**Общи понятия за класове в C#**

**Класовете** са градивните елементи в ООП. Чрез класовете потребителя дефинира нови типове данни за описание на неща, които могат да имат общи характеристики. Такива неща могат да бъдат, както от реалния свят (Превозно средство, Човек, Служител, Животно, Речник от думи, Изображение и др. ), така и важни елементи от програми ( Файл, Мрежов интерфейс, Връзка към база данни и др.). Разбира се много от елементите, които са били част от реалния свят са били пренесени в програмите. Именно това пренясяне и по-скоро описание е било направено чрез класовете.

Когато се говори за класове винаги се споменават и обектите. **Обект** наричаме конкретен представител от даден клас. Например студента Иван Петров от специалност СИ, 2 курс е представител (инстанция) на класа Student.

**Инстанцирането** е процесът на създаване на обект от даден клас. Например обекта st1 е инстанция на класа Student:

Student st1 = new Student("Ivan", "Petrov");

**Свойство** се нарича видимата за външния свят характеристика на обектите от даден клас. Например свойствата на класа Student са личните му имена, личните му данни, оценките му и др.

**Методите** са действия, които всички обекти от даден клас могат да извършват. Например всички обекти от класа Student могат да извършват действието „явяване на изпит“.

**Интерфейсът** е описание на съвкупност от действия, които даден обект може да извършва. Ако един обект може да извършва всички действия от даден интерфейс, казваме че обектът реализира, или имплементира интерфейста. Например класа Student може да реализира интерфейса „ILearner“ (учащ) съдържащ действието Learning (учене). Много е важно да се прави разлика между понятието интерфейс от ООП и потребителски интерфейс на дадена програма.

**Наследяване на класовете** в ООП наричаме възможността на един клас, наречен наследник да придобие характеристики, свойства и методи на друг клас – родител (или базов клас). Това ще се разгледа по-късно. Но като пример може да дадем класа „InterruptedStudent“, като наследник на базовия Student, като към наследените методи и свойства може да добави и собствени, от типа на “MassiveBrawlByMom”.

=======================================================

Нека разгледаме първо основните понятия за класовете. Важно е да се знае, че те са референтен тип. Най-простата декларация е следната:

class YourClassName

{

}

Ключови думи преди „class“ са най-често Attributes и class modifiers. За non-nested класове модификаторите са: public, internal, abstract, sealed, static, unsafe, partial.

След името на класа YourClassName може да има параметри за Generic type , базов клас и интерфейс.

В дефиницията на класа може да има members (които са methods, properties, indexers, events, delegates, fields, constructors, overloaded operators, nested types, and a finalizer).

Полетата може да имат следните модификатори:

Static modifier: static

Access modifiers: public, internal, private, protected

Inheritance modifier: new

Unsafe code modifier: unsafe, extern

Read-only modifier: readonly

Threading modifier: volatile

Да разгледаме пример 1.1 :

class Student

{

private string mFirstName;

private string mLastName;

private int mAge;

public Student()

{ }

public Student(string fn, string ln, int age)

{

this.mFirstName = fn;

this.mLastName = ln;

this.mAge = age;

}

public void PrintStudent()

{

Console.WriteLine("FirstName:{0}\nLastName:{1}\nAge:{2}\n", mFirstName, mLastName, mAge);

}

}

////////////////////////////////////

class Program

{

static void Main(string[] args)

{

Student st1 = new Student("Ivan", "Petrov", 21);

Student st2 = new Student("Stefan", "Marinov",20);

st1.PrintStudent();

st2.PrintStudent(); }

}

==============================

Класа Student е добавен в отделен файл към проекта. Той има три полета, които започват с m, за да се знае че са private членове (members). Има добавен конструктор по подразбиране и конструктор с параметри. Ключовата дума this се използва като референция към класа за да се поясни, че ще се използва имено променливата която е член на класа.

==================================================

Статични полета и методи – Пример 1.2

class Panda

{

public string Name; // Instance field

public static int Population; // Static field

public Panda(string n) // Constructor

{

Name = n; // Assign the instance field

Population = Population + 1; // Increment the static Population field

}

public static void PrintFavouriteFood()

{

Console.WriteLine("Pandas eat Bamboo.");

}

}

////////////////////////////////////

class Program

{

static void Main(string[] args)

{

Panda.PrintFavouriteFood();

Panda p1 = new Panda("Pan Dee");

Panda p2 = new Panda("Pan Dah");

Console.WriteLine(p1.Name); // Pan Dee

Console.WriteLine(p2.Name); // Pan Dah

Console.WriteLine(Panda.Population); // 2

}

}

===================

Разлика между const и readonly. И двете се използват за константи. Но разликата е, че const трябва да бъде инициализирана при декларация и по подразбиране е static – т.е. може да бъде използвана само с името на класа. А readonly може да се инициализира по-къно, но само веднъж – след това не може да се променя.

class Test

{

public const int CONST\_VALUE = 2;

public readonly int RO\_VALUE;

public Test()

{

RO\_VALUE=3;

}

public int getRO() { return RO\_VALUE; }

}

class Program

{

static void Main(string[] args)

{

int x = Test.CONST\_VALUE;

Test a = new Test();

int y = a.getRO();

}

}

Разбира се, че може да се направи RO\_VALUE също да е статично поле. Но тогава конструктура и метода getRO() трябва да са обявени също като статични.

=======================================

Да разгледаме по-подробен пример при който се използват пропъртита (properties). Пропъртитата имат два аксесора: get и set. Ако един от тях липсва тогава пропъртито ще бъде readonly или setonly. Важно е да се знае, че типа на връщания резултат от get е този с който е дефинирано пропъртито. При аксесора set се използва служебен параметър value, който също е с типа на пропъртито и служи да съхрани стойноста, която стои от дясно на равното.

String s = person1.Name; // тук се извиква аксесора get, който връща стойността на полето

// mFirstName

person1.Name = “Ivan”; //тук се извиква аксесора set, като служебния параметър value има

//стойност „Ivan“

Пример 1.3

class Person

{

private string mFirstName;

private string mLastName;

private DateTime mDateOfBirth; //this is a reference of DateTime!!

public Person()

{ }

public Person(string mFirstName, string mLastName)

{

this.mFirstName = mFirstName; //example what do “this”

this.mLastName = mLastName;

}

public Person(string fName) : this(fName, "unknown")

{ }

//public void SetFirstName(string fName)

//{

// this.mFirstName = fName;

//}

//public string GetFirstName()

//{

// return this.mFirstName;

//}

public string Name

{

get

{

return this.mFirstName;

}

set

{

// value in this case is string type

if ((value != null) && (value.Length > 0)) //string.IsNullOrEmpty()

{

this.mFirstName = value;

}

else

{

throw new ArgumentException("Class Person: Invalid first Name!");

}

}

}

public string Family

{

get

{

return this.mLastName;

}

set

{

// value in this case is string type

if ((value != null) && (value.Length > 0)) //string.IsNullOrEmpty()

{

this.mLastName = value;

}

else

{

throw new ArgumentException("Class Person: Invalid last Name!");

}

}

}

public DateTime DateOfBirth

{

get

{

return this.mDateOfBirth;

}

set

{

if ((value.Year >= 1930) && (value.Year <= DateTime.Now.Year))

{

this.mDateOfBirth = value;

}

else

{

throw new ArgumentOutOfRangeException("Class Person: Invalid date of birth!");

}

}

}

public int Age

{

get

{

DateTime now = DateTime.Now;

int yearsOld = now.Year - mDateOfBirth.Year;

DateTime birthdayThisYear = new DateTime(now.Year, mDateOfBirth.Month, mDateOfBirth.Day, mDateOfBirth.Hour, mDateOfBirth.Minute, mDateOfBirth.Second);

if (DateTime.Compare(now, birthdayThisYear) < 0)

{

yearsOld--;

}

return yearsOld;

}

}

}

////////////////////////////////////

class Program

{

static void Main(string[] args)

{

try

{

Person p = new Person("Gosho", "Peshev");

Person f = new Person { Name="Gosho", Family="Peshev", DateOfBirth= new DateTime(1997,8,5) };

Person g = new Person();

g.Name = "Pesho";

g.Family = "Goshev";

g.DateOfBirth = new DateTime(1998, 7, 15);

Console.WriteLine("{0} {1} is born on {2:dd.MM.yyyy}", g.Name, g.Family, g.DateOfBirth);

Console.WriteLine("{0} is {1} years old", g.Name, g.Age);

}

catch (ArgumentException ex)

{

Console.WriteLine(ex.Message);

}

}

}

Тук се вижда, че използването на двата метода SetFirstName() и GetFirstName() е поето от пропъртито Name.

Трябва да обърнем внимание, че полето mDateOfBirth е само референция към дата и за да се използва трябва да я създадем като обект.

В класа Person се използва извикване на конструктор от друг конструктор. Навсякъде в класа използването на this. означава извикване на member, докато this() е объщение към конструктор. Когато се извиква един конструктор от друг, извиквания се изпълнява първи (в нашия случай този, който е с два параметъра).

В метода Main() са инстанцирани три обекта по три различни начина.