**TextBox**) и 4 кнопки (**Button**), а проект должен действовать следующим образом.

Одно текстовое поле (**TextBox6**) со свойством Multiline **=** *True* предназначено для вывода результатов работы проекта. В 5 текстовых полей вводится следующая информация, касающаяся садового участка:

**TextBox1** Владелец участка;

**TextBox2**  Длина участка;

**TextBox3**  Ширина участка;

**TextBox4** Высота забора на участке;

**TextBox5** Расход краски на 1 кв. м забора.

Эти данные будут полями класса **УЧАСТОК**. Кроме того, объявим в классе еще три поля: одно - под вспомогательную переменную **FPer** для расчета периметра участка, второе – поле **FNomer** для хранения номера участка и третье – статическое (разделяемое) поле **FKol** с модификатором **Shared** для числа созданных объектов класса **УЧАСТОК**. Все эти поля, (за исключением поля **Rashod1kv),** в соответствии с принципом инкапсуляции объявляем, как **Private**, и соответственно для доступа к этим полям создадим в классе свойства.

Поле **Rashod1kv**, в котором записывается расход краски на 1кв. м. является статическим, так как его значение является одинаковым (разделяемым, **Shared**) для всех участков. Это поле объявляется как общедоступное (**Public**), так как его значение вводится извне класса и для статических полей свойств нет.

Для создания объектов класса предусмотрены два конструктора: один - с параметрами, в котором сразу происходит заполнение полей объекта класса, и второй – без параметров. В обоих конструкторах при создании очередного участка увеличивается на единицу статическая переменная класса **FKol**. Номер участка **FNomer** равен количеству созданных к данному моменту садовых участков. Поэтому к моменту создания очередного участка эта переменная автоматически приобретает нужное значение:

|  |
| --- |
| **FNomer = FKol** |

В классе также предусмотрены методы для вычисления периметра и площади участка, а также метод вычисления расхода краски на весь забор для участка. Полный программный код класса приведен на рис. 2.5-1.

|  |
| --- |
| **Public Class УЧАСТОК**  *'====ПОЛЯ====*  **Private FNomer As Integer** *'Номер\_участка*  **Private FFam As String** *'Владелец участка*  **Private FLen As Double** *'Длина участка*  **Private FWidth As Double** *'Ширина участка*  **Private FHigh As Double** *'Высота забора*  **Private FPer As Double** *'Периметр-вспомогательная переменная*  *'статическое поле- счетчик созданных объектов (участков)*  **Private Shared FKol As Integer = 0**  *'Расход краски на 1кв м - разделяемое(статическое) поле*  *'д.б. объявлено как Public, т.к. иначе этому полю невозможно*  *'присвоить значение снаружи(извне класса)*  **Public Shared Rashod1kv As Integer**  *'=====Свойства======*  **Public Property Fam() As String**  **Get**  **Return FFam**  **End Get**  **Set(ByVal value As String)**  **FFam = value**  **End Set**  **End Property**  **Public Property Len() As Double**  **Get**  **Return FLen**  **End Get**  **Set(ByVal Value As Double)**  **If Value < 500 Then**  **FLen = Value**  **Else**  **MsgBox("Cлишком длинный участок")**  **FLen = 0**  **End If**  **End Set**  **End Property**  **Public Property Width() As Double**  **Get**  **Return FWidth**  **End Get**  **Set(ByVal value As Double)**  **FWidth = value**  **End Set**  **End Property**    **Public Property High() As Double**  **Get**  **Return FHigh**  **End Get**  **Set(ByVal value As Double)**  **FHigh = value**  **End Set**  **End Property**  *'для номера участка св-во только для чтения,*  *'т.к. номер определяется внутри класса*  *'кол-вом созданных объектов*  **Public ReadOnly Property Nomer() As Integer**  **Get**  **Return FNomer**  **End Get**  **End Property**  *'=============================*  *'Конструктор с параметрами*  **Public Sub New(ByVal fam As String, ByVal Leng As Double, \_**  **ByVal width As Double, \_**  **ByVal high As Double)**  **FFam = fam**  **Len = Leng**  **FWidth = width**  **FHigh = high**  **FKol = FKol + 1**  **FNomer = FKol**  **End Sub**  *'Конструктор без параметров*  **Public Sub New()**  **FKol = FKol + 1**  **FNomer = FKol**  **End Sub**  *' ===Внутренние методы====*  **Private Sub Вычисляем\_периметр()**  **FPer = 2 \* (FLen + FWidth)**  **End Sub**    **Private Function Площадь\_забора() As Double**  **Вычисляем\_периметр()**  **Return FPer \* FHigh**  **End Function**  *'Общедоступный метод вычисления*  *' расхода краски на весь забор*  **Public Function RashodAll() As Double**  **Return Rashod1kv \* Площадь\_забора()**  **End Function**  **End Class** |

*Рис. 2.5-1 Программный код класса* ***УЧАСТОК***

Итак, на форме имеется 6 текстовых по­лей (**TextBox**) и 4 кнопки (**Button**), а проект должен действовать следующим образом. В модуле формы объявляется массив объектов класса:

|  |
| --- |
| **Dim Mas(100) As УЧАСТОК** |

В 5 текстовых полей вводится информация, касающаяся одного садового участка.

После этого пользователь нажатием на кнопку **Button2** создает (конструктором без параметров) из класса **УЧАСТОК** объект этого класса, как элемент массива объектов класса, и созданный экземпляр принимает в себя все данные из текстовых полей в качестве своих полей.

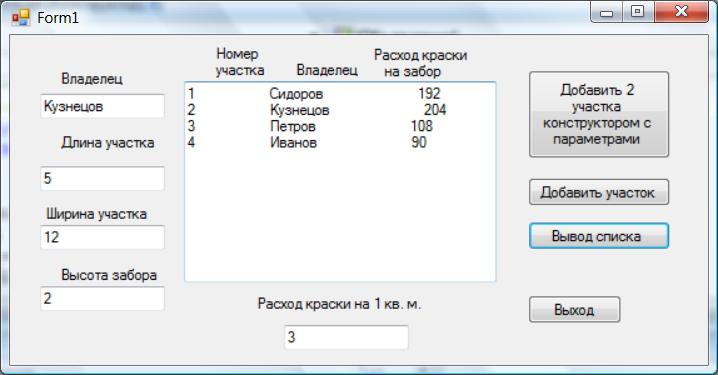
Затем пользователь вводит в текстовые поля информацию, касающуюся второго участка, и снова нажатием на кнопку **Button2** создает из класса **УЧАСТОК** объект, который также принимает в себя все данные из текстовых полей в качестве своих полей и т.д.

В учебных целях на форме предусмотрена кнопка **Button1**, в которой показано создание двух объектов класса конструктором с параметрами.

При нажатии на третью кнопку **Button3** в текстовое поле **TextBox6** выводится список созданных участков.Полный программный код модуля формы приведен на рис. 2.5-2, а результат работы проекта – на рис. 2.5-3.

|  |
| --- |
| **Option Strict On**  **Public Class Form1**  **Dim Mas(100) As УЧАСТОК** *'массив участков*  **Dim k As Integer = 0** *' счетчик элементов массива*  *'Кнопка добавления двух участков через конструктор с параметрами*  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button1.Click**  *'Обращение к статическому полю расход краски на 1кв.м.*  **УЧАСТОК.Rashod1kv = 3**  **k = k + 1 : Mas(k) = New УЧАСТОК("Петров", 5, 4, 2)**  **k = k + 1 : Mas(k) = New УЧАСТОК("Иванов", 800, 5, 3)**  **End Sub**  *'Кнопка добавления одного участка вводом значений в текст. поля*  **Private Sub Button2\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button2.Click**  *'Обращение к статическому полю расход краски на 1кв.м.*  **УЧАСТОК.Rashod1kv = CInt(TextBox5.Text)**  **k = k + 1**  **Mas(k) = New УЧАСТОК**  **Mas(k).Fam = TextBox1.Text**  **Mas(k).Len = CDbl(TextBox2.Text)**  **Mas(k).Width = CDbl(TextBox3.Text)**  **Mas(k).High = CDbl(TextBox4.Text)**  **End Sub**  *'Кнопка вывода списка созданных участков*  **Private Sub Button3\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button3.Click**  **Dim z As String = ""**  **TextBox6.Text = ""**  **For i As Integer = 1 To k**  **z = z & (Mas(i).Nomer) & Space(20) & Mas(i).Fam & \_**  **Space(25) & CStr(Mas(i).RashodAll) & vbNewLine**  **Next**  **TextBox6.Text = z**  **End Sub**      **Private Sub Button4\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button4.Click**  **End**  **End Sub**  **End Class** |

*Рис. 2.5-2 Программный код модуля формы   
с массивом объектов класса* ***УЧАСТОК***



*Рис. 2.5-3 Результаты работы программы   
с массивом объектов одного класса*

### 2.6. Задачи для самостоятельного решения

1. Построить концептуальную модель для предметной области из п. 1.4 предыдущей Темы 1.

1. Разработать программный проект, в котором класс должен содержать: закрытые поля, свойства для работы с закрытыми полями, конструкторы с параметрами и без параметров, и методы.

В событийной процедуре класса формы должна выполняться проверка всех разработанных членов класса.:

1. Описать класс **ТОЧКА**. Для точки задаются декартовы координаты **x**, **y**. Включить в описание класса методы, позволяющие вывести координаты точки на экран, рассчитать расстояние от начала координат до точки, переместить точку на плоскости на вектор **(a,b)** и свойство, позволяющее умножить координаты точки на скаляр (доступное только для записи).
2. Описать класс **ТРЕУГОЛЬНИК**, заданный длинами сторон. Включить в описание класса методы, позволяющие вывести длины сторон треугольника на экран, рассчитать периметр, площадь и высоты треугольника и свойство, позволяющее установить, существует ли треугольник с данными длинами сторон (доступное только для чтения).
3. Описать класс **ПРЯМОУГОЛЬНИК**, заданный длинами сторон. Включить в описание класса методы, позволяющие вывести длины сторон прямоугольника на экран, рассчитать периметр, площадь и диагональ прямоугольника, изменить его размеры, умножив его длины на скаляр, и свойство, позволяющее установить, является ли данный прямоугольник квадратом (доступное только для чтения).
4. Описать класс **ДЕНЬГИ**, заданный двумя полями, которые определяют номинал купюры и количество купюр. Включить в описание класса методы, позволяющие вывести номинал и количество купюр, определить, хватит ли денежных средств на покупку товара на сумму **N** рублей, и метод определения, сколько штук товара стоимости m рублей можно купить на имеющиеся денежные средства, а также свойство, позволяющее рассчитатать сумму денег (доступное только для чтения).
5. Описать класс **УГОЛ**, заданный величиной в градусах и минутах (двумя полями). Включить в описание класса методы, позволяющие реализовать перевод в радианы, привести величину угла к диапазону 0-360 градусов, увеличить и уменьшить угол на заданную величину и свойство, позволяющее определить, является ли угол тупым (доступное только для чтения).
6. Описать класс для работы с одномерным массивом целых чисел размером n. Включить в описание класса методы ввода элементов массива с клавиатуры, вывода массива на экран, нахождения макимального и минимального элементов массива и их индексов, а также свойства, позволяющее определить, является ли массив упорядоченным по убыванию (доступное только для чтения), и позволяющее домножить все элементы массива на скаляр (доступное только для записи).
7. Описать класс для работы с двумерным массивом целых чисел размером **nxn**. Включить в описание класса методы ввода элементов массива с клавиатуры, вывода массива на экран, вычисления суммы элементов i-го столбца, а также свойства: для вычисления количества нулевых элементов в массиве (доступное только для чтения), и позволяющее установить значение всех элементов главной диагонали массива равным скаляру (доступное только для записи).
8. Описать класс для работы с двумерным массивом вещественных чисел размером **nxm**. Включить в описание класса методы ввода элементов массива с клавиатуры, вывода массива на экран, отсортировать элементы каждой строки массива в порядке убывания, а также свойства: возвращающее общее количество элементов в массиве (доступное только для чтения), и позволяющее увеличить значение всех элементов массива на скаляр (доступное только для записи).
9. Составить описание класса многочлена вида **ax2+bx+c**. Включить в класс метод вывода описания многочлена на экран и метод вычисления значения многочлена для заданного аргумента, а также свойства: позволяющее определить, имеет ли квадратное уравнение действительные корни (доступное только для чтения), и позволяющее умножить многочлен на на скаляр (доступное только для записи).
10. Составить описание класса многочлена вида **ax+b**. Включить в класс методы: вывода описания многочлена на экран, вычисления значения многочлена для заданного аргумента, вычисления корня линейного уравнения с проверкой неравенства коэффициента b нулю, а также свойство, позволяющее возвести многочлен в квадрат (доступное только для записи).
11. Описать класс **ТРЕУГОЛЬНИК**, заданный длиной одной стороны и двумя прилежащими углами (в градусах). Включить в описание класса методы, позволяющие вычислить две другие стороны и третий угол треугольника, и свойство, доступное только для чтения, позволяющее установить вид треугольника (равносторонний, равнобедренный, прямоугольный и т.п.).
12. Описать класс **СЧЁТ**. Для банковского счёта задаются фамилия владельца, номер счёта, процент начисления за год и сумма в рублях. Включить в описание класса методы: пополнения счёта, снятия денег со счёта, перевода суммы в доллары и в евро, а также свойство, позволяющее начислить процент за заданное количество месяцев (доступное только для записи).
13. Описать класс **ДАТА**, заданный тремя полями для года, месяца и дня типа **UInteger**. Предусмотреть 2 конструктора с параметрами: для задания даты тремя беззнаковыми целыми числами и строкой в формате «год.месяц.день» и один конструктор без параметров. Включить в описание класса методы, позволяющие вычислить дату следующего дня, определить сколько дней осталось до конца месяца, и свойство, позволяющее выяснить, является ли год высокосным (доступно только для чтения).
14. Описать класс **ЗАРПЛАТА**. Для класса задаются поля: фамилия-имя-отчество, год поступления на работу, оклад в рублях, процент надбавки, количество отработанных дней в месяце, количество рабочих дней в месяце, начисленная и удержанная суммы. Включить в описание класса методы: вычисления начисленнной суммы, вычисления удержанной суммы, вычисления суммы, выдаваемой на руки а также свойство только для чтения, позволяющее определить стаж работы (вычисляется как полное количество лет, прошедших с момента зачисления на работу до задаваемого текущего года). Начисленная сумма вычисляется за отработанные дни месяца плюс надбавка. Удержания – подоходный налог 13 процентов.
15. Описать класс **ДАТА**, заданный тремя полями для года, месяца и дня типа **UInteger**. Предусмотреть 2 конструктора с параметрами: для задания даты тремя беззнаковыми целыми числами и строкой в формате «месяц.день.год» и один конструктор без параметров. Включить в описание класса методы, позволяющие вычислить дату предыдущего дня, вычислить дату через заданное число дней, и свойство только для чтения, позволяющее определить время года (зима, весна, лето, осень).
16. Описать класс **ВРЕМЯ**, заданный тремя полями для часов, минут и секунд типа **UInteger**. Предусмотреть 3 конструктора с параметрами: для задания времени количеством секунд (одним параметром), тремя беззнаковыми целыми числами, строкой в формате «час:минута:секунда», и один конструктор без параметров. Включить в описание класса методы, позволяющие перевести время в секунды, изменить время на заданное количество секунд, и свойство только для чтения, позволяющее определить время суток (утро, день, вечер, ночь).
17. Описать класс **ДРОБЬ**, заданный двумя целыми числами для числителя и знаменателя несократимой дроби. Включить в описание класса открытые методы умножения и деления дроби на заданное целое число, и закрытый метод сокращения дроби, который обязательно вызывается при ваполнении умножения и деления. Также предусмотреть свойство только для чтения, позволяющее вывести десятичное представление дроби в виде вещественного числа.
18. Описать класс **БАНКОМАТ**. В классе задаются поля для хранения идентификационного номера банкомата, минимальной и максимальной суммах денег, которые позволяются снять клиенту за один раз. Сумма денег, оставшаяся в банкомате, представляется 6-ю полями – номиналами российских рублей (10, 50, 100, 500, 1000, 5000), значениями которых является количество купюр данного достоинства. Включить в описание класса методы загрузки денег в банкомат и снятия определённой суммы денег, а также свойство, позволяющее вывести на экран сумму денег в банкомате в виде строки (доступное только для чтения).
19. Описать класс **ОКНО** для работы с моделями экранных окон. В качестве полей задаются координаты левого верхнего угла и размеры окна по вертикали и по горизонтали (целые числа), заголовок окна, состояние (видимое/невидимое). Включить в описание класса методы передвижения окна по горизонтали и по вертикали, изменения высоты и ширины окна с проверкой на пересечение границ экрана, и свойство, позволяющее установить, является ли данное окно квадратным (доступное только для чтения).
20. Описать класс **ТРЕУГОЛЬНИК**, заданный координатами трех вершин. Включить в описание класса методы вычисления сторон треугольника, и свойство, доступное только для чтения, позволяющее установить, принадлежит ли начало координат данному треугольнику.
21. Создать два проекта в одном решении и выполнить Windows-проект с несколькими исходными данными:
22. Создать DLL с классом **Point**, описывающим точку на плоскости, заданную декартовыми и полярными координатами. Среди методов класса **Point** предусмотреть метод **Distance()**, вычисляющий расстояние до заданной точки. Построить Windows-проект, предоставляющий пользователю интерфейс для работы с классом **Point**.
23. Создать DLL с классами **Point** и **Line**, описывающими точку и линию на плоскости. Построить Windows-проект, предоставляющий пользователю интерфейс для работы с DLL. Среди методов класса **Line** предусмотреть метод, определяющий принадлежность заданной точки линии.
24. Создать DLL с классами **Point** и **Square**, описывающими точку и квадрат на плоскости со сторонами, параллельными осям координат. Построить Windows-проект, предоставляющий пользователю интерфейс для работы с DLL. Среди методов класса **Square** предусмотреть метод, определяющий принадлежность заданной точки квадрату.
25. Создать DLL с классами **Point** и **Circle**, описывающими точку и круг на плоскости. Построить Windows-проект, предоставляющий пользователю интерфейс для работы с DLL. Среди методов класса **Circle** предусмотреть метод, определяющий принадлежность заданной точки кругу.
26. Создать DLL с классами **Point** и **Rectangle**, описывающими точку и прямоугольник на плоскости со сторонами, параллельными осям координат. Построить Windows-проект, предоставляющий пользователю интерфейс для работы с DLL. Среди методов класса **Rectangle** предусмотреть метод, определяющий принадлежность заданной точки прямоугольнику.
27. Создать DLL с классами **Point** и **Rhomb**, описывающими точку и ромб на плоскости с осями, параллельными осям координат. Построить Windows-проект, предоставляющий пользователю интерфейс для работы с DLL. Среди методов класса **Rhomb** предусмотреть метод, определяющий принадлежность заданной точки ромбу.
28. Создать DLL с классами **Point** и **Tetragon**, описывающими точку и четырехугольник на плоскости. Построить Windows-проект, предоставляющий пользователю интерфейс для работы с DLL. Среди методов класса **Tetragon** предусмотреть метод, определяющий принадлежность заданной точки четырехугольнику.
29. Создать DLL с классами **Point** и **Triangle**, описывающими точку и треугольник на плоскости. Построить Windows-проект, предоставляющий пользователю интерфейс для работы с DLL. Среди методов класса **Triangle** предусмотреть метод, определяющий принадлежность заданной точки треугольнику.
30. Создать DLL с классами **Account** и **Accounts**, описывающими счет в банке и множество счетов клиентов, хранящихся в файле. Построить Windows-проект, предоставляющий пользователю интерфейс для работы с DLL.

### 2.7. Лабораторная работа по теме

### *«Объектная модель, средства описания и использования классов в приложении на Visual Basic»*

**Цель** данной лабораторной работы состоит в изучении основных понятий объектно-ориентированной технологии и получении практических навыков разработки объектной модели, и создании класса с определением набора необходимых данных (полей), свойств и заданием набора методов (функциональных действий, выполняемых объектами класса).

#### 2.7.1. Задание

1. ***Изучите основные понятия*** объектно-ориентированной технологии и объектной модели и ***средства создания классов в*** *VB* ***(п.п. 2.1*** – ***2.5)***
2. ***Выберите вариант задания*** *из п. 2.7.2.*
3. ***Разработайте концептуальную модель задачи****, для чего:*

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

1. ***Разработайте объектную модель задачи логического уровня, определив необходимые связи между классами объектов****, для чего****:***

* определите образующие класс(ы) и их структуру;
* создайте схему 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*;*
* при необходимости приведите расчетные формулы, которые используйте для преобразования исходных данных в результаты, и геометрическую иллюстрацию преобразования исходных данных в результат.

1. ***Разработайте интерфейс пользователя – две формы в проекте:***

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

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

#### 2.7.2. Варианты индивидуальных заданий

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

#### 2.7.3. Содержание отчета

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

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

1. Программный проект на уровне реализации (приложения) с подробными комментариями:

6.1) Содержание классов проекта.

6.2) Схемы алгоритмов функциональных задач;

6.3) Программный код проекта с использованием классов и 2-х форм;

1. Схемы объектных моделей уровня реализации, построенных по программному коду.
2. Результаты выполнения приложения.
3. Обоснование правильности работы проекта.

#### 2.7.4. Пример выполнения задания

1. ***Тема и название работы:***

Объектная модель, средства описания и использования классов.

Формирование списка фамилий студентов, которым по итогам сессии

необходимо повысить успеваемость до фактического среднего

уровня.

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

В группе из **n** студентов получено в сессию по **m** оценок для каждого студента.

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 порядке убывания среднего балла, полученного за сессию, Исходные списки студентов и результатов сессии вводятся с клавиатуры.

1. ***Концептуальная объектная модель предметной области и задачи:***
2. **Проведите объектно-ориентированную декомпозицию, определив множество образующих классов.**

В результате анализа предметной области задачи и объектно-ориентированной декомпозиции определены два класса **СТУДЕНТ** и **ГРУППА** (рис. 2.7-1). Класс **СТУДЕНТ** содержит сведения об одном студенте (моделирует одного студента), а класс **ГРУППА** будет моделировать группу из **n** студентов (т.е. содержать массив из **n** объектов класса **СТУДЕНТ**). Кроме того, в классе **ГРУППА** будут содержаться общий средний балл всей группы и полученный список (фамилии и средние оценки) тех студентов, у которых средняя оценка ниже общего среднего балла группы, т.е. которым надо повысить успеваемость.

**

*Рис. 2.7-1. Объектная модель программного проекта   
на концептуальном уровне*

1. **Определите для выделенных классов существенные в заданной предметной области члены класса.**

Класс **СТУДЕНТ**:

***Входные атрибуты***:

* **fam** – фамилия студента;
* **m** – количество экзаменов в сессию;
* **oc(m-1)** - одномерный массив оценок студента.

***Выходные атрибуты:***

* **so** - средняя оценка студента;

***Операции:***

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

Класс **ГРУППА**:

***Входные атрибуты***:

* **n** – количество студентов в группе;
* **stud(n-1)** – одномерный массив, хранящий сведения о студентах группы (массив объектов класса **СТУДЕНТ**).

***Выходные атрибуты:***

* **sr** - общий средний балл за сессию студентов группы;
* **k** - количество студентов, которым надо повысить успеваемость, т.е. имеющих средний балл ниже общего среднего балла группы;
* **f(k-1)** - одномерный массив их фамилий;
* **sb(k-1)** – одномерный массив их средних оценок.

***Операции:***

* процедура **– Sred()** нахождения общего среднего балла группы;
* две перегружаемые процедуры **vvodGr()** для ввода данных о группе студентов с клавиатуры и из текстового файла;
* процедура **vivod()** для вывода сведений о группе студентов («собирает» все сведения обо всех студентах группы в одну строку);
* процедура **FormirSpisok()** формирования списка студентов, имеющих средний балл ниже общего среднего балла группы;
* процедура **sort()**- сортировка сформированного списка студентов по убыванию среднего балла;
* две перегружаемые процедуры **vivodMas()** для вывода результирующих массивов.

Объектная модель программного проекта на концептуальном уровне с учетом интерфейсных членов класса представлена на рис. 2.7-2. **!!!ПОМЕНЯТЬ в рис2.7-2, 2.7-3 названия классов – атрибуты и операции ГРУППЫ принадлежат СТУДЕНТУ и наоборот**



*Рис. 2.7-2. Объектная модель программного проекта   
на концептуальном уровне с учетом интерфейсных членов классов*

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



*Рис. 2.7-3. Объектная модель программного проекта   
на логическом уровне с учетом интерфейсных членов классов*

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).

Диаграмма классов объектной модели на логическом уровне представлена на рис. 2.7-3.

Геометрическая иллюстрация решения задачи (класс **ГРУППА** содержит сведения об **n** объектах класса **СТУДЕНТ**) приведена на рис. 2.7-4.

***Входные данные:***

***Вводимые Рассчитываемые***

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Фамилия**  **студента**  **fam**  0   |  | | --- | | Шакин | | Сосновиков | | Кравченко | | Загвоздкина | | Минязов | |  | |  |   n-1 | **СТУДЕНТ**  **Массив оценок**  **oc()**  **0 m-1**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **2** | **3** | **3** | **4** | **2** | | **3** | **3** | **3** | **3** | **3** | | **4** | **4** | **4** | **4** | **4** | | **4** | **5** | **4** | **5** | **4** | | **5** | **5** | **5** | **5** | **5** | |  |  |  |  |  | |  |  |  |  |  |   **n-1** | **Средняя оценка студента**    **so()**  0   |  | | --- | |  | |  | |  | |  | |  | |  | |  |   n-1 | **Общий средний балл всей группы**  **sr**   |  | | --- | |  | |

***Выходные данные:***

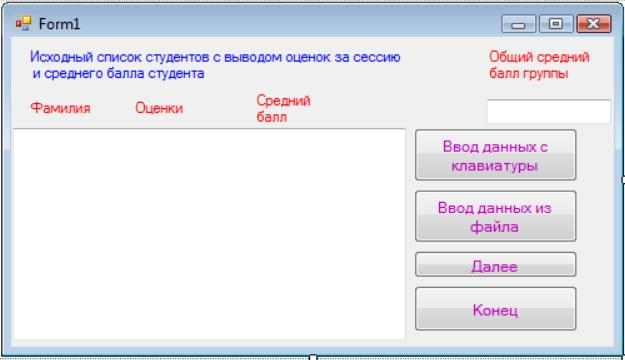
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Массив фамилий студентов с низким**  **средним баллом**  **f ( )**  0   |  | | --- | |  | |  | |  | |  |   k-1 | **Средние баллы, которые ниже общего среднего**  **sb( )**  0   |  | | --- | |  | |  | |  | |  |   k-1 |

*Рис. 2.7-4. Геометрическая интерпретация   
(входные и выходные данные) решения задачи*

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

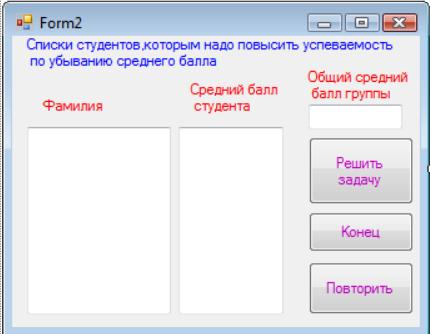
Разработайте две формы:

Форма, предназначенная для ввода исходных данных, как с кла- виатуры, так и из текстового файла, и для отображения входных данных на форме приведена на рис. 2.7-5;



*Рис. 2.7-5. Входная форма интерфейса пользователя*

Форма, предназначенная для вывода результатов решения задачи, на вторую форму приведена на рис. 2.7-6.



*Рис. 2.7-6. Выходная форма интерфейса пользователя*

1. ***Объектная модель задачи на уровне реализации.***

**6.1) Содержание классов:**

Определите содержание классов:

**Класс СТУДЕНТ:**

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

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

Для работы с внутренними 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 для чтения и записи полей класса:**
* **fam** – для чтения и записи фамилии студента;
* **m** – для чтения и записи количества экзаменов;
* **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:**
* 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).
* **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.

**Класс** **ГРУППА**:

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

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

Для работы с внутренними 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 для чтения и записи полей класса:**
* **stud** – для чтения и записи сведений о студентах (массива объектов класса **СТУДЕНТ**);
* **n** – для чтения и записи количества студентов в группе.
* **Свойства только для чтения полей класса:**
  + **sr** - для чтения общего среднего балла группы;
  + **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 методами **Sred()** и **FormirSpisok()** самого класса из значений массива **Fstud** объектов класса **СТУДЕНТ**.

* **Методы:**
* 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 **–** подпрограмма **FormirSpisok()** формирования списка студентов, имеющих средний балл ниже общего среднего балла группы;
* процедура **–** подпрограмма **sort()**- сортировка сформированного списка студентов по убыванию среднего балла;
* перегружаемые процедуры **vivodMas()** вывода одномерных массивов разного типа (для **String** и для **Double**);
* 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.

**6.2) Схемы алгоритмов функциональных задач:**

Схемы алгоритмов функциональных задач реализуются по указа-

нию преподавателя.

**6.3) Программный код проекта:**

Программный код решаемой задачи, реализованный с использо-

ванием двух классов и двух форм, представлен на рис. 2.7-7.

В программном коде проекта используются классы **СТУДЕНТ** и   
**ГРУППА** и две формы; исходные данные вводятся с клавиатуры или из текстового файла и отображаются на 1-ой форме; результаты отображаются на 2-ой форме.

|  |
| --- |
| **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 m и выделяет память под Foc()*  **Public Sub New(ByVal kol As Integer)**  **m = kol** *'заполнение поля кол-ва экзаменов через св-во m*  **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 студентов (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**    *' только для чтения целиком массива оценок студентов (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-оценки*  **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**  *'Класс ГРУППА*  **Public Class ГРУППА**  *'========== Поля ==========================*  **Private Fn As Integer** *'кол-во студентов в группе*  **Private Fstud() As СТУДЕНТ** *'массив студентов*  *'Результир. поля*  **Private Fsr As Double** *' Общий сред. балл группы за сессию*  **Private Fk As Integer** *'кол-во студ, к-рым надо повысить*  *'успеваемость*  **Private Ff() As String** *'массив их фамилий*  **Private Fsb() As Double** *'массив их средних баллов*  *'========== КОНСТРУКТОРЫ ================*  *' Конструктор класса получая кол-во студентов, через св-во n*  *'заполняет поле Fn и выделяет память под поля-массивы*  **Public Sub New(ByVal KolCt As Integer)**  **n = KolCt**  **ReDim Fstud(n - 1)**  **ReDim Ff(n - 1), Fsb(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**  *'Остальные свойства только для чтения,*  *'т.к. значения полей вычисляются*  *'методами самого класса и*  *'не могут изменяться снаружи*  **Public ReadOnly Property sr() As Double**  **Get**  **Return Fsr**  **End Get**  **End Property**  **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**  *'==========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 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 студент***е**  **st.vvod()**  *'созданный экземпляр класса СТУДЕНТ*  *'присваивается полю класса ГРУППА*  **stud(i) = st**  **Next**  *'подсчет среднего балла всей группы*  **Fsr = Sred()**  **End Sub**  *'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 класс 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**  *'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**  *'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,*  *' 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 < Fsr 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**  *'сортировка полей класса по убыванию среднего балла студента*  *'здесь используется значение поля кол-ва "нерадивых"*  *'студентов Fk, но т.к. нумерация элементов в массиве*  *'начинается от 0,а кол-во студентов считается*  *' от единицы,то 1-й цикл выполнается*  *'от 0 до k - 2, а 2-й цикл идет до k - 1*  **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) = Ff(j) : Ff(j) = temp**  **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 глобальной переменной G (не экземпляра!) класса*  *'**ГРУППА необходимо, т.к. она должна быть доступна*  *'для двух событийных процедур*  *'(кнопки ввода из файла и ввода с клавиатуры)*  *'и после создания объект этого класса*  *'должен быть доступен и на второй форме*  **Public G As ГРУППА**  *'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 экземпляра класса через конструктор*  **G = New ГРУППА(kolSt)**  *'вызов процедуры ввода группы с клавиатуры*  **G.vvodGr(ke)**  *'вызываем метод вывода сведений о группе*  **TextBox1.Text = G.vivod()**  *'вывод общего среднего балла группы*  **TextBox2.Text = CStr(G.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 экземпляра класса через конструктор*  **G = New ГРУППА(kolSt)**  *'вызов процедуры ввода группы из файла*  *'через массив строк файла*  **G.vvodGr(mas, kolSt)**  *'вызов метода вывода сведений о группе*  **s = G.vivod()**  **TextBox1.Text = s**  *'вывод общего среднего балла группы*  **TextBox2.Text = CStr(G.sr)**  *'Кнопку3 делаем доступной после ввода данных*  *'и расчета средних баллов*  **Button3.Enabled = True**  **End Sub**  *'Кнопка "Далее"*  **Private Sub Button3\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button3.Click**  **Form2.Show()**  **End Sub**  *'Кнопка "Конец"*  **Private Sub Button4\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) Handles Button4.Click**  **End**  **End Sub**  **End Class**  *'2 форма*  **Public Class Form2**  *' Кнопка решения задачи*  **Private Sub Button1\_Click(ByVal sender As System.Object, \_**  **ByVal e As System.EventArgs) \_**  **Handles Button1.Click**  **Dim z As String = ""** *'строка для вывода массивов*  *'вызов метода формирования списка нерадивых студентов*  **Form1.G.FormirSpisok()**  *'вызов метода сортировки списка нерадивых студентов*  **Form1.G.sort()**  *'вывод результир. полей-массивов(фамилий и сред. баллов)*  **Form1.G.VivodMas(Form1.G.f, z)**  **TextBox1.Text = z**  **Form1.G.VivodMas(Form1.G.sb, z)**  **TextBox2.Text = z**  *'вывод общего среднего балла группы для наглядности*  **TextBox3.Text = Format(Form1.G.sr, "0.0000")**  **End Sub**  **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.Button1.Enabled = True**  **Form1.Button2.Enabled = True**  **Form1.Button3.Enabled = False**  **Form1.Show()**  **End Sub**  **End Class** |

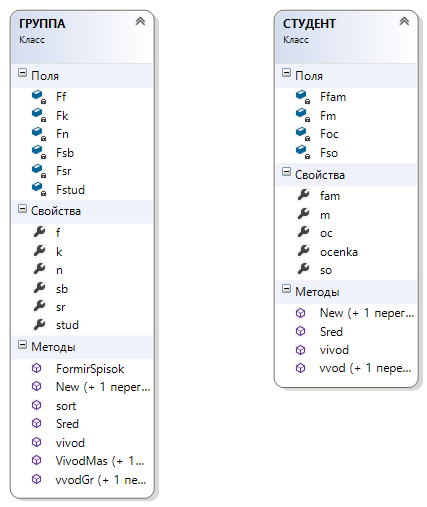
*Рис. 2.7-7. Программный код решения задачи*

*с использованием двух классов и двух форм*

1. ***Схема классов.***

Схема классов уровня реализации здесь приведена в качестве иллюстрации. Эта схема (рис. 2.7-8) была получена по готовому программному коду проекта VS .NET.

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.**

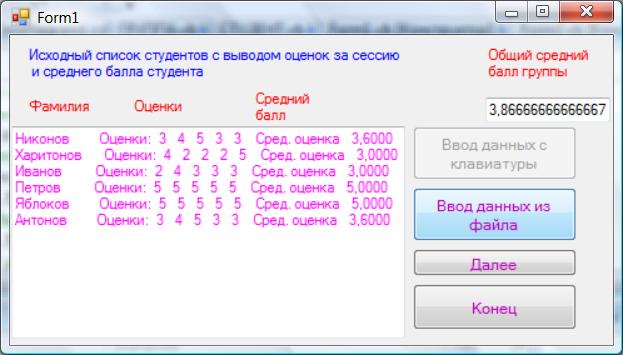


*Рис. 2.7-8. Схема классов уровня реализации*

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

Результаты выполнения программного проекта приведены на рис. 2.7-9

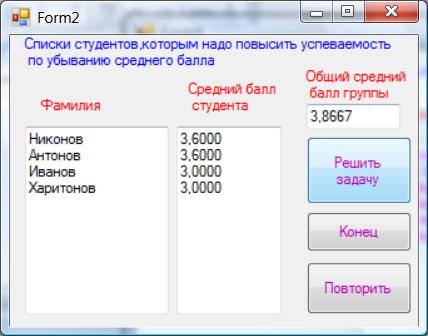
и рис. 2.7-10.



*Рис. 2.7-9. Результаты решения задачи*

*с использованием двух классов и двух форм*

*при вводе данных из файла на* **Form1**



*Рис. 2.7-10. Результаты решения задачи*

*с использованием двух классов и двух форм*

*при выводе данных на* **Form2**

1. ***Обоснование правильности работы проекта.***

Обоснование правильности работы программы приведите самостоятельно

### 2.8. Контрольные вопросы

1. Что означает объектно-ориентированный подход?
2. Что является классом в ООП?
3. Что является объектом в ООП?
4. Какие основополагающие принципы ООП известны?
5. Что лежит в основе ООП?
6. Что такое инкапсуляция?
7. Что такое наследование?
8. Что такое полиморфизм?
9. Из каких элементов состоит класс?
10. Какие модификаторы доступа известны?
11. Что такое модификаторы, устанавливающие принадлежность к классу или к экземпляру класса?
12. Как описывается класс?
13. Что может быть членом класса?
14. Какие правила обращения к переменным и процедурам?
15. Какие области видимости известны?
16. Какие средства описания свойства класса известны?
17. Что такое конструкторы и для чего они используются?
18. Какие члены класса называются разделяемыми (статическими)?
19. Какие члены класса называются экземплярными?
20. В чем сходство и различие стандартных модулей, форм и классов?
21. Что такое поле класса?
22. Что такое свойство класса?
23. Что такое метод класса?
24. Какая разница между экземпляром и классом? Типом данных и классом?
25. При решении каких проблем лучше использовать объектно-ориентированный подход?
26. Какие характеристики являются фундаментальными в объектно-ориентированном мышлении?
27. На каких принципах базируется объектная модель?
28. Какие преимущества дает объектная модель?
29. В чем заключаются преимущества инкапсуляции?
30. Из каких этапов состоит процесс построения объектной модели?
31. Какие действия обычно выполняются в части **set** свойства?
32. Может ли свойство класса быть не связанным с его полями?
33. Можно ли описать разные спецификаторы доступа к частям **get** и **set** свойства?