Байдачный С.С.

«СОЛОН»

.NET Framework 2.0

Секреты создания Windows-приложений

- Новые возможности языка программирования C # 2.0
- Использование .NET Framework 2.0 и Visual Studio .NET 2005 для эффективного создания пользовательского интерфейса
- Использование XML при работе с базами данных
- Взаимодействие с COM и неуправляемым кодом
- Создание собственных элементов управления
- Взаимодействие приложений с данными через ADO.NET 2.0
- Новые возможности SQL Server 2005 при построении приложений на .NET Framework 2.0
- **Подготовка к сдаче экзамена для получения статуса MCSD.NET**

Библиотека Профессионала

УДК 621.396.218
ББК 32.884.1
Б18

С. С. Байдачный
Б18 .NET Framework 2.0. Секреты создания Windows-приложений. — М.: СОЛОН-Пресс, 2010. — 520 с.: ил. — (Серия «Библиотека профессионала»).

ISBN 5-98003-245-2

На сегодняшний день .NET Framework настолько опередила конкурирующие платформы, что предполагается платформой «по умолчанию» для разработки любых программных продуктов. Это связано с тем, что компания Microsoft не просто учла ошибки других производителей и пожелания разработчиков, но и внесла много нового в разработку приложений всех типов. .NET Framework реализует множество идей, которые раньше были лишь на бумаге.

Компания Microsoft, как разработчик платформы .NET, постоянно совершенствует и расширяет возможности этой платформы. В 2005 году появится очередной релиз — .NET Framework 2.0. Совместно с SQL Server 2005 и Visual Studio .NET 2005 он станет мощным инструментом для разработки приложений. Купив эту книгу, вы будете готовы к появлению новой версии платформы, так как тут идет речь именно о .NET Framework 2.0.

Материал книги позволит освоить основы программирования на языке C# и приступить к профессиональной разработке Windows-приложений. Вы сможете убедиться, что разрабатывать приложения с использованием .NET Framework так же просто, как и Windows-приложения с использованием таких продуктов, как Visual Basic или Delphi. Так, Visual Studio.NET предлагает редактор форм, позволяющий разрабатывать интерфейс Windows-приложений с использованием технологии Drag and Drop. Система IntelliSense избавит вас от ошибок при наборе имен методов, а возможности рефакторинга позволят создавать ясные приложения без дополнительных усилий. Кроме того, вы сможете разворачивать готовые приложения на клиентских машинах, используя лишь мышь.

Книга была написана таким образом, чтобы удовлетворить требования Microsoft к сдаче экзамена 70-316 — «Разработка Windows-приложений с использованием Visual Studio.NET». Таким образом, взяв эту книгу за основу и немного попрактиковавшись, вы сможете пройти еще одну ступень в **получении сертификата MCSD**.

Книга будет понятна и начинающим программистам. Поэтому ее можно использовать как для самостоятельного изучения .NET Framework, так и как пособие для профессионала, который решил перейти к разработке приложений с использованием .NET Framework 2.0.

УДК 621.396.218
ББК 32.884.1

Распространение ООО «Альянс-книга» (095) 258-91-94 www.abook.ru

www.solon-press.ru. E-mail: solon-avtor@coba.ru

ISBN 5-98003-245-2

Глава 2
Переменные

Понятие переменной

В процессе выполнения программы существует необходимость где-то временно или постоянно хранить данные, которые необходимы для работы приложения. Это означает, что должны быть механизмы, которые реализуют различного рода хранилища.

Выделяют два типа хранилищ — временные и постоянные.

Под постоянными хранилищами понимают жесткие диски, компакт-диски и все те устройства, которые способны сохранять информацию после завершения работы приложения и после выключения компьютера.

В качестве временного хранилища используется оперативная память, которая выделяется операционной системой на время работы программы для хранения промежуточных данных, однако после завершения работы программы эти данные не сохраняются.

Для доступа к информации в оперативной памяти используют переменные. Это своеобразные ячейки, которые расположены в памяти и способны хранить какую-то информацию. Информацию можно получать и изменять, используя имя переменной. Кроме необходимости знать имя переменной, необходимо знать еще и ее тип, который характеризует размер выделяемой памяти и способ хранения информации. Тип данных определяет допустимые величины переменных и операции, которые можно выполнять с переменной. Таким образом, переменная определяется двумя параметрами — типом переменной и именем переменной. Наиболее общий синтаксис определения переменной показан ниже:

<тип переменной> <имя переменной>;

Типы данных

Каждый язык программирования предоставляет свой набор встроенных типов, причем количество встроенных типов может различаться для различных языков. Это затрудняет повторное использование компонентов, написанных на одном языке, в приложении, разрабатываемом на другом языке программирования. Поэтому разработчикам приходилось быть достаточно внимательными и предусматривать возможность преобразования типов. В .NET Framework эта проблема отсутствует, и вы можете разрабатывать ваше приложение, используя одновременно несколько языков программирования.

Для того чтобы сделать возможным написание приложений на любом из языков, .NET Framework предоставляет общую систему типов (**Common Type System** — CTS). Используя эту систему, можно разрабатывать приложение на любом из языков, поддерживающих .NET Framework, и при этом не заботиться о том, что типы будут несовместимы. Рассмотрим более детально, как реализована CTS.

Итак, языки программирования на платформе .NET не содержат типов. В то же время в языке программирования C# существует ряд простых типов. Напри-

мер, тип int. Но если мы используем другой язык, к примеру, Visual Basic, то там этот тип отсутствует, а вместо него мы найдем Integer. Так в чем же дело? На самом деле, ключевые слова, описывающие типы, являются псевдонимами соответствующих классов или структур, представляющих эти типы в .NET Framework. Именно структур, так как в .NET Framework даже простые типы являются структурами или классами, порожденными от класса **System.Object**. Поэтому int — лишь псевдоним структуры **System.Int32**. Вы можете использовать как псевдонимы, так и непосредственно имена соответствующих структур или классов. Аналогично тип Integer в Visual Basic является псевдонимом структуры System.Int32. Отсюда следует, что int и Integer в этих языках — одно и то же.

Ниже приведена таблица некоторых простых типов данных — структур и их псевдонимов.

sbyte	System.SByte
byte	System.Byte
short	System.Int16
int	System.Int32
long	System.Int64
char	System.Char
float	System.Single
double	System.Double
bool	System.Boolean
decimal	System.Decimal

Объявление переменных

CTS позволяет работать со значениями двух видов: величины и ссылки.

Величины

Под величинами понимаются такие переменные, которые непосредственно хранят данные. Причем каждая переменная содержит свою копию данных и операция над одной переменной не может повлиять на другую переменную.

Рассмотрим пример:

```
using System;

 class VarTest
 {

      static void Main()
      {
         int nVar1=10;
         int nVar2=nVar1;
         nVar1=30;
         Console.WriteLin("Var1={0}, Var2={1}",nVar1,nVar2);
       }
  }
```

Результат работы: Var1=30, Var2=10

Как видно из примера выше, переменная nVar2 хранит свою копию данных, и операции над переменной nVar1 (мы изменили значение переменной) никак не влияют на nVar2.

Отметим, что переменные этого типа всегда содержат какое-либо значение, в отличие от ссылок, которые могут указывать в никуда (null).

Ссылки

Переменные, которые имеют тип ссылки, содержат только ссылку на данные, то есть в них хранятся указания о том, в каком месте находятся данные в памяти. При объявлении ссылки память под данные не выделяется, а выделяется лишь память под переменную, где будет храниться адрес возможных объектов. Для того чтобы выделить память, необходимо использовать оператор **new**. Компилятор берет на себя обязанность по проверке инициализации ссылок, и если вы попытаетесь использовать ссылку, предварительно не проинициализировав ее, компилятор выдаст ошибку.

Рис. 1.2.1

На одни и те же данные могут указывать несколько ссылок, причем если мы меняем данные по одной из этих ссылок, то естественно, что другая ссылка, указывающая на эти же данные, будет ссылаться уже на обновленную информацию.

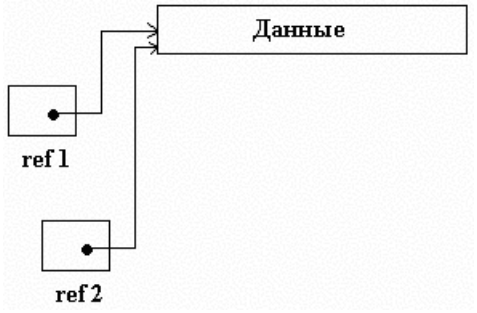

Рис. 1.2.2

Рассмотрим пример, который демонстрирует этот факт:

```
using System;

class Class1
{
     public int Value;
```

```
}

class RefTest
{
     static void Main()
     {
            Class1 ref1 = new Class1();
            ref1.Value=7;
            Class1 ref2 = ref1;
            ref2.Value = 123;
            Console.WriteLine(
              "Ref1={0}, Ref2={1}", ref1.Value, ref2.Value);
      }
 }
```

Инициализация

Для того чтобы объявить переменную, необходимо записать ее тип, а затем имя переменной. Этот синтаксис подобен синтаксису языков C++, Java.

```
int nVal; //объявление переменной целого типа
```

Допускается также определение нескольких переменных в одной строке через запятую, но это считается плохим стилем программирования, так как затрудняет читабельность кода и усложняет его модификацию.

```
int nVal1, nVal2; //объявление двух переменных целого типа
```

Мы можем инициализировать переменную при объявлении, то есть присвоить ей какое-то начальное значение.

Для инициализации переменных используют оператор присваивания «=».

```
int nVal1=0;
```

Если после объявления ваша переменная так нигде и не была проинициализирована, то это вызовет ошибку компиляции. Так, следующая программа работать не будет:

```
using System;

class InitTest
{
     static void Main()
     {
        int nVal1;
        Console.WriteLine(nVal1);          //ошибка компиляции
     }

}
```

Инициализировать переменные можно как во время объявления, так и после. Так, следующая программа отработает нормально:

```
using System;

class InitTest
```

```
{
     static void Main()
     {
        int nVal1;
        nVal1=0;
        Console.WriteLine(nVal1);
     }
}
```

Хорошим стилем программирования является инициализация переменной при ее объявлении.

Если вы инициализируете переменную типа char, то символ заключается в одинарные кавычки:

```
char sym='a';
```

Если вы инициализируете строковую переменную, то текст необходимо заключить в двойные кавычки:

```
string str="text";
```

При инициализации строки вы можете использовать специальные символы, такие как переход на другую строку и табуляция. Ниже приведены специальные обозначения для некоторых из них.

\' — одинарная кавычка;
\" — двойная кавычка;
\0 — null;
**** — косая черта;
\b — удаление символа;
\n — переход на другую строку;
\r — возврат каретки;
\t — горизонтальная табуляция;
\v — вертикальная табуляция.

```
using System;

 class StringTest
 {
    static void Main()
    {
       string temp="\'Hello \"world\"\'";
       Console.Write(temp);
    }
 }
```

В примере выше на экране будет отображен текст **"'Hello "world'"**.

Область видимости

При определении классов и методов используются открывающие и закрывающие фигурные скобки. Эти скобки используются во всех структурах, которые поддерживает C#. Содержимое между открывающей и закрывающей скобкой называется блоком. Блоки могут быть вложенными друг в друга. Любая переменная, объявленная внутри конкретного блока, называется локальной переменной для этого

блока, и она не существует вне него, то есть ее нельзя использовать в других блоках, если они не являются вложенными.

```
{
   {
       double a=3;
       . . . . . .
       {
            Console.WriteLine(a);     //все верно
       }
   }
   Console.WriteLine(a);          //ошибка компиляции
}
```

В отличие от C++, язык C# не допускает объявление переменной во вложенном блоке с тем же именем, что и переменная, объявленная в основном блоке. Код ниже вызовет ошибку компиляции.

```
{
     double a;
     . . . . .
     {
            double a;      //ошибка компиляции
            . . . . .
     }

 }
```

Однако компилятор позволяет объявлять одинаковые переменные в не пересекающихся между собой блоках.

```
{
     double a;
     . . . . .
 }
 . . . . .
 {
     double a;
     . . . . .
 }
```

Именование переменных

Как говорилось выше, каждая переменная имеет имя. Именно с помощью имени программист получает доступ к содержимому переменной и к возможности изменять ее значения. При выборе имени переменной необходимо придерживаться некоторых правил.

- Имя переменной должно начинаться с буквы либо знака подчеркивания:

```
int nVar;      //верно
int _Var;      //верно
int 1Var;      //ошибка, имя начинается с цифры
```

- Все символы, следующие за первым, могут быть буквами, цифрами или знаком подчеркивания:

```
double val13_my; //верно
```

- C# включает так называемые ключевые слова, которые он использует для построения конструкций языка, например для построения условий, циклов, выделения памяти. Переменная не может быть названа именем, совпадающим с ключевым словом.

```
double abstract; // ошибка
double abstract1; // верно
```

Список ключевых слов приведен ниже.

abstract	do	in	protected	true
as	double	int	public	try
base	else	interface	readonly	typeof
bool	enum	internal	ref	uint
break	event	is	return	ulong
byte	explicit	lock	sbyte	unchecked
case	extern	long	sealed	unsafe
catch	false	namespace	short	ushort
char	finally	new	sizeof	using
checked	fixed	null	stackalloc	virtual
class	float	object	static	void
const	for	operator	string	volatile
continue	foreach	out	struct	while
decimal	goto	override	switch	
default	if	params	this	
delegate	implicit	private	throw	

Кроме правил, существует также ряд рекомендаций, которые позволят вам избежать ошибок и сделать код удобным для чтения.

- Избегайте именовать переменные только маленькими или только большими буквами.
- Избегайте применять символ подчеркивания в качестве первого символа имени переменной.
- Избегайте аббревиатур.
- Используйте для определения переменных столько слов, сколько необходимо, чтобы описать имя переменной как можно более полно. Используйте несколько слов. Каждое слово, кроме первого, начинайте с большой буквы.

```
int countBookPerMounth;
```

Операторы

Любое выражение представляет собой набор операторов и операндов, которые после вычисления дают какое-то значение. Операторы реализуют операции над операндами. Рассмотрим некоторые общие операторы, которые встречаются в C#.

Назначение	Операторы
Проверка на равенство	== !=
Операторы сравнения	> < >= <= is
Операторы сдвига	>> <<
Условные операторы	&& || ?:
Икремент	++
Декремент	--
Арифметические операторы	+ - * / %
Операторы присваивания	= *= /= %= += -= <<= >>= &= ^= |=

Использование этих операторов не вызывает сложностей и аналогично использованию в других языках программирования, поэтому рассматриваться тут не будет.

Выше были представлены не все операторы, существуют еще целые классы операторов по приведению типов, выделению памяти и проверке переполнения. Все эти операторы будут рассматриваться ниже.

Пользовательские типы

Рассмотрим, как создавать более сложные типы и переменные этих типов.

Перечислимые типы

Перечислимые типы предназначены для создания переменных, которые могут принимать лишь значения из какого-то заранее заданного множества.

Для создания перечислимого типа используется ключевое слово enum, после которого идет имя типа и список значений, которые может принимать переменная этого типа.

```
enum Color {Red, Green, Blue}
```

Значения, которые перечисляются в этом типе, представляют собой константы целого типа int. По умолчанию, первое значение, которое присваивается константе, — это 0, то есть в нашем примере red=0, green=1, blue=2. При создании этого типа вы можете поменять нумерацию.

```
enum Color {Red=2, Green=30, Blue=10}
```

Объявление переменных перечислимого типа такое, как и для обычных типов.

```
enum Color {Red, Green, Blue}
Color carColor;
carColor=Color.Red;
```

В примере выше мы создали переменную перечислимого типа и присвоили ей значение 0, которое соответствует константе red. Этот код аналогичен следующему:

```
enum Color {Red, Green, Blue}
```

```
Color carColor;
carColor=(Color)0;
```

Этот тип позволяет пользователю не запоминать значение констант, а просто использовать их имена. Например, для создания объекта **FileStream** требуется указать режим, в котором поток будет открыт и который будет определяться некоторой константой. Но для простоты использования все эти константы были помещены в перечислимый тип **FileMode**

```
public enum FileMode
{
  Append,
  Create,
  CreateNew,
  Open,
  OpenOrCreate,
  Truncate
}
```

Таким образом, для создания объекта **FileStream** достаточно указать имя константы. Например, если мы хотим открыть файл для записи в конец, то указывается значение **FileMode.Append**.

Если ваша функция принимает значение перечислимого типа, то можно проверить, входит ли передаваемая константа в ваш тип на тот случай, если программист передаст целую константу, которая не входит в допустимые значения.

```
public void SetColor(Color color)
{
     if (!Enum.IsDefined (typeof(Color), color)
          throw new ArgumentOutOfRangeException();
}
```

Метод **IsDefined** класса **Enum** проверяет, входит ли в тип **Color** (тип указывается в качестве первого аргумента методу **IsDefine**) значение, которое передается методу в качестве параметра. В случае, если константа не определена в типе **Color**, программа выбрасывает исключение.

Структуры

Вы можете объединять переменные различных простых типов в группы, тем самым создавая собственный сложный тип, состоящий из набора простых значений. Такой пользовательский тип носит название структуры.

Ниже приведен пример определения структуры.

```
public struct People
{
     public string firstName;
     public string lastName;
     public int age;
}
```

В этом примере определяется новый тип данных, который называется People. Этот тип содержит три элемента, представляющих из себя переменные элементарных типов. Первые два элемента являются строковыми переменными и предназ-

начены для хранения имени и фамилии человека, а третий элемент представляет собой переменную целого типа для хранения возраста.

Объявление структуры очень похоже на объявление **enum**, но элементами структуры могут быть переменные любых типов, кроме того, элементы структуры не константы.

Рассмотрим, как ведется работа с типом People. Ниже приводится часть кода, демонстрирующая, каким образом определяется переменная структурного типа, инициализируется и используется.

```
using System;

class StructTest
{
  static void Main()
  {
    People worker;
    worker.firstName=Console.ReadLine();
    worker.lastName=Console.ReadLine();
    worker.age=24;
    Console.WriteLine("Your name:{0} {1}", worker.firstName,
         worker.lastname);
    Console.WriteLine("Your age: {0}", worker.age);
  }
}
```

Приведение типов

В C# существуют два варианта приведения типов: явное приведение и неявное.

Неявное приведение

Как и в других языках программирования, в C# существует неявное приведение типов. Так, например, если вы используете выражение, которое имеет тип int, а результат его вы хотите поместить в переменную типа double, то программа автоматически расширит вашу переменную до необходимого размера, причем сами данные не будут потеряны:

```
using System;

class ImpConvTest
{
   static void Main()
   {
      int i=5;
      double a;
      a=i;          // верно a=5.0
      Console.WriteLine("a={0}",a);
   }
}
```

Явное приведение

В некоторых случаях вы можете попытаться присваивать переменной большего размера переменную, имеющую тип, занимающий меньший размер. Например, присвоить переменной типа int переменную типа double. В этом случае компилятор выдаст ошибку, если вы явно не укажете тип, к которому вы хотите привести вашу переменную или выражение. Пример ниже работать не будет, так как неявное приведение типа double к типу int не может быть выполнено.

```
using System

class ConvertTest
{
     static void Main()
     {
         double nVar1=6.489;
         int nVar2=nVar1;
         Console.WriteLine("nVar2={0}",nVar2);
     }
 }
```

Однако существует также явное приведение типов. Пример ниже работать будет, но в этом случае вы не застрахованы от потери данных.

```
using System
class ConvertTest
{
     static void Main()
     {
          double nVar1=6.489;
          int nVar2=(int)nVar1;
          Console.WriteLine("nVar2={0}",nVar2);
     }
}
```

Результатом работы этой программы будет число 6. Таким образом, часть данных потеряна. Можно использовать оператор **checked** для проверки целостности данных в переменных, тогда в случае утери информации будет выбрасываться исключение.

```
using System

class ConvertTest
{
    static void Main()
    {
         double nVar1=6.489;
         checked
         {
              int nVar2=(int)nVar1;
              Console.WriteLine("nVar2={0}",nVar2);
         }
    }
}
```

Глава 3
Основные структуры языка

Независимо от выбора языка программирования, в любом из них вы сможете встретить специальные структуры, реализуемые операторами, которые имеют сходный вид во всех языках программирования. Это такие структуры, как структура выбора и структура повторения (цикла). Доказано, что этих структур достаточно, чтобы написать любую программу или реализовать любой алгоритм. Поэтому, прочитав эту главу, можно будет с гордостью сказать, что вы можете написать на C# любую программу — ☺. Во всех языках программирования операторы, реализующие эти структуры, в чем-то сходны между собой синтаксически, но вместе с тем есть и различия. Рассмотрим, как реализованы эти структуры в C#.

Структуры выбора

C# поддерживает две структуры выбора, это конструкция **if...** и конструкция **switch**. Благодаря этим двум операторам можно реализовывать ветвление в ваших программах.

Условие if...

Рассмотрим синтаксис конструкции **if...**.

```
 if (выражение)
{
  . . . . . .
  //блок1
}
else
{
  . . . . . .
  //блок 2
}
```

Выражение в скобках возвращает значение либо истина, либо ложь (**true**, **false**), то есть значение типа **bool**. Если выражение истинно, то выполняются операторы, которые стоят в блоке 1, если выражение ложно, то выполняются операторы, стоящие в блоке 2.

Оператор **if** можно использовать и без оператора **else**:

```
if (выражение)
{
  . . . . . .
  //блок1
}
```

В этом случае выполняются операторы, стоящие в блоке 1, если выражение истинно; если выражение ложно, то ни один из операторов блока if не выполняется.

Рассмотрим пример.

```
using System;

class CompareString
{
    public static Main()
    {
         Console.WriteLine("Enter first string:");
         string firstString=Conole.ReadLine();
         Console.WriteLine("Enter second string:");
         string secondString=Console.ReadLine();

         if (firstString.Equals(secondString))
         {
              Console.WriteLine(
                 "First string is equal to second string");
         }
         else
         {
              Console.WriteLine(
                 "First string is not equal to second string");
         }
    }
}
```

Пример, который приведен выше, демонстрирует работу оператора **if**. Пользователь вводит с клавиатуры 2 строки, и если они равны, то пользователю выводится сообщение о равенстве этих строк, в противном случае выводится сообщение о неравенстве строк.

В случае если в блоке **if** всего одна строка кода, то вы можете не указывать фигурные скобки, хотя это и является плохим стилем программирования. Так, два примера ниже эквивалентны:

```
//пример 1
if (x==0)
{
    x=x+10;
}

//пример 2
if (x==0)
    x=x+10;
```

Если необходимо сделать выбор между более чем двумя вариантами, то существует более сложная версия условия **if** с множественным выбором. Для построения таких конструкций используется выражение **else if**.

```
if (x<0)
{
    x=x+10;
```

```
}
else if (x>10)
{
    x=x-10;
}
else
{
    x=5;
}
```

В этом примере происходит проверка условия в первом **if**, и если условие истинно, то выполняется код x = x + 10, после чего осуществляется переход на оператор, следующий за всем блоком **if**. В случае если первое условие ложно, проверяется условие во втором блоке **if**, и если оно возвращает значение **true**, то выполняется код x = x – 10. Если ни одно из условий не вернуло значение **true**, то переход осуществляется в блок **else**. Важным является то, что, после выполнения операторов в одном из блоков оператора **if**, программа переходит сразу на оператор, следующий за **if**, даже если условия в нижестоящих блоках этого оператора истинны.

Структуры **if** могут быть вложены друг в друга:

```
if (x<10)
{
    if (y<5)
    {
        y=y+x;
    }
    else
    {
        y=y-x;
    }
}
```

Отметим, что в отличие от языка C++, тип **int** не приводится автоматически к типу **bool**, поэтому мы не можем использовать конструкцию, допустимую в C++:

```
int i=0;
. . . . . .
if (i)        //ошибка
{
    . . . . . .
}
```

Таким образом, необходимо использовать выражения, возвращающие значение типа bool, или преобразовывать к этому типу явно.

В примерах выше мы использовали только простые выражения в виде условий, на самом деле чаще приходится комбинировать несколько выражений с помощью логических операторов.

Предположим, что у нас есть два выражения — X и Y, которые возвращают значения типа **bool**. Ниже приведена таблица (таблица истинности), которая показывает, как будет происходить вычисление, если мы начнем комбинировать эти выражения с помощью логических операций.

Содержание

Часть III. ADO.NET