ВЪВЕДЕНИЕ В ОБЕКТНО-ОРИЕНТИРАНОТО ПРОГРАМИРАНЕ. КЛАСОВЕ И ОБЕКТИ

Лекция 1

СЪДЪРЖАНИЕ

- Цел на програмирането.
- Обектно-ориентирано програмиране.
- Класове и обекти.
- Създаване и освобождаване на обекти.
- Извикване на методи на обект.
- Примери за системни C# класове.
- Пространствата от имена в C#.

ЦЕЛ НА ПРОГРАМИРАНЕТО

- Целта на всяка една програма, която създаваме е да **реши даден проблем** или да **реализира някаква идея**

1) Създаваме опростен модел на реалността, който не отразява всички факти от нея
2) Фокусираме се само върху тези факти, които **имат значение за намирането на решение** на нашата задача
3) Използвайки модела, създаваме алгоритъм за решение на проблема
4) Описваме решението чрез средствата на даден език за програмиране

ОБЕКТНО-ОРИЕНТИРАНО ПРОГРАМИРАНЕ

- **Обектно-ориентираното програмиране (ООП)** е **парадигма в компютърното програмиране**, при която една програмна система се моделира като **набор от обекти, които взаимодействат помежду си**
- ООП е близко до начина на мислене на човека
  - лесен за описание на модели
  - създава опростен модел на предметната област
  - предоставя средство, за описание на съвкупността от понятия, които описват обектите във всеки модел - клас (class)

ООП - ХАРАКТЕРИСТИКИ

- **Фундаментална концепция на ООП** – предаване на съобщения между обектите
- Всеки **обект е способен да:**
  - **получава съобщения**
  - **обработва данни**
  - **изпраща съобщения** на други обекти

- **Процедурно програмиране** - програмата е списък от инструкции, които се изпълняват последователно

ООП - ПРИМЕРИ

- Всяка програма работи с данни, описващи **предмети и явления** от **реалния живот**
- Например:
  - **счетоводна програма** - работи с фактури, стоки, складове, наличности, продажби и т.н.
  - **банкова система**
  - **самолет** – модел, тегло, вместимост и т.н.
  - **студент** – име, специалност, успех, факултетен номер и др.
  - ….

КАКВО Е ОБЕКТ?

- **Софтуерните обекти** моделират обекти от реалния свят или абстрактни концепции
- **Обект** – описва **характеристиките** (**свойства**) и **поведението** (**методи**) на обекти от реалния живот
  - **реални обекти** - хора, автомобили, стоки, покупки и т.н.
  - **абстрактните обекти -** понятия в някоя предметна област, които се налага да моделираме и използваме в компютърна програма
    - структурите от данни - стек, опашка, списък и дърво и др.

ХАРАКТЕРИСТИКИ НА ОБЕКТИТЕ

- **Състояния** (states) – това **са физически характеристики на обекта**, които по някакъв начин го определят и описват по принцип или в конкретен момент
- **Поведения** (behaviors) – това са **функционални характеристики** или специфични **характерни действия**, които обектът може да извършва

ОБЕКТИ ОТ РЕАЛНИЯ СВЯТ

- Самолет
  - **състояния:** модел, тегло, вместимост
  - **поведения:** излитане, кацане, дестинация
- Студент
  - **състояния:** име, факултетен номер, факултет, специалност, успех по дисц., среден успех ...
  - **поведения:** посещение на лекции, самообучение, явяване на изпит, получаване на стипендия ...
- Хищник
  - **състояния:** име, цвят на козината, порода
  - **поведения:** лаене, седене, ходене
- Университет, факултет ...
- Банка, финансова сфера ...
- ...

ОБЕКТИТЕ СЪДЪРЖАТ

- **Членове-данни** (data members) – представляват **променливи**, вградени в обектите, които **описват състоянията им**
- **Методи** (methods) – описват **поведението на обектите**

КАКВО Е КЛАС?

- **Клас** (**class**) в ООП наричаме **описание (модел) на реални предмети или явления**
- **Клас** - множество от **обекти със сходни характеристики на поведение и състояние**
- Примери
  - клас **Dog**
  - клас **Student**
  - клас **Rectangle**
  - **клас Lecture**
  - **...**

ОБЕКТИТЕ – ИНСТАНЦИИ НА КЛАСОВЕТЕ

- **Инстанциране** (instantiation) - създаване на обект от вече дефиниран клас
- **Инстанция** (instance) е **фактическият обект**, който **се създава** от класа по **време на изпълнение** на програмата
- Всеки обект е **инстанция** на конкретен клас
- Тази инстанция се характеризира със **състояние** (state) – **множество от стойности**, асоциирани с **атрибутите на класа**

ХАРАКТЕРИСТИКИ НА ОБЕКТ

- **Обектът се състои от две части:** моментно състояние и поведение, дефинирано в класа на обекта
- **състоянието** е специфично за инстанцията (обекта)
- **поведението** е общо за всички обекти, които са представители на този клас

КЛАСОВЕ В C#

- Дефинират се чрез ключовата дума **class**, последвана от идентификатор (име) на класа и съвкупност от членове-данни и методи, обособени в собствен блок код

```
class Child
{
  private int age;
  private string name;
   // Printing method:
  public void PrintChild()
  {
   Console.WriteLine("{0}, {1} years old.", name, age);
  }
}
```

- **Например:** класът **"Студент“**

```
class Program
{
   public class Student
    {  // Field definition
      private int number;
      private string name;
      protected string faculty;
       // Property definition
     private string Name (get; set; )
       // Methods definition
      public void ShowStud()
      {
       Console.WriteLine(number + "    " + name + "    " +
       faculty); }
      }
     static void Main(string[] args)
    {
      Student first = new Student(12301, "Maria Petrova", "FMI");
      Student second = new Student(12302, “Ivan Georgiev", "FMI");
      first.ShowStud();
      second.ShowStud();
}
```

СЪЗДАВАНЕ И ОСВОБОЖДАВАНЕ НА ОБЕКТИ

- Създаването на обекти от предварително дефинирани класове става чрез оператора **new**

```
Student first = new Student();
Student first = new Student(12301, "Maria Petrova", "FMI");
```

- Обектите, към които в даден момент вече няма референция в програмата, **автоматично се унищожават и паметта, която заемат се освобождава**

ДОСТЪП ДО ПОЛЕТА И МЕТОДИ НА ОБЕКТ

- Извикването на методите на даден обект става отново чрез оператора . (точка)

```
first.GiveNumber = 13001;
first.GiveName = "Ivan Atanasov";
first.GiveFaculty = "FMI";
first.ShowStud();
```

СТАНДАРТНИ БИБЛИОТЕКИ НА .NET FRAMEWORK

.NET Framework включва обширна библиотека с **повече от 4000 класа**, организирани в **пространства от имена (namespaces)**

- Те предлагат **полезна функционалност** за:
  - вход/изход
  - манипулация на низове
  - XML parse-ване
  - контроли за Windows Forms
  - създаване на графика и т.н.

Класът System.String

- Позволява **обработка на символни низове** (последователности от символи)
- **Символните низове** (strings) **представляват последователности от Unicode знаци** в кодиране UTF-16
  - Unicode поддържа много езици едновременно
  - Съхранява низовете в динамичната памет (защото `System.String` е референтен тип)
- Низовете са примитивен тип данни в C#

Символни низове – пример

```
static void Main(string[] args)
{
    string s = "Това е програма, която";
    string s2;
    s2 = "демонстрира класът ";
    s2 += " String!";
    Console.WriteLine("s = {0}", s);
    Console.WriteLine("s2 = {0}", s2);
    Console.ReadKey();

}
```

Класът `System.String`

По-важни методи и свойства на класа

- **`Length`** – връща дължината на низа
- **`Equals(string)`** – сравнява низа с друг низ
- **`Compare(string, string)`** – сравнява два низа лексикографски един с друг (прави се разлика между малки и главни букви)
- **операторите `==` и `!=`** – също сравняват низове
- **операторите `+` и `+=`** – слепват низове
- **`this[int]`** – индексатор, който връща символа на зададената позиция (броенето започва от 0)
- **`StartsWith(string)`** – проверява дали низът започва с посочения низ
- **`EndsWith(string)`** – проверява дали низът завършва с посочения низ

Класът System.String

По-важни **методи и свойства** на класа

- **`Substring(int startIndex, int Length)`** – извлича подниз по дадено начало и дължина
- **`IndexOf(string)`** – връща позицията на първото срещане на посочения низ или **`-1`**.
  - ✓ Има още няколко overload варианта – за търсене започвайки от даден начален индекс, по начален индекс и дължина и т.н. (вж. MSDN за повече информация)
- **`LastIndexOf(string str)`** – връща позицията на последното срещане на посочения низ

Класът `String` – пример

```
static void Main(string[] args)
{
   string s = "Това е програма на C#!";
    string b = s.Substring(6, 10);   // b = " програма "
   Console.WriteLine("\"{0}\".Substring(6,10) is \"{1}\".", s, b);
// Резултат: "Това е програма на C#". Substring(6, 10) is е " програма ".
   int progIndex = s.IndexOf("програма");
   Console.WriteLine("The index of \"{0}\" in \"{1}\" is {2}.",
                     "програма”, s, progIndex);
// Резултат: The index of "програма" in "Това е програма на C#!" is 7.
   int happyIndex = s.IndexOf("отличен"); // -1
   Console.WriteLine("The index of \"{0}\" in \"{1}\" is {2}.",
                     "отличен”, s, happyIndex);
// Резултат: The index of "отличен" in "Това е програма на C#!" is -1.
   Console.ReadKey();
}
```

ПАРСВАНЕ НА ТИПОВЕ

- **Парсването** на типовете има за цел да конвертира символното представяне на **даден тип в инстанция от този тип**
- Методът `Parse(string)` е обратен на метода `ToString()`
- Метод `Parse(string)` имат:
  - числовите типове (`Byte, Int32, UInt32, Decimal, Double, Boolean`, …)
  - изброените типове (наследниците на `System.Enum`)
  - типът `DateTime` (основен тип в .NET Framework за представяне на часове и дати)

ПАРСВАНЕ НА ТИПОВЕ - пример

```
static void Main(string[] args)
{
    Console.Write("Въведете цяло число a=");
    int a = Int32.Parse(Console.ReadLine());
    Console.Write("Въведете реално число side=");
    double side = double.Parse(Console.ReadLine());
    bool flag = bool.Parse("true");
    Console.WriteLine(" цяло число a= {0} \n реално числко side
  = {1} “, a, side);
}
```

КЛАСЪТ System

- **System** – съдържа **основни типове**, използвани от всяко .NET приложение
  - **System.Object**
  - **System.Console**

  осигурява средства за вход и изход от конзолата
- **Вход от конзолата**
  `Console.ReadLine()`
  чете цял символен ред и връща **`string`**
  `Console.Read()`
  чете единичен символ и връща **`char`**
- **Изход към конзолата**
  `Console.Write(…)`
  - печата на конзолата подадените като параметри данни (приема `string`, `int`, `float`, `double`, …
  - приема параметрични форматиращи низове

```
Console.WriteLine("Днес е {0:dd.MM.yyyy} г.", DateTime.Now);
```

КЛАСЪТ System.Math

- Класът **System.Math** съдържа методи за извършване на основни числови операции като повдигане в степен, логаритмуване, коренуване и някои тригонометрични функции

```
Console.WriteLine("Izpolzvane na matematicheski funkcii");
Console.WriteLine("Abs. stojnost = " + Math.Abs(-a));
Console.WriteLine("Max stojnost = " + Math.Max(5,10));
Console.WriteLine("Min stojnost = " + Math.Min(3,1));
Console.WriteLine("Stepenuvane X na stepen Y = " + Math.Pow(2,5));
Console.WriteLine(" Koren vtori ot arg = " + Math.Sqrt(25.64));
double pi=Math.PI;
Console.WriteLine(pi);
```

Класът System.Random

- Понякога в програмирането се налага да използваме случайни числа
- Например искаме да генерираме 6 случайни числа в интервала между 1 и 49 (не непременно различни)
- Това можем да направим използвайки класа **System.Random** и неговия метод **Next()**
- Преди да използваме класа **Random** трябва да създадем негова **инстанция**
- Тя **се инициализира със случайна стойност**

ГЕНЕРИРАНЕ НА СЛУЧАЙНО ЧИСЛО

- Чрез извикване на метода **Next(n)** можем да генерираме случайно число в интервала **[0…n)**
- Забележете, че този метод може да върне **нула**, но връща **винаги** случайно число по-малко от зададената стойност **n**.
- Ако искаме да получим число в интервала **[1…49]**, трябва да използваме израза

  Next(49) + 1

Примерен код за програма, която генерира 6 случайни числа в интервала [1…49]

```
class RandomNumbersBetween1and49
{
static void Main(string[] args)

  {
        Random rand = new Random();
        for (int number = 1; number <= 6; number++)
        {
            int randomNumber = rand.Next(49) + 1;
            Console.Write("{0} ", randomNumber);
        }
        Console.ReadKey();
   }
}
```

ПРОСТРАНСТВА ОТ ИМЕНА

- **Пространство от имена** (namespace/package) в ООП наричаме контейнер за група класове, които са обединени от общ признак или се използват в общ контекст
- **Пространството от имена** в C# (**namespace**) е логическо групиране на типове за целите на идентификацията.
  - спомагат за една по-добра логическа организация на изходния код
  - създават семантично разделение на класовете в категории и улесняват употребата им в програмния код

ПРОСТРАНСТВА ОТ ИМЕНА

Прието е:

- **името на** папката да съвпада **с името** на пространството
- **имената на** файловете да съвпадат с **имената на** класовете, които се съхраняват в тях
- в някои езици за програмиране **компилацията** на изходния код на дадено пространство зависи от разпределението на елементите на пространството в папки и файлове на диска

(за Java файловата организация е задължителна)

ВЛОЖЕНИ ПРОСТРАНСТВА

- Освен **класове**, пространствата могат да **съдържат в себе си и други пространства** (**вложени пространства**, nested namespaces)
- По този начин се изгражда **йерархия от пространства**, която позволява още по-прецизно разделение на класовете според тяхната семантика
- При **назоваването на пространствата** в йерархията се използва символът **.** за разделител (точкова нотация)

Например пространството **System от .NET Framework** съдържа в себе си подпространството **Collections** и така пълното название на вложеното пространство **Collections** добива вида **System.Collections**

ПЪЛНИ ИМЕНА НА КЛАСОВЕТЕ

- Класовете трябва да **имат уникални имена** само в рамките на пространството от имена, в което са дефинирани
- Извън дадено пространство може да има класове с произволни имена, без значение дали **съвпадат** с някои от имената на класовете в пространството

ПЪЛНО ИМЕ НА КЛАС

- наричаме **собственото име на класа**, предшествано от името на пространството, в което този клас е дефиниран
- пълното име на всеки клас е уникално
- отново се използва точковата нотация

```
<namespace_name>.<class_name>
```

В .NET Framework понякога има класове от различни пространства със съвпадащи имена, например:

```
System.Windows.Forms.Control
System.Web.UI.Control
System.Windows.Controls.Control
```

ВКЛЮЧВАНЕ НА ПРОСТРАНСТВО

- В зависимост от предметната област често се налага **многократното използване на класове** от някое пространство
- Има механизъм за **включване** на пространство към текущия файл със сорс код

```
using <namespace_name>;
```

След като е включено дадено пространство, всички класове дефинирани в него могат свободно да се използват, без да е необходимо използването на техните пълни имена

Включването на пространства не е рекурсивно, т.е. при включване на пространство не се включват класовете от вложените в него пространства

Благодаря за вниманието!