

Методическое пособие по курсу «Разработка приложений на платформе Microsoft .Net»

Данный курс рассчитан на 18 академических часов. Microsoft .Net находит все большее применение у системных разработчиков как очень удобное средство для разработки как Windows-приложений, так и Web-приложений. При этом механизм разработки один и тот же, что делает Microsoft .Net и Microsoft Visual Studio уникальным средством разработки приложений. С выходом осенью 2005 года .Net v2.0 и Visual Studio 2005 возможности по разработке расширились.

В данном методическом пособии проводится краткий обзор основных возможностей платформы Microsoft .Net. В пособии рассматриваются: объектно-ориентированный язык программирования C#, создание пользовательского интерфейса Windows-приложений, работа с БД. После каждого из этих этапов проводится лабораторная работа. Вкратце описываются возможности ASP.NET 2 и создание распределенных приложений на основе Web Services.

Содержание

Методическое пособие по курсу «Разработка приложений на платформе Microsoft .Net»	1
Содержание	2
Платформа Microsoft .Net	3
Язык C#. Основные конструкции	3
Структура программы	3
Библиотека типов и пространства имен	3
Компиляция программы	4
Структура типов	5
Перечисления	5
Структуры	5
Переменные	6
Статические и экземплярные переменные	6
Локальные переменные	7
Средства ввода и вывода	8
Средства вывода	8
Средства ввода	8
Массивы	9
Условные операторы и операторы циклов	9
Классы	10
Методы	11
Конструкторы	11
Свойства	12
Лабораторная работа №1	12
Библиотека Windows Forms	12
Класс Form, MessageBox и компоненты	13
Класс <i>Form</i>	13
Диалог <i>MessageBox</i>	14
Компоненты и панель <i>ToolBox</i>	14
Работа с элементами управления	16
Меню	17
Лабораторная работа №2	17
ADO.NET	17
Архитектура ADO.NET	18
Основные классы провайдера	19
Создание приложения	19
Размещение компонента отображения таблицы на форме	19
Создание соединения	19
Лабораторная работа №3	21
Рекомендации по курсу	22
Разбивка тем по часам	22
Рекомендации по лабораторным работам	22
Рекомендации по использованию электронных источников	22
Литература	22

Платформа Microsoft .Net

На протяжении нескольких лет платформа Microsoft .Net анонсировалась практически во всех источниках как новая, хотя эта платформа существует уже более 5 лет. Не будем углубляться в историю, скажем только, что она не нова и потому лишена многих «детских болезней».

Не будем вдаваться в подробности, описывая ее. Тем, кому интересны тончайшие нюансы этой платформы, могут найти их в книге [1]. Microsoft® .NET – это платформа нового поколения для создания распределенных бизнес-приложений. Основа .Net – это Microsoft .Net Framework – своеобразный верстак, набор средств и технологий по разработке и выполнению таких приложений.

.NET имеет в себе общезыковую среду времени выполнения (CLR, common language runtime). Ее можно сравнивать с Java Runtime Environment, хотя есть и отличия. Любой программный код, написанный под новую платформу, называется управляемым (managed code) и компилируется в бинарный вид, понятный .NET runtime. Этот формат называется Microsoft Intermediate Language (MSIL, IL). В связи с этим появляется возможность легкой интеграции кодов, написанных на разных языках программирования, т.к. платформа более низкого уровня у них одна.

В среду включен автоматический подсчет ссылок и сборщик мусора. Среда предоставляет расширенные возможности по управлению безопасностью кода, имеет стандартные возможности по работе с различными версиями одних и тех же компонент (versioning).

Язык C#. Основные конструкции

В платформу .Net входит много языков, которые без особого труда интегрируются друг с другом. Мы будем рассматривать язык C#, так как именно этот язык был создан специально для данной платформы, прочие языки были лишь адаптированы. Язык C# схож с языками C++ и Java, но, безусловно, имеет и отличия.

Структура программы

Рассмотрим структуру на самом классическом примере «Hello, world!».

```
class Hello
{
    public static void Main(string[] args)
    {
        System.Console.WriteLine("Hello, world!");
    }
}
```

Пример 1. Программа «Hello, world»

Любая программа на языке C# - это набор типов. В одном из типов программы должна находиться так называемая «точка входа» - статический метод Main. Наличие или отсутствие этого метода определяет тип получаемого результата компиляции – сборки. Если метод присутствует – получаем исполняемую программу, в противном случае – библиотеку DLL. Типы могут быть вложены друг в друга. Но точка входа должна быть только в одном типе¹.

Библиотека типов и пространства имен

Выше было сказано, что сборка в .Net – это набор типов. Также упоминали о том, что в .Net Framework входит большая библиотека типов, единая для всех языков этой платформы - Framework Class Library (FCL). В ней хранится множество типов. Библиотека

классов имеет в себе части, посвященные работе с данными (ADO.NET), с веб приложениями(ASP.NET), с windows интерфейсом (Windows Forms), с XML (DOM, XSD, XSLT, сериализация/десериализация, интеграция с ADO.NET), с сетевыми приложениями (System.Net), с распределенными приложениями (remoting), с COM+ (Enterprise Services, Serviced components), с шифрованием данных (System.Security.Cryptography).

Для удобства ее использования и удобства работы со всеми создаваемыми типами сама библиотека имеет иерархическую структуру пространств имен, в которых и находятся типы. Одним из наиболее важных пространств имен верхнего уровня является пространство имен **System**. Рассмотрим внимательнее строку примера

```
System.Console.WriteLine("Hello, world!");
```

В этой строке для вывода заветного **"Hello, world!"** мы используем метод **WriteLine** класса **Console**, который находится в пространстве имен **System**. Что же делать в случае, если нам предстоит тридцать раз использовать этот класс? Разумеется, сочетания клавиш <Ctrl+C> и <Ctrl+V> могут помочь. Но есть вариант получше. Рассмотрим следующий пример.

```
using System;  
class Hello  
{  
    public static void Main(string[] args)  
    {  
        Console.WriteLine("Hello, world!");  
    }  
}
```

Пример 2. Программа «Hello,world» с использованием using

Теперь мы вынесли пространство имен System в блок using, и можем далее нигде не указывать это пространство имен, если будем использовать типы из этого пространства. Следует заметить, что, если в этом пространстве имеются другие пространства имен, то указание этого пространства в блоке using не означает, что к классам вложенных пространств можно получить доступ без указания этого пространства.

Компиляция программы

Компиляция программ на C# осуществляется компилятор **csc.exe** (C Sharp Compiler). Как правило, при использовании Windows XP и .Net Framework v2.0 он находится в папке **c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727**. Этот путь при установке .Net Framework 2 прописывается в переменную среды net. То есть компилятор можно вызвать следующим образом **%net%\csc.exe**. Компилятор обладает множеством параметров. Укажем только один основной наиболее часто применяемый.

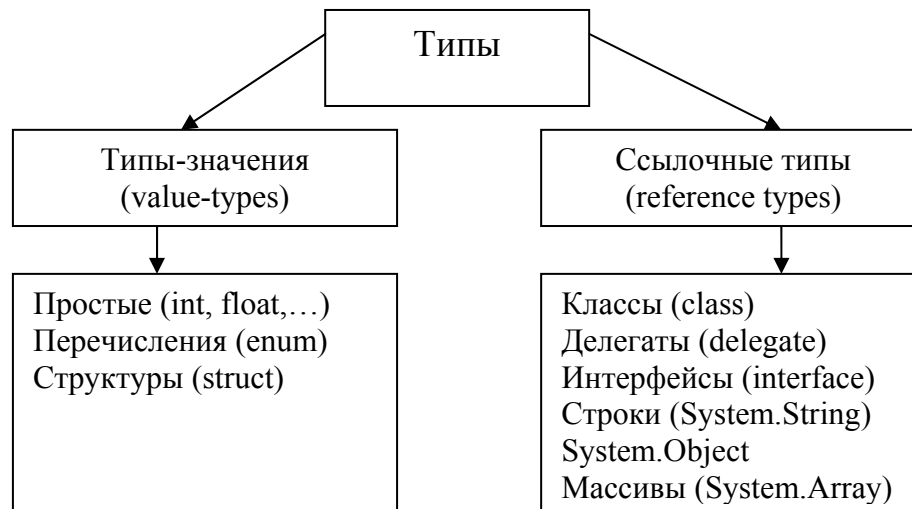
/t:exe	тип файла – консольное приложение
/t:winexe	тип файла –Windows приложение
/t:library	тип файла – динамическая библиотека
/t:module	тип файла – программный модуль

Параметр **/t:exe** берется по умолчанию. То есть, чтобы получить из нашего файла исполняемую программу, необходимо ввести следующую команду **%net%\csc hello.cs**, и мы получим файл **hello.exe**.

1. Возможен случай, когда статический метод Main имеется в нескольких типах. Но в данном пособии этот случай не рассматривается, поэтому будем исходить из того, что данный метод может быть только в одном типе.

Структура типов

Типы в C# имеют следующую структуру.



К простым типам относятся: **sbyte**, **byte**, **char**, **short**, **ushort**, **int**, **uint**, **long**, **ulong**, **float**, **double**, **decimal**.

Перечисления

Перечисление - это тип-значение, состоящий из набора поименованных констант. Укажем формат определения перечисления.

```
enum Имя [:базовый класс]
{
    Имя0 = Значение0, ..., ИмяN = ЗначениеN
}
```

По умолчанию нумерация начинается с нуля и увеличивается на 1. В качестве примера опишем перечисление, указывающее день недели:

```
enum DayOfWeek {Monday=1, Tuesday, Wednesday,
Thursday, Friday, Saturday, Sunday};
```

Структуры

Структуры во многом схожи с классами. Основное их отличие в том, что структуры не являются ссылочным типом. Дополнительно на структуры накладываются следующие ограничения:

- структура не может иметь деструктор;
- структура не может иметь конструктор без параметров;
- структура не может использоваться как базовый тип.

В качестве примера укажем структуру, описывающую окружность, которая в свою очередь использует структуру, описывающую точку.

```
struct Point
{
    public int x,y;
    public Point(int p1, int p2)
    {
        x=p1; y=p2;
    }
}

struct Circle
{

```

```
public Point Centre;  
public int Radius;  
public Circle(int p1,int p2,int p3)  
{  
    Centre=new Point(p1,p2) ;  
    Radius=p3;  
}  
  
}
```

Пример 3. Две структуры, описывающие точку и окружность.

Ссылочные типы будут рассмотрены ниже.

Переменные

C# в отличие от Visual Basic является языком со строгой типизацией. В C# определено 7 видов переменных: статические, экземплярные, элементы массивов, входные параметры, выходные параметры, локальные переменные.

Статические и экземплярные переменные

Статические переменные – это переменные класса, объявленные с указанием служебного слова **static**. Эти переменные одинаковы для всех экземпляров класса. Обращение к ним производится по имени класса, а не по имени экземпляра. Экземплярные переменные для каждого экземпляра могут хранить свое значение. Рассмотрим это на примере.

```
class Variables  
{  
    static int N1;  
    int N2;  
    public Variables(int p1, int p2)  
    {  
        N1=p1;  
        N2=p2;  
    }  
  
    public static void Main(string[] args)  
    {  
        Variables var1 = new Variables(10,20) ;  
        System.Console.WriteLine("Статическая пер.:  
"+Variables.N1.ToString());  
        Variables var2 = new Variables(100,200) ;  
        System.Console.WriteLine("Статическая пер.:  
"+Variables.N1.ToString());  
        System.Console.WriteLine("Экземплярные пер.:  
"+var1.N2.ToString()+" "+var2.N2.ToString());  
    }  
}
```

Пример 4. Статические и экземплярные переменные

После инициализации переменной **var2** значение **N1** изменилось, т.к. это общая переменная для всего класса.

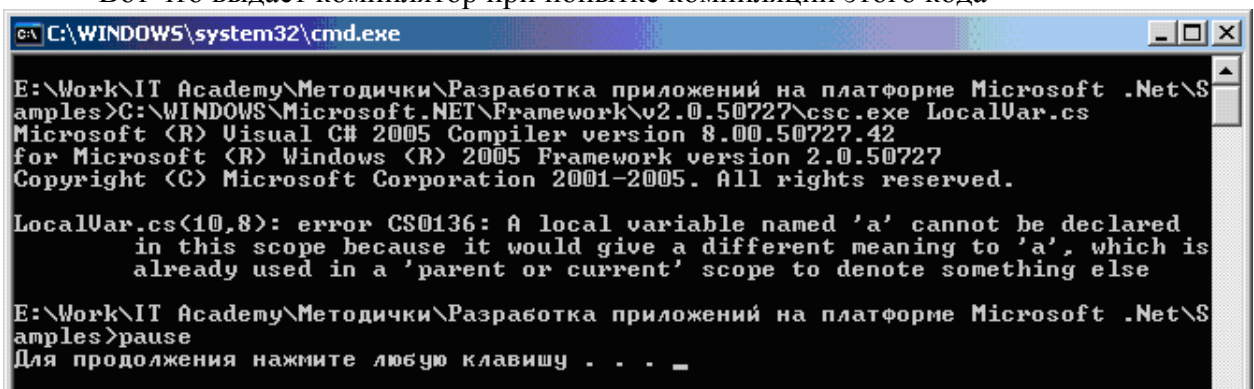
Локальные переменные

Локальные переменные – это переменные, объявленные в теле блока. Рассмотрим на двух следующих примерах. Первый – нерабочий, второй – исправный.

```
class Variables
{
    public static void Main(string[] args)
    {
        int a;
        {
            int b;
        }
        {
            int a; //компилятор сошлется на эту
строку, она пересекается с внешней переменной
            int b; // с этой переменной все в
порядке, т.к. она находится в разных блоках кода
        }
    }
}
```

Пример 5. Локальные переменные (код с ошибкой)

Вот что выдает компилятор при попытке компиляции этого кода



```
C:\WINDOWS\system32\cmd.exe
E:\Work\IT Academy\Методички\Разработка приложений на платформе Microsoft .Net\S
amples>C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\csc.exe LocalVar.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.

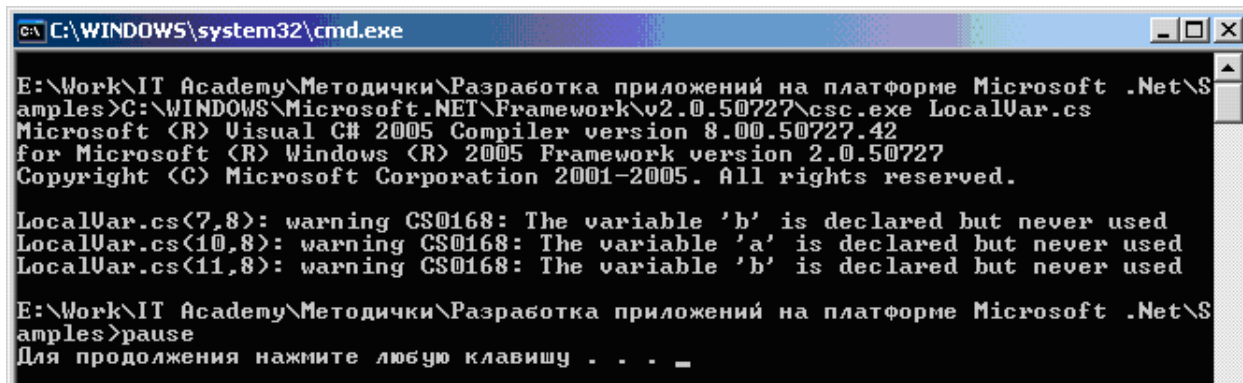
LocalVar.cs(10,8): error CS0136: A local variable named 'a' cannot be declared
in this scope because it would give a different meaning to 'a', which is
already used in a 'parent or current' scope to denote something else

E:\Work\IT Academy\Методички\Разработка приложений на платформе Microsoft .Net\S
amples>pause
Для продолжения нажмите любую клавишу . . . _
```

```
class Variables
{
    public static void Main(string[] args)
    {
        //
        int a;
        {
            int b;
        }
        {
            int a;
            int b;
        }
    }
}
```

Пример 6. Локальные переменные (без ошибки)

Компиляция этого кода хоть и проходит с предупреждениями о неиспользовании объявленных переменных, но без ошибок.



```
C:\WINDOWS\system32\cmd.exe

E:\Work\IT Academy\Методички\Разработка приложений на платформе Microsoft .Net\S
amples>C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\csc.exe LocalVar.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.

LocalVar.cs(7,8): warning CS0168: The variable 'b' is declared but never used
LocalVar.cs(10,8): warning CS0168: The variable 'a' is declared but never used
LocalVar.cs(11,8): warning CS0168: The variable 'b' is declared but never used

E:\Work\IT Academy\Методички\Разработка приложений на платформе Microsoft .Net\S
amples>pause
Для продолжения нажмите любую клавишу . . . _
```

Средства ввода и вывода

Средства вывода

Для организации консольного ввода и вывода используем статические методы класса **System.Console**.

```
Console.WriteLine("Hello, World!");
Console.Write("Hello, World!");
```

Это наиболее простой вариант вывода. Данные методы позволяют также осуществлять форматированный параметризованный вывод.

```
Console.WriteLine("{0}, {1}{2}", "Hello", "World", "!");
```

При помощи параметров и осуществляется форматирование строки.

Общий вид параметра **{N,M:F<R>}**, где:

- N** – номер параметра (начинаются с нуля);
- M** – ширина поля;
- F** – формат вывода;
- R** – количество выводимых разрядов.

Форматы вывода следующие:

- C** - форматирование числа как денежной суммы;
- D** - Целое число;
- E** - Вещественное число в виде 1e+3;
- F** - Вещественное число в виде 123.456;
- G** - Вещественное число в наиболее компактном формате;
- N** - Вещественное число в виде 123,456,789.5;
- X** - Целое число в шестнадцатеричном виде.

Средства ввода

C# позволяет осуществлять консольный ввод как одного символа, так и целой строки.

```
int i = Console.Read();
string str = Console.ReadLine();
```

После получения входной строки ее несложно преобразовать в необходимый тип.

```
string str = Console.ReadLine();
int i = Int32.Parse(str);
float f = Float.Parse(str);
double d = Double.Parse(str);
```


Массивы

Все массивы наследуются от базового класса **System.Array**. CLR поддерживает три вида массивов:

- одномерные массивы;
- многомерные массивы;
- вложенные (jagged) массивы.

Рассмотрим работу с массивами на базе примера.

```
class Arrays
{
    static void Main(string[] args)
    {
        int[] a = new int[3];
        int[] b = { 1,2,3 };
        int[] d; // d = null
        d = b; // d и c – это один и тот же массив!
        a = (int[])b.Clone(); //копируем массив
        int[,] e = new int[2, 3] { { 1, 2, 3 }, { 4,
5, 6 } }; //прямоугольный массив
        int[][] f = new int[2][]; //вложенный
массив
        f[0] = new int[3] { 0, 1, 2 };
        f[1] = new int[2] { 3, 4 }; //строки
вложенного массива могут быть разной длины
    }
}
```

Пример 7. Массивы

Для получения размерностей массива можно использовать следующие методы:

Length	возвращает количество элементов в массиве;
Rank	возвращает количество измерений массива;
GetLength(int i)	возвращает размер i-го измерения.

Условные операторы и операторы циклов

В C# имеются следующие условные операторы: **if**, **switch**; и операторы циклов: **for**, **while**, **do**, **foreach**. Оператор безусловного перехода **goto** не рассматривается как морально устаревший.

```
if (a>b)
{
    Console.WriteLine("a>b");
}
else
{
    Console.WriteLine("a<=b");
};
if (a==b) { Console.WriteLine("a=b");}; //условие
равенства в C# обозначается двумя знаками равенства
```

Пример 8. Условный оператор IF

```
int a=10;
switch (a)
```

```
{
    case 0: ...; break;
    case 1: ...; break;
    default: ...; break;
}
```

Пример 9. Условный оператор SWITCH

Операторы, указанные после **default**, выполняются в том случае, если ни одно из вышестоящих условий не было выполнено.

```
int[] a = { 10, 20, 30, 40 };
for (int i=0; i<a.Length; i++)
{
    System.Console.WriteLine(a[i].ToString());
}
```

Пример 10. Цикл FOR

Переменная-счетчик, объявляемая в заголовке цикла, существует только в теле цикла и является локальной переменной

```
int a=5678;
while (a>1)
{
    a /= 10;
}
```

Пример 11. Цикл WHILE

```
int a=5678;
do
{
    a /= 10;
} while (a>1);
```

Пример 12. Цикл DO

```
int[] a = { 10, 20, 30, 40 };
foreach (int i in a) { System.Console.WriteLine(i); }
```

Пример 13. Цикл FOREACH

Цикл **foreach** предназначен для работы с коллекциями и является аналогом цикла **for**.

Классы

Классы в C# имеют тот же смысл, что и в языках C++ и Java. Формально класс описывается следующим образом:

```
модификатор-доступа class Имя-класса {
    ... описание данных и методов ...
}
```

Модификаторы доступа определяют поле видимости данного класса. Для классов предназначены два модификатора доступа:

public класс доступен для других компонент;

internal класс видим только внутри данной компоненты (приложения).

По умолчанию применяется модификатор **internal**.

Членами класса могут быть:

- Константы
- Поля (field)
- Конструкторы (в том числе без параметров)
- Деструктор
- Методы (статические и экземплярные)
- Свойства (property)
- Индексаторы (свойства с параметрами)
- События (event)
- Вложенные типы

Модификаторы доступа также указываются и перед полями и методами класса:

- **private** (умолчение)
- **public**
- **protected**
- **internal** - поле видимо только внутри компоненты
- **protected internal**.

Методы

Формальное описание метода:

модификатор-доступа другие-модификаторы
возвращаемый-тип имя-метода (описание-параметров)
{ тело-метода }

Модификаторы доступа:

public, private, protected - аналогично C++

internal - метод видим только внутри компоненты

Другие модификаторы:

static метод класса (аналогично C++)

virtual, override, new управление виртуальными методами

Конструкторы

Конструктор – это метод, которые выделяет память под объект и инициализирует выделенную область.

Variables var = new Variables();

new выделяет память, а **Variables()** инициализирует ее.

Конструктор может быть с параметрами и без. Конструктор без параметров называется конструктором по умолчанию. Если в классе нет ни одного конструктора, то компилятор сам создает конструктор по умолчанию.

В классе может быть определено несколько конструкторов с различными списками параметров. Если в классе определен хотя бы один конструктор, конструктор по умолчанию не создается.

Конструктор по умолчанию для структуры всегда создается вне зависимости от наличия других конструкторов. При его использовании все поля структуры оказываются нулевыми. Конструктор для структуры должен явно инициализировать все поля данных и не может быть **protected**.

Статический конструктор – это конструктор класса. Он инициализирует статические поля класса.

Свойства

Свойство – пара методов со специальными именами. Метод `set()` вызывается при задании значения свойства, метод `get()` – при получении значения свойства. Обращение к свойству выглядит как обращение к полю данных, но транслируется в вызов одного из двух методов.

Определяя в свойстве только один из двух методов, получаем свойства только для чтения и для записи. Каждый из методов может иметь модификатор доступа.

Лабораторная работа №1

В этой лабораторной работе обучающиеся закрепляют знания по C#, создавая консольное приложение. Программа представляет собой автоматизированную систему учета банковских сведений.

На каждого клиента банка хранятся следующие сведения:

- Ф.И.О.;
- Возраст;
- Место работы;
- Номера счетов.

На каждом счете хранится информация о текущем балансе и история прихода, расхода. Для каждого клиента может быть создано неограниченное количество счетов. С каждым счетом можно производить следующие действия: открытие, закрытие, вклад денег, снятие денег, просмотр баланса, просмотр истории.

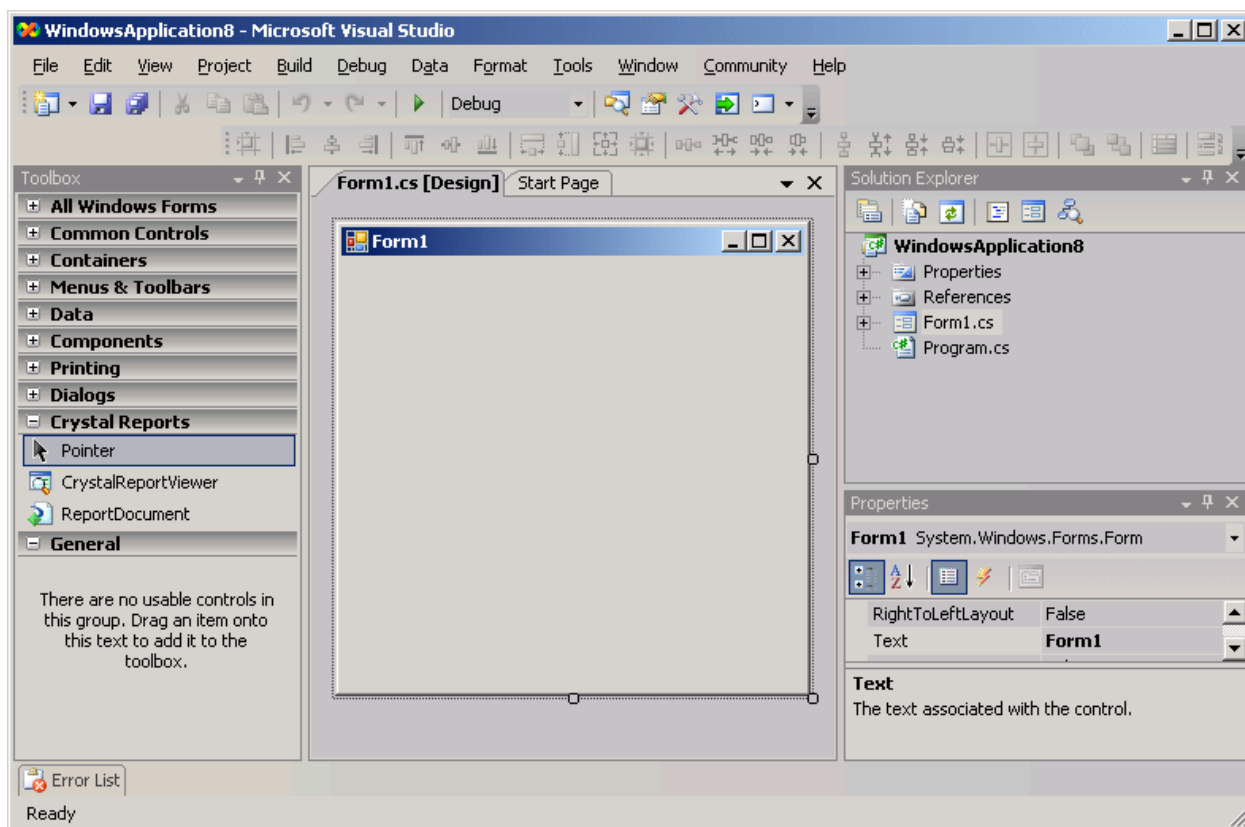
Вся информация должна храниться в массивах. Рекомендуется объекты клиента и счета реализовать в виде классов. Баланс счета организовать в виде свойства только для чтения.

Библиотека Windows Forms

Все рассмотренное выше является основой всех приложений .Net и удобно для создания консольных приложений, однако для создания серьезных приложений лучше подходит Microsoft Visual Studio 2005.

WindowsForms – подмножество типов библиотеки FCL для создания пользовательского интерфейса приложений. Все типы WindowsForms находятся в пространстве имен `System.Windows.Forms`.

Внешний вид Windows-приложения на стадии разработки в Microsoft Visual Studio 2005 представлен на рисунке.



Принцип визуального проектирования приложения в Visual Studio 2005 тот же, что и в Delphi. Visual Studio 2005 позволяет создавать как MDI, так и SDI-приложения.

Класс *Form*, *MessageBox* и компоненты

Класс *Form*

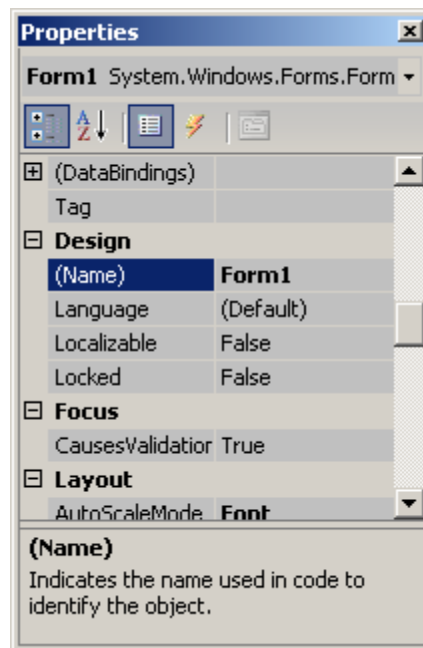
Основной класс – это класс **Form**. В .Net 2 появилась возможность разбивать класс по нескольким файлам. VS2005 разбивает описание класса на два файла. Первый называется <имя формы>.cs. В нем располагается пользовательский код по обработке разных событий формы. Второй - < имя формы>.Designer.cs. В нем располагается код формы, сгенерированной самой Visual Studio. Механизм разбиения классов можно применять и вручную.

```
partial class PartCl
{
    int a;
    int b;
}
partial class PartCl
{
    public PartCl(int p1, int p2)
    {
        a=p1;
        b=p2;
    }
}
```

Пример 14. Разбиение классов

Каждую из частей можно расположить в отдельном файле.

Свойства форм и обработчики событий формы отображаются в окне Properties.



Для создания в проекте новой формы необходимо открыть окно Solution Explorer (по умолчанию оно видно), в котором отображается структура всего проекта, и в нем в контекстном меню проекта выбрать пункты **Add->Windows Form**. После это надо выбрать нужный тип формы, ввести ее имя и нажать **OK**. Если окно Solution Explorer закрыто, его можно вызвать при помощи пунктов главного меню **View->Solution Explorer**.

За время жизненного цикла формы происходят следующие события с ней:

- **Load;**
- **GotFocus;**
- **Activated;**
- **Closing;**
- **Closed;**
- **Deactivate;**
- **LostFocus;**
- **Dispose.**

Отображение главной формы происходит при запуске программы. Остальные формы можно вызвать при помощи двух методов класса **Form**: **Show** и **ShowModal**. Метод **Show** отображает обычную форму, **ShowModal** отображает модальную форму.

Диалог **MessageBox**

Для вывода каких-либо сообщений можно использовать метод **Show** класса **MessageBox** из пространства имен **System.Windows.Forms**.

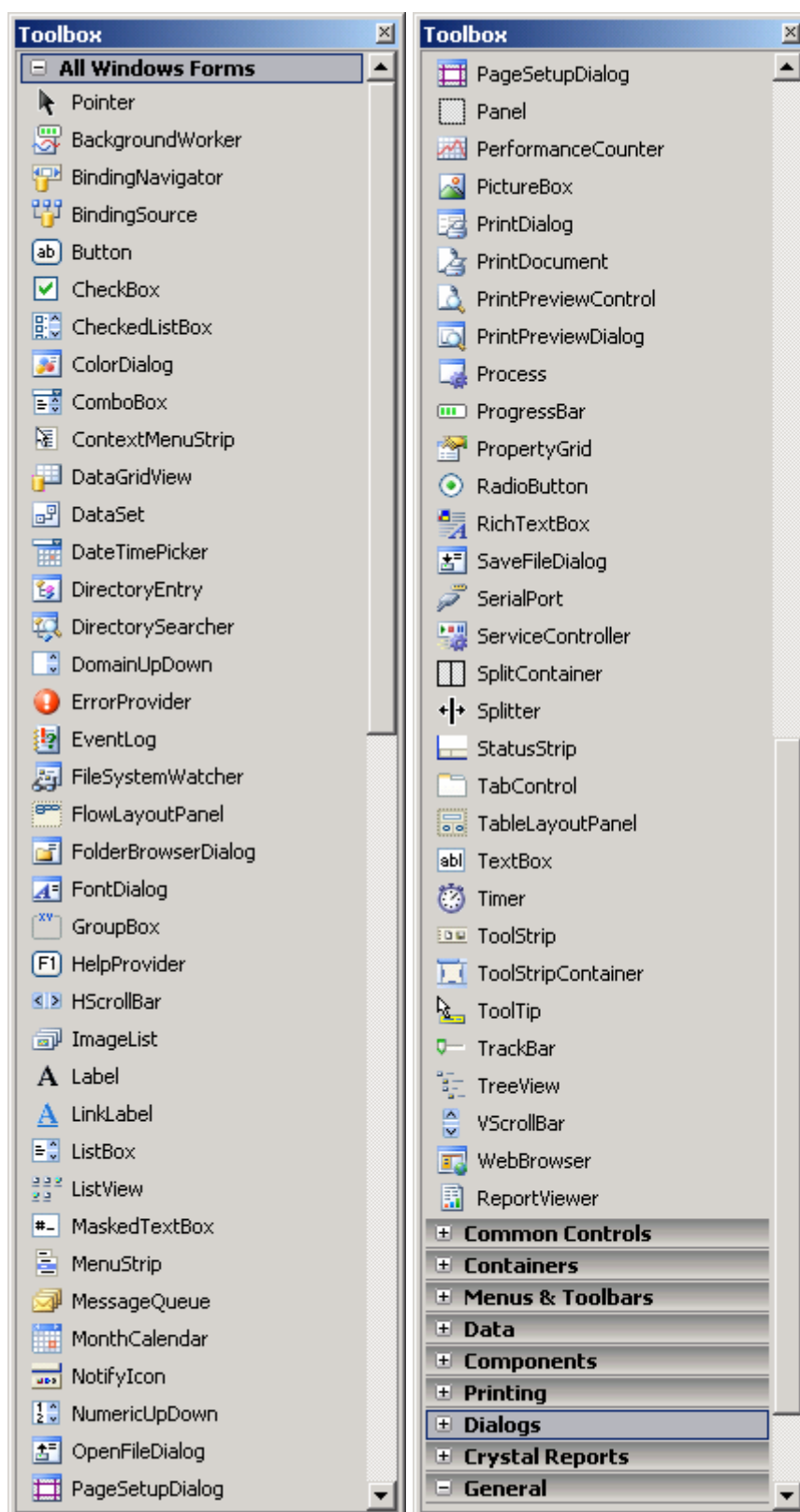
```
MessageBox.Show("This is a test", "Title",  
MessageBoxButtons.OK);
```

Используя этот класс, можно организовать простую интерактивность с пользователем.

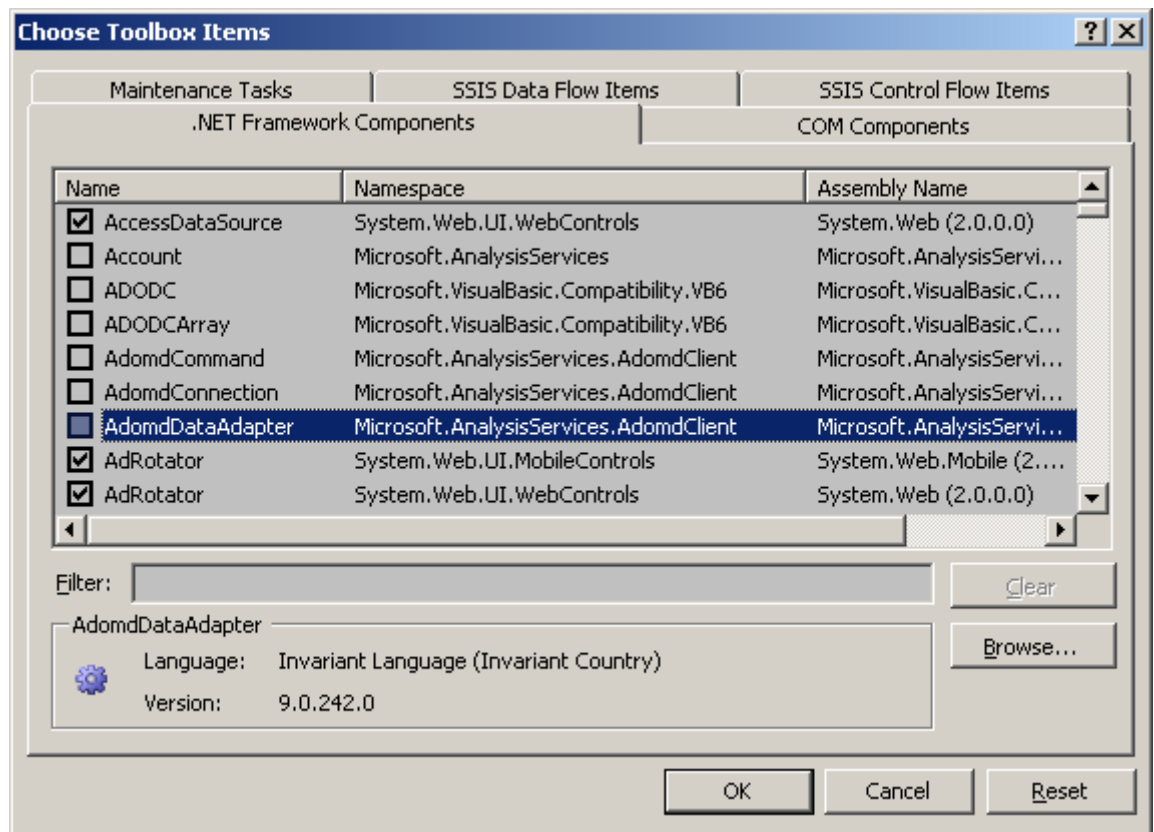
```
if(MessageBox.Show("Press Yes or No?", "Title",  
MessageBoxButtons.YesNo) == DialogResult.Yes) {...};
```

Компоненты и панель **ToolBox**

Компоненты панели управления для Windows-приложений представлены ниже на рисунке. Имея опыт программирования в визуальных средах проектирования, можно из названий догадаться о предназначении того или иного компонента.

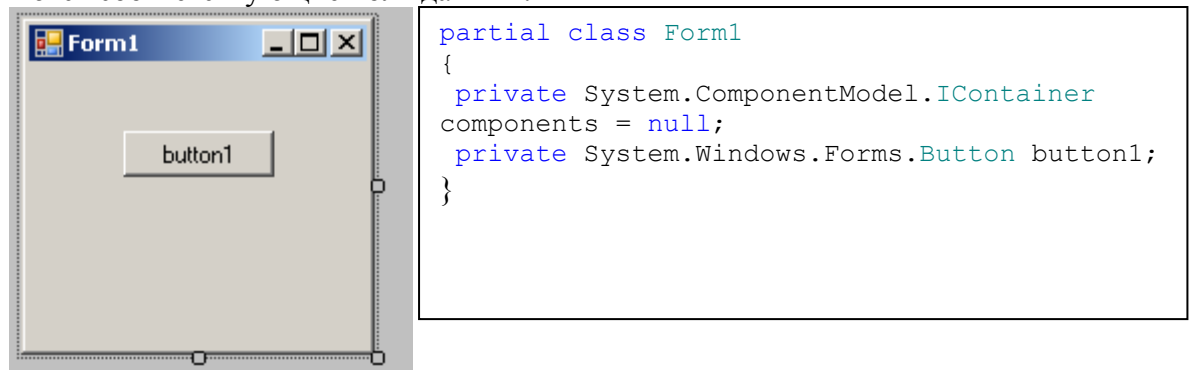


Изменение содержимого панели ToolBox осуществляется через диалоговое окно Choose ToolBox Items. Данное окно можно вызвать через пункт контекстного меню **Choose Items...** панели ToolBox.



Работа с элементами управления

При добавлении элемента управления из Toolbox на форму в класс формы добавляются соответствующие поля данных.



Если же мы какие-либо элементы впишем в код собственноручно, то они будут отображены на форме в дизайнера. Свойства полей данных формы могут быть считаны или изменены в процессе выполнения программы. Все свойства или установки обработчиков событий, сделанные программистом в дизайнера формы или в окне Properties, записываются в код метода **private void InitializeComponent()**.

Все обработчики событий формы являются ее методами. Любому обработчику событий передаются два параметра: объект, вызвавший событие и параметры этого события.

private void button1_Click(object sender, EventArgs e)

Параметры события являются экземпляром класса **EventArgs** или его потомком.

Для удобного размещения элементов управления на форме можно воспользоваться пунктами главного меню **Format**.

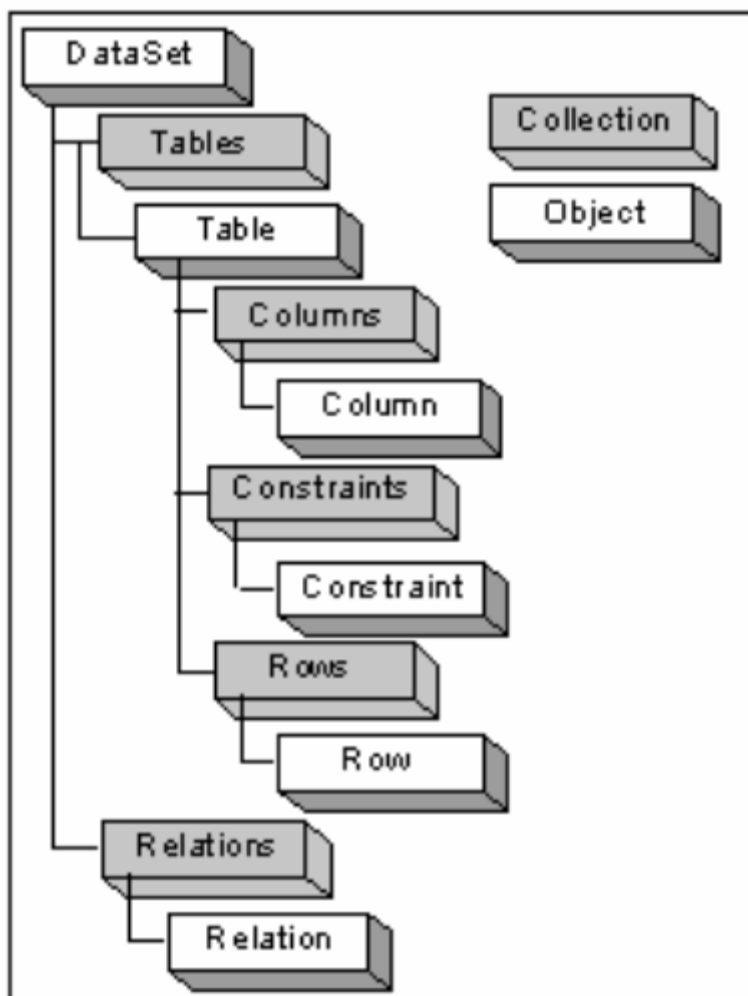
Для динамического изменения размеров двух соседних элементов управления имеется элемент **Splitter**.

Архитектура ADO.NET

Доступ к данным основан на двух компонентах: наборе данных и провайдере данных. По сути, набор данных это локальная копия фрагмента БД. Набор данных представлен экземпляром класса **DataSet**. Работа с набором данных осуществляется в три шага:

1. Загрузка данных с сервера;
2. Изменение данных в наборе на локальной машине;
3. Обновление данных на сервере на основе локальной копии.

Компонент **DataSet** содержит два важных набора компонентов: **DataTable** и **DataRelation**.



Каждый объект **DataTable** представляет собой одну таблицу из БД. **DataTable** состоит из компонентов:

- DataColumns** – описания столбцов таблицы;
- Constraints** – ограничения на данные в таблице;
- DataRows** – собственно, набор данных в таблице.

Объект **DataRelation** задает связи между таблицами.

Провайдер данных отвечает за взаимодействие с БД. .NET Framework предоставляет два наиболее используемых провайдера данных.:

- SQL Server .NET Data Provider – осуществляет взаимодействие с СУБД Microsoft Sql Server 7.0 и выше;
- OleDb .NET Data Provider – осуществляет взаимодействие с БД других типов.

Каждый из провайдеров представляет собой набор функционально взаимосвязанных классов: **Connection**, **Command**, **DataReader**, **DataAdapter**.

Основные классы провайдера

SqlConnection и **OleDbConnection** – обеспечивают подключение к БД.

SqlCommand и **OleDbCommand** – управляют источником данных с помощью SQL.

SqlDataReader и **OleDbDataReader** – обеспечивают последовательный доступ к результату выполнения команды Select. Для работы требуют монопольный доступ к соединению.

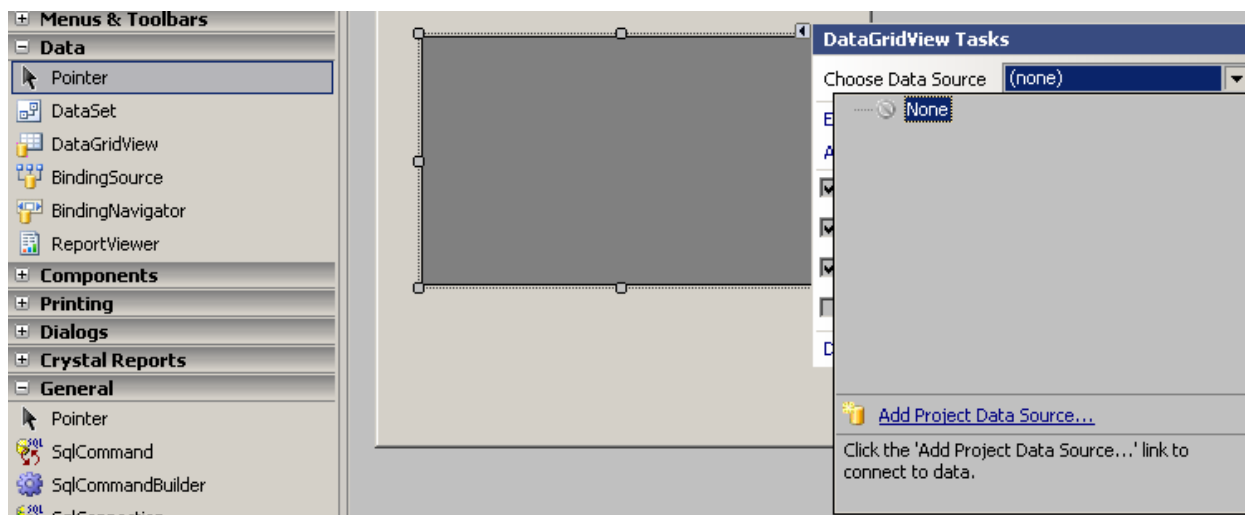
SqlDataAdapter и **OleDbDataAdapter** – заполняют отсоединенный объект **DataSet** или обновляют данные на сервере из **DataSet**.

Создание приложения

Visual Studio2005 делает процесс создания приложения БД очень простым. Достаточно работы в дизайнере формы. Рассмотрим создание приложения, отображающего данные из БД Microsoft SQL Server 2005.

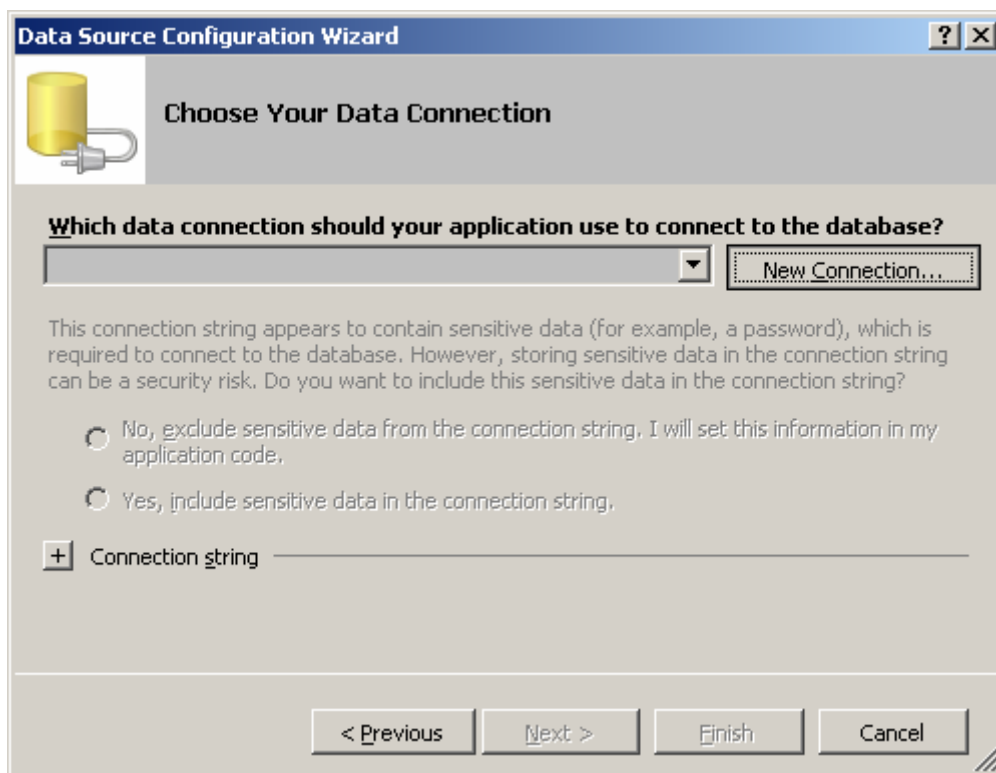
Размещение компонента отображения таблицы на форме

Элементом управления, отображающим таблицу на форме, является **DataGridView**.



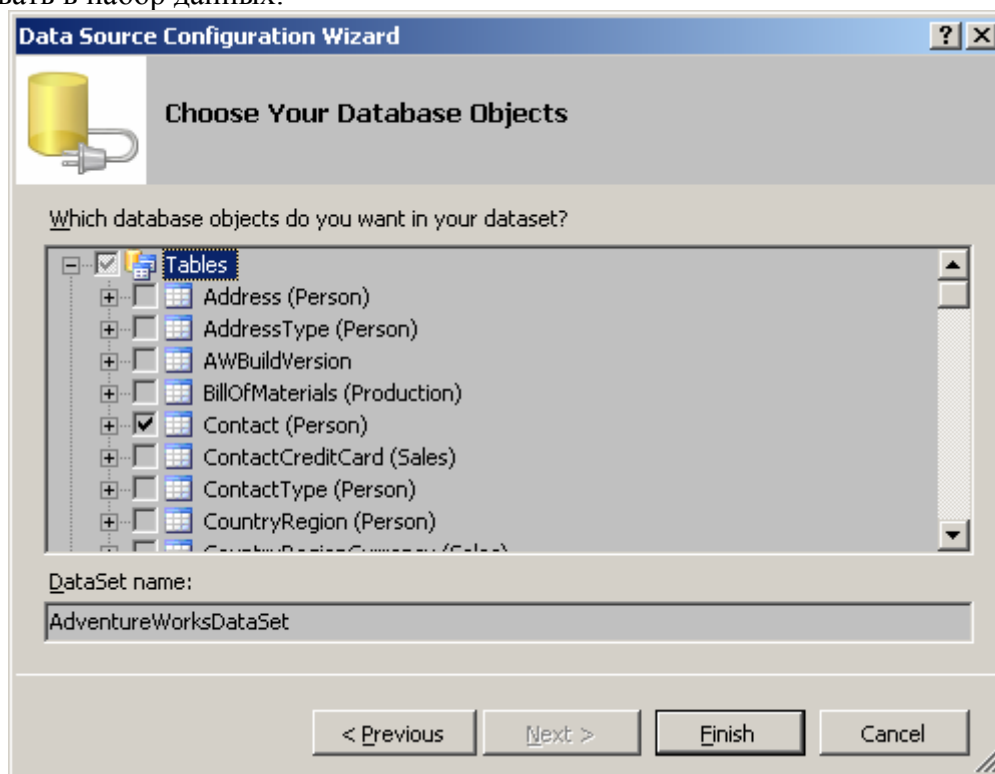
Создание соединения

Щелчком на ссылке **Add Project Data Source...** создаем источник данных. Открывается форма мастера **Data Source Configuration Wizard**. На втором шаге мастера. Создаем соединение, если у нас подходящего нет.



В диалоговом окне **Choose Data Source** в поле **Data Source** выбираем Microsoft Sql Server и нажимаем **<Continue>**. Далее в диалоговом окне **Add Connection** указываем все параметры подключения и требуемую БД. Нажимаем **<OK>**. Возвращаемся к мастеру **Data Source Configuration Wizard**. Дважды нажимаем кнопку **< Next >**, сохранив соединение под предлагаемым по умолчанию именем.

На шаге **Choose Your Database Objects** выбираем объекты, которые хотим скопировать в набор данных.



Заканчиваем работу мастера нажатием на кнопку **<Finish>**.

Microsoft Visual Studio 2005 всю работу сделает за нас и на форме мы увидим готовую таблицу. После запуска приложения можно работать с БД.

Обновление данных на сервере осуществляется через объект **DataAdapter**. В нашем случае эта команда будет выглядеть следующим образом.

```
this.contactTableAdapter.Update(this.adventureWorksDataSet);
```

Ключевое слово **this** означает ссылку на объект-владелец, т.е. на саму форму **Form1**.

Лабораторная работа №3

Создать в Microsoft SQL Server 2005 БД Bank. В этой БД хранить информацию о клиентах и их счетах. В Microsoft Visual Studio 2005 переделать программу, созданную в лабораторной работе №2.

Рекомендации по курсу

Данный курс рекомендуется проводить в 5 дней, в первые 4 дня - по 4 академических часа, в последний день - 2 ч.

Разбивка тем по часам

Платформа Microsoft .Net – 1 ч.;
Язык C#. Основные конструкции – 6 ч;
Библиотека Windows Forms – 5 ч.;
ADO.NET – 6 ч.

Рекомендации по лабораторным работам

Для лабораторных работ рекомендуется организовать заготовки, содержание которых в курсе не дается. Например: создание структуры БД для работы и предложение уже готовой структуры.

Для незнающих язык SQL можно составить краткие аннотации для основных команд.

Для нормального проведения лабораторных работ желательно использовать ПК со следующими характеристиками:

- процессор с частотой не ниже 2 ГГц;
- объем оперативной памяти не ниже 512 Мб (лучше 1 Гб).

Для удобства работы желательно использовать мониторы с диагональю экрана 17” или 19”.

Рекомендации по использованию электронных источников

При подготовке курса можно использовать материалы **e-Learning**. Рекомендуется использовать материалы по следующим курсам:

- [Collection 2364: What's New in Microsoft® Visual Studio® 2005 for Existing Visual Studio .NET Developers](#);
- [Collection 5160: Core Development with the Microsoft® .NET Framework 2.0 Foundation \(formerly part of Collection 2956\)](#);
- [Collection 5161: Advanced Development with the Microsoft® .NET Framework 2.0 Foundation \(formerly part of Collection 2956\)](#);

Эти материалы и многие другие по Microsoft SQL Server 2005 доступны по ссылке <https://www.microsoftlearning.com/catalog/itproDev.aspx#VisualStudio>.

Также можно рекомендовать использовать материалы MSDN, доступны через справку и на сайте <http://www.microsoft.com/learning/2005/vstudio/default.mspx>.

Информацию по базовым курсам можно получить на сайтах:

- <http://it-university.ru>;
- <http://it-university.samgtu.ru>.

Литература

1. Рихтер Дж. Программирование на платформе Microsoft .Net Framework. Мастер-класс. / Пер. с англ. – 3-е изд. – М.: Издательско-торговый дом «Русская редакция»; СПб.: Питер, 2005. – 512 стр.: ил.
2. Робинсон С., Корнес О., Глин Д. и др. C# для профессионалов. Том I,II. – М.: «Лори»
3. Знакомство с Microsoft .NET. Д. Платт, 2001 г., 240 стр., с ил.
4. Э. Гуннерсон. Введение в C#

5. Герберт Шилдт. С#: учебный курс. Издательство Питер.
6. Программирование для Microsoft .NET. Дж. Просиз, 2002 г., 704 стр.
7. Программирование для MS Windows на С#. том 1/2. Ч. Петцольд, 2002 г., 624 стр., с ил.
8. Microsoft ADO.NET. Д. Сеппа, 2003 г., 640 стр., с ил.
9. Разработка Windows-приложений на MS VB .NET и MS VC# .NET. Учебный курс Сертификационный экзамен №№70-306, 70-316.