Проектиране и разработка на Windows базирани приложения (част I)

Лекции 2 ч.

Лабораторни упражнения 1 ч.

Изпит

Лектор: Доц. д-р Мариана Горанова
Катедра „Програмиране и компютърни технологии“, ФКСУ
Технически университет – София
Кабинет: 2304
Електронен адрес: mgor@tu-sofia.bg

Литература

I. Основна литература
1. Tom Archer, Andrew Whitechapel, Inside C#, Second Edition, Microsoft Press, 2002.
Том Арчър, C# – поглед отвътре, СофтПрес, 2001.
2. John Sharp, Jon Jagger, Microsoft Visual C# .NET Step by Step, Microsoft Press, 2002.
3. Charles Petzold, Programming Microsoft WINDOWS with C#, Microsoft Press, 2002.
Чарлз Петцолд, Windows на C#, том I и том II, СофтПрес, ООД, 2003.

II Допълнителна литература
1. Джефри Рихтер, Microsoft .NET Framework – приложно програмиране, СофтПрес ООД, 2002.
2. Damien Watkins, Mark Hammond, Brad Abrams, Programming in the .NET Environment, Microsoft Corporation, 2003.
3. Judith Bishop, Nigel Horspool, C# Concisely, Pearson Education Limited, 2004.

III. Лабораторни упражнения
1. М. Горанова, В. Димитрова, Д. Гоцева, Ръководство по програмиране на C#, ТУ – София, 2006.

Въведение в Windows програмирането

- 1985 г. – Microsoft създава първа версия на Windows
- Преход от 16-битова в 32-битова архитектура
1993 г. – Windows NT
1995 г. – Windows 95

Подходи при проектиране на Windows приложения, използвайки C-базиран език

Година	Език	Интерфейс
1985	C	API (Windows Application Programming Interface)
1992	C++	MFC (Microsoft Foundation Class Library)
2001	C# или C++	Windows Forms (част от .NET Framework)

Въведение в езика за програмиране C#

Anders Hejlsberg (Delphi, Java Foundation classes)
Scott Wiltamuth (Microsoft)
Peter Golde (Microsoft)

C# – стандарт
ECMA (European Computer Manufacturer’s Association)
ISO 2003

Цели

1. Развит за .NET Framework
2. Удобен
3. Съответства на .NET CLR (Common Language Runtime) – обща среда за изпълнение на различни езици.
4. Опростява модела на C++
   а. липсва заглавен файл и препроцесор;
   б. липсва управление на паметта чрез система от указатели и събиране на боклука – използва система от референтни типове.
5. Гъвкав
6. Поддържа разработка на компоненти

Характеристики

Много близък до Java

70% Java, 10% C++, 5% Visual Basic, 15% ново

Както в Java:	Както в C++:
Обектно-ориентиран (единично наследяване)	Презареждане на оператори
Интерфейси	Аритметика с указатели в несигурен код
Изключения	Някои детайли от синтаксиса
Нишки	
Наименовани пространства (като пакетите)	
Силно типизиран	
Събиране на боклука	
Динамично зареждане на кода	

Нови характеристики: (спрямо Java)

Референтни и изходни параметри	Компонентно-базирано програмиране • свойства • събития
Обекти, разположени в стека (struct)	Делегати
Правоъгълни масиви	Индексатори
Изброим тип (enum)	Презареждане на оператори
Унифицирана система на типовете	Оператор foreach
Оператор goto	Опаковане/разопаковане
Създаване на версия	Атрибути

Основи на платформата .NET

Платформа .NET – стратегия на Microsoft за разработка на големи разпределени софтуерни системи.

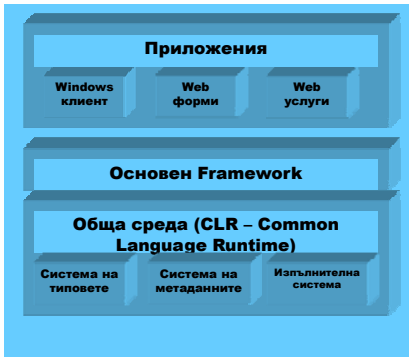
.NET Framework – компонентен модел за Интернет – отделни софтуерни компоненти, написани на различни езици, се комбинират във функциониращa система.

Различия на .NET Framework от:

1. COM (Component Object Model) на Microsoft – компонентен модел за десктоп (не за разпределени системи).
2. CORBA (Common Object Request Broker Architecture) – програмен модел за Интернет с обектно-ориентирана архитектура за разпределени системи – няма компонентна архитектура. CORBA3 е с компонентна архитектура.

Сравнява се с Java – програмен език за Интернет с характеристики на COM, CORBA и .NET но за единствен програмен език.

Архитектура на .NET Framework



Предимства на .NET Framework

1. **Едни и същи концепции и услуги за всички типове приложения.**
2. **Многократно използване на компонентите.**
3. **Поддръжка на много програмни езици.**

Езици в .NET Framework

1. **Visual Basic .NET**
2. **Visual C++ .NET**
3. **C#**
4. **Python за .NET**
5. **Perl за .NET**
6. **Компонентен Pascal за CLR**
7. **HotDog Scheme**
8. **Mondrian**
9. **Active Oberon за .NET**
10. **J#**

Структура на програма на C#

```
using <име_на_пространство>;
namespace <име_на_потребителско_пространство>
{
   class <име_на_потребителски_клас>
   {
      static void Main (string[] args)
      {
         // тяло
      }
   }
}
```

namespace – **пространство от имена**

Пълно име на клас:
<име_на_пространство>.<име_на_клас>

using – **директива – път за търсене на** <име_на_пространство>
(не се прилага за <име_на_клас>**)**

Вход

1. **Конзолно приложение – клавиатура**
2. **Windows приложение – вход, управляван от събития**
   - **всеки вид вход е свързан с подходящ метод в клас;**
   - **събитие – член на класа;**
   - **метод за обработка на събитие – манипулатор на събитие с аргументи, съответстващи на прототипа на функция, наречен делегат;**
   - **събитие** Paint – **изобразява изхода върху прозореца.**

Пример: Конзолно приложение

```
using System;
class Welcome
{
    public static void Main()
    {
        Console.WriteLine("Добре дошли!");
        Console.WriteLine("Днес е " +DateTime.Now);
    }
}
```

System – **пространство от имена**

Клас System.Console – **представя стандартните потоци за конзолни приложения.**

Метод Console.WriteLine – **извежда данните в стандартния изходен поток и добавя край на ред.**

Форматиране на изхода:

Console.WriteLine("{N[,M][:S]}", аргумент0, …, аргументN);

N – **номер на позицията на аргумента в списъка от стойности (начална позиция** 0**);**

M – **(незадължителна) ширина и подравняване с добавени празни позиции:**
- M<0 **или липсва – подравняване от ляво;**
- M>0 – **подравняване от дясно;**

S – **(незадължителен) форматиращ низ – при липса съответният метод** ToString **определя форматирането:**

Xm

X – **форматен спецификатор;** m – **точност;**
- C/c **парична единица**
- D/d **десетично цяло число**
- E/e **експоненциално реално число**
- F/f **реално число с фиксирана точка**
- G/g **основно представяне**
- N/n **числено представяне подобно на** F **със запетая за хилядите**
- P/p **процент**
- R/r **закръглено (само за плаваща точка) – гарантира коректно преобразуване**
- X/x **шестнадесетично**

Потребителски форматни спецификатори:

0	**допълва с незначещи нули**
#	**изобразява само значещи цифри**
.	**място на десетичния разделител**
,	**разделител за хилядите**
%	**изобразява %**
E+0 E-0 e+0 e-0	**експоненциално представяне**
\	**за форматиращи последователности (нов ред** \n**)**
'ABC' "ABC"	**изобразява низ в литерали или кавички**
;	**разделител на секция – различно форматиране в зависимост от това дали стойността е >0, <0 или 0**
аргументN	**израз; ако е** null**, се използва празен низ.**

Структура DateTime – **представя времето (дата и час).**

Свойства

Now	**(статично) връща текущата дата и време;**
Date	**връща датата;**
TimeOfDay	**връща времето;**
Today	**(статично) връща текущата дата.**

Форматиращи символи:

D	LongDatePattern	dddd, mmmm dd, yyyy
d	ShortDatePattern	mm/dd/yyyy
T	LongTimePattern	hh:mm:ss
t	ShortTimePattern	HH:mm
m, M	MonthDayPattern	mmmm dd

Пример:

```
DateTime dt=DateTime.Now;
Console.WriteLine(dt);
Console.WriteLine
        ("Дата={0:d}, време={1:T}, днес е {2:m}",
        dt.Date, dt.TimeOfDay, DateTime.Today);
```

Резултати:

```
03.2.2004 г. 13:32:46
Дата=03.2.2004 г., време=13:32:46.1234567, днес е 11 Март
```

Пример:

```
using System;
class TestWriteLine
{  public static void Main(string[] args)
   {    Console.WriteLine("{0,5},{1:D5}", 123, 456);
        // □□123,00456
        Console.WriteLine("{0,-10:D6},{1,-10:D6}", 123, 456);
        // 000123□□□□,000456□□□□
        Console.WriteLine("{0,-10}{1,-8}", "Име", "Фак.N");
        // Име□□□□□□□Фак.N□□□
        Console.WriteLine("----------");
        // - - - - - - - - - -
        Console.WriteLine("{0,-10}{1,8}", "Иван", 123456);
        // Иван□□□□□□□123456
        Console.WriteLine("{0,-10}{1,8}", "Гергана", 7890);
        // Гергана□□□□□□7890
        Console.WriteLine("{0:C}{1,5:F2}", 7890, 5.6);
        // 7□890,00□лв□5,60
```

```
        float f=-123456.7890F;
        Console.WriteLine("{0:$#,##0.00;($#,##0.00);Zero}", f);
        // ($123□456,80)
        int i=1234567890;
        Console.WriteLine("{0:(###) ### - ####}", i);
        // (123) □456□-□7890
        Console.WriteLine("{0:#%}", i);
        // 123456789000%
    }
}
```

Пример: Windows приложение

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace Test
{
  public class Welcome
  {
    public static void Main()
    {
      Form form = new Form();
      form.Text = "Windows програмиране";
      form.Paint += new PaintEventHandler(MyPaintHandler);
      Application.Run(form);
    }
```

Пространства от имена

Прозорец (форма)

Текст в заглавния ред

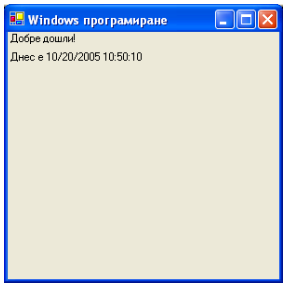
Добавяне на манипулатор на събитието Paint

Изпълнение на приложението

```
    static void MyPaintHandler(object sender,
                               PaintEventArgs pea)
    {
        Form form = (Form)sender;
        Graphics grfx = pea.Graphics;
        grfx.DrawString("Добре дошли!", form.Font,
                        Brushes.Black, 0, 0);
        grfx.DrawString("Днес е " + DateTime.Now, form.Font,
                        Brushes.Black, 0, 20);
    }
  }
}
```

Манипулатор на събитието Paint

Изобразява текст



Създаване на конзолно приложение във Visual Studio .NET

1. **Стартиране на Visual Studio .NET.**

2. **File ⇒ New ⇒ Project**
   Project Type ⇒ Visual C# Project
   Templates ⇒ Console Application
   Location ⇒ директория на проекта
   Name ⇒ име на проекта

3. **View ⇒ Solution Explorer**
   име_на_приложение.sln – **файл на решението (един за приложение с няколко проекта);**
   име_на_проект.csproj – **C# проектен файл (няколко сорс файла на един и същ програмен език);**
   име_на_клас.cs – **C# сорс файл;**
   AssemblyInfo.cs – **C# сорс файл за добавяне на атрибути към програмата;**
   App.ico – **икона на приложението.**

4. **Въвеждане кода на програмата**

5. **Изграждане и изпълнение на конзолно приложение**
   Build ⇒ Build
   Debug ⇒ Start Without Debugging

Създаване на конзолно приложение от команден режим

C# компилатор
csc.exe
Установяване на пътя до изпълнимия файл csc.exe
VCVARS32.bat
(напр. C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\bin\vcvars32.bat**)**

1. Компилация
csc приложение.cs

2. Изпълнение
приложение

Асембли – фундаментална част в .NET;
- **създава се от компилатора;**
- **съдържа код, изпълняващ се от системата;**
- **колекция от модули, експортирани типове и ресурси, която има име и версия (от гледна точка на клиента);**
- **начин за пакетиране на модули, типове и ресурси, използвани от клиента (от гледна точка на създателя).**

Манифест – съдържа информация за мета данните; включва три записа:
- .assembly **за обръщение към външно асембли;**
- .assembly **с информация за асемблито;**
- .module**, съдържащ името на физическия файл и друга външна информация.**

Деасемблиране на изпълним файл – приложение ILDASM

Start ⇒ Programs ⇒ Microsoft Visual Studio .NET 2003 ⇒ Visual Studio .NET Tools ⇒ Visual Studio .NET Command prompt ⇒ команден прозорец : ildasm

прозорец на ILDASM : File ⇒ Open ⇒
драйв:\...\директория_на_приложение\bin\Debug\приложение.exe

Основни входно-изходни операции

1. **Входни операции**
   Метод Console.ReadLine **– въвежда низ от стандартния входен поток.**
   Метод Console.Parse **– преобразува символен низ в друг тип стойност.**

```
низова_променлива  = Console.ReadLine();
друга_променлива   = тип.Parse(Console.ReadLine());
```

2. **Изходни операции**
   Методи Console.WriteLine **и** Console.Write**.**

```
Cosole.WriteLine (данни);
Console.Write (данни);
```

Пример:

```
using System;
class CalculateFee
{
  static void Main(string[] args)
  {
        float dailyRate, fee;
        int noOfDays;
        Console.Write("Въведете дневна ставка: ");
        dailyRate = float.Parse(Console.ReadLine());
        Console.Write("Въведете брой работни дни: ");
        noOfDays = int.Parse(Console.ReadLine());
        fee = dailyRate*noOfDays;
        Console.WriteLine("Заплата: {0:C}", fee);
  }
}
```

Пример: **Числено преобразуване на низ**

```
int j = int.Parse(" 123456 ");
Console.WriteLine("j={0}", j);                    // j=123456
float f = float.Parse("-123,456");
Console.WriteLine("f={0}", f);                    // f=-123,456
string s = j.ToString();
Console.WriteLine("s={0}", s);                    // s=123456
```

Създаване на документация чрез XML (Extensible Markup Language)

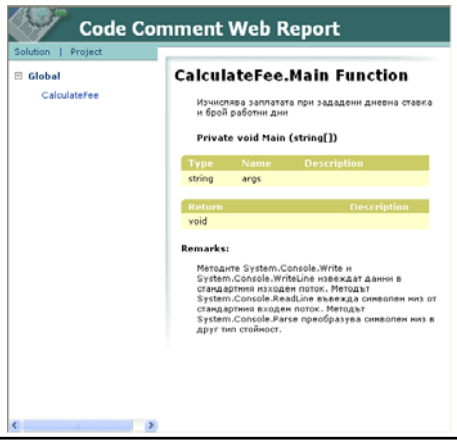
<summary>	**кратко резюме от един ред за клас, метод или свойство;**
<remarks>	**по-дълга и детайлна информация;**
<value>	**описание на свойство;**
<exception>	**изключения, вдигнати от методи и свойства;**
<param>	**параметри на метод.**

1. **Tools ⇒ Build Comment Web Pages**
2. **OK**
   Създава се структурирана последователност с хипервръзки от HTML документи, основани на XML.
3. **Solution Explorer ⇒ Show All Files**
   Появява се директорията CodeCommentReport с HTML файлове.

Пример:

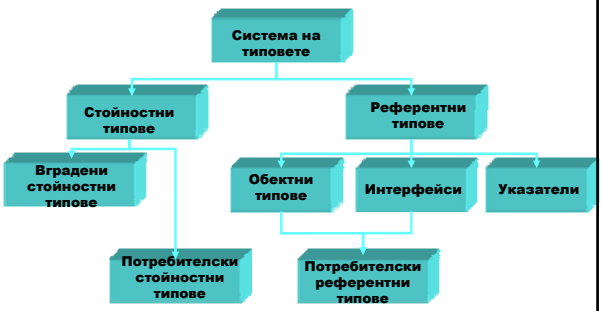
```
using System;
class CalculateFee
{
    /// <summary>
    /// Изчислява заплатата при зададени дневна ставка и
    /// брой работни дни.
    /// </summary>
    /// <remarks>
    /// Методите System.Console.Write и
    /// System.Console.WriteLine извеждат данни в
    /// стандартния изходен поток.
    /// Методът System.Console.ReadLine въвежда
    /// символен низ от стандартния входен поток.
    /// Методът System.Console.Parse преобразува
    /// символен низ в друг тип стойност.
    /// </remarks>
```

```
    static void Main(string[] args)
    {
      float dailyRate, fee;
      int noOfDays;
      Console.Write("Въведете дневна ставка: ");
      dailyRate = float.Parse(Console.ReadLine());
      Console.Write("Въведете брой работни дни: ");
      noOfDays = int.Parse(Console.ReadLine());
      fee = dailyRate*noOfDays;
      Console.WriteLine("Заплата: {0:C}", fee);
    }
}
```



Типове данни

Обща система на типовете



Типове данни

I. **Стойностни типове – примитивни типове, структури и изброими типове**
   1. **Съдържат действителните данни – директен достъп.**
   2. **Съхраняват се в стека.**
   3. **Не може да имат стойност** null**.**
   4. **Предава се стойността на променливата като параметър – променливата не се модифицира.**

II. **Референтни типове – класове, масиви, интерфейси и делегати**
   1. **Съдържат адреса на обект от определен тип – косвен достъп чрез указател към обекта.**
   2. **Съхраняват се в хийпа.**
   3. **Може да имат стойност** null**.**
   4. **Предава се адресът на обекта като параметър – обектът се модифицира.**

Идентификатори – имена на елементите в програмата.
- **букви (малки и главни) и цифри;**
- **започват с буква (долна черта _ е буква);**
- **разлика между главни и малки букви.**

result _score twentyOne plan9 TwentyOne

Променливи – място за съхранение на стойност. Чрез името се осъществява достъп до съдържащата се стойност.

Правила за имена на променливи в .NET Framework
- **да започват с малка буква;**
- **всяка следваща дума да започва с главна буква;**

  twentyOne
- **да не се използват долни черти;**
- **да не се различават идентификаторите само по това дали буквите са главни или малки.**

  myVariable **и** MyVariable

Примитивни типове данни

Тип	Размер [бит]	Област	Примери
byte	8	0÷255	byte b=42;
short	16	-2^{16}÷2^{16}-1	short s=42;
int	32	-2^{31}÷2^{31}-1	int count=42;
long	64	-2^{63}÷2^{63}-1	long wait=42L;
float	32	±3.4x10^{38}	float away=0.42F;
double	64	±1.7x10^{308}	double trouble=0.42;
decimal	128	**28 значещи цифри**	decimal coin=0.42M; (за финансови изчисления)
string	16/символ	не се използва	string vest="42";
char	16	0÷2^{16}-1	char grill='4';
bool	8	true или false	bool flag=false;
object	базов тип за всички типове		

2^{16}=32768 2^{31}=2147483648 2^{63}=9223372036854775808

Деклариране на променливи

тип идентификатор;

Оператор за присвояване =

идентификатор = израз;

Проверка за препълване при аритметични операции и преобразувания checked/unchecked

checked – **аритметичното препълване предизвиква изключение**

unchecked – **аритметичното препълване се игнорира и резултатът се отрязва**

checked блок
checked (израз)
unchecked блок
unchecked (израз)

Преобразуване на типовете

1. **Неявно преобразуване – към следващия по-висш тип.**
2. **Явно (принудително – cast) преобразуване**

   (тип) израз
   checked ((тип) израз)

   При включена проверка checked **се получава** OverflowException**, ако** типът **не може да побере резултантната стойност.**

Опаковане (boxing) и разопаковане (unboxing)

1. **Опаковане – преобразуване на стойностен тип в референтен тип.**

```
int i = 42;              // стойностен тип
object bar = i;          // i е опакован за bar
```

2. **Разопаковане – преобразуване на референтен тип в стойностен тип – чрез принудително преобразуване на типа.**

```
int i = 42;              // стойностен тип
object bar = i;          // i е опакован за bar
int baz = (int)bar;      // обратно разопаковане в int
```

Изрази и оператори

Оператор

1. **Символ за операция върху операнди.**
2. **Резултат**
   - **нова стойност от операцията върху операндите;**
   - **запазва се в паметта в променлива.**
3. **Видове – според броя на операндите:**
   - **унарни;**
   - **бинарни;**
   - **тринарни.**

Приоритет и асоциативност

1. **Ред на изпълнение на операторите – според приоритета.**
2. **Изчисление на ляв или десен операнд на израз – според асоциативността.**

Приоритет на операциите

Група на оператора	Оператори	Асоциа-тивност
Първични	(x) x.y f(x) a[x] x++ x-- new typeof sizeof checked unchecked	дясна
Унарни	+ - ! ~ ++x –x (T)x	дясна
Мултипликативни	* / &	лява
Адитивни	+ -	лява
Преместване	<< >>	лява
Отношение	< > <= >= is as	лява
Равенство	== !=	лява
Логически AND	&	лява
Логически XOR	^	лява
Логически OR	|	лява
Условен AND	&&	лява
Условен OR	||	лява
Условен	?:	лява
Присвояване	= += -= *= /= %= >>= <<= &= ^= !=	дясна
Запетая	,	лява

Управляващи структури

Категории:

1. **Операции за разклонения**
2. **Операции за цикли**
3. **Операции за преход**

Операции за разклонения

Оператор if

```
if (израз)
        оператор1
[else
        оператор2]
```

израз – **тип** bool

Оператор if-else-if

```
if (израз1)
        оператор1
else if (израз2)
        оператор2
else
        оператор3
```

Оператор switch

```
switch (израз)
{       case константен_израз1:
                оператор1
                оператор_за_преход
        ...
        case константен_изразN:
                операторN
                оператор_за_преход
        [default
                операторN+1
                оператор_за_преход]
}
```

израз – **тип** byte**,** short**,** int**,** long**,** char**,** string**;**
оператор_за_преход **(напр.** break**) – за всеки оператор** case**.**

Оператори за цикли

Оператор while

```
while (логически_израз)
        оператор
```

Оператор do-while

```
do
        оператор
while (логически_израз);
```

Оператор for

```
for (инициализация; логически_израз; актуализация)
        оператор
```

Оператор foreach

```
foreach (тип идентификатор in израз)
        оператор
```

```
string s = "Programming in the .NET Environment";
int count = 0;
foreach (char c in s)
        if (c>='A' && c<='Z')
                count++;
Console.WriteLine("Броят на главните букви е " + count);
```

Оператори за преход

Оператор break

```
break;
```

Оператор continue

```
continue;
```

Оператор goto

```
goto идентификатор
идентификатор: оператор
goto case константен_израз
goto default
```

Оператор return

```
return [върнат_израз]
```