Object Oriented Programing

Lecture 5 Constructor and more

Arsalan Rahman Mirza

Computer Science Department

Faculty of Science

2022-2023

How to access Private data

```
public class Author
{
  private string name;
  public void  SetName(string n)
  {
       name = n;
  }
}
```

```
public class Author
{
  private string name;
  public string  GetName()
  {
       return name;
  }
}
```

Property in C# Classes

- A property in C# object is a memory that is set and get through a filter.

- Property Keywords:
- set : to define how a variable is set.
- get : to define how a variable is get.
- value: the value we receive from user which we will use in SET.
- return: the value we will return to user which we have in GET.

Property in C# Classes

```
public class Author
    {
        private string name;
        public string Name
        {
            get
            {
                return name;
            }
            set
            {
                name = value;
            }
        }
    }
```

Property in C# Classes

```
public class Person
   {
      private string name;
      private int age;
      public string Name{
         get { return name!=null ? name : "NA"; }
         set { name = value.ToLower(); } }
      public int Age {
         get { return age; }
         set { if ((value > 0) && (value < 150)) age = value; else age = 0; } }
   }
```

Property in C# Classes

```
public class Person
   {
      private string name;
      private int age;
      public Person(string n, int a)
      {
         name = n;
         age = a;
      }
      public string Name { get { return name; } private set { } }
      public int Age { get { return age; } }
   }
```

```
static void Main()
      {
         Person p = new Person("Ali",23);
         Console.WriteLine(p.Name);
         Console.WriteLine(p.Age);
         p.Name = "Mostafa"; // error
         p.Age = 34; // error

      }
```

Copy Constructor

- A constructor that creates an object by copying variables from another object or that copies the data of one object into another object is termed as the **Copy Constructor**.

```
static void Main(string[] sr)
      {
         // Create a Person object by using the instance constructor.
         Person person1 = new Person("George", 40);
         // Create another Person object, copying person1.
         Person person2 = new Person(person1);
         // Change each person's age.
         person1.Age = 39;
         person2.Age = 41;
         // Change person2's name.
         person2.Name = "Charles";
         // Show details to verify that the name and age fields are distinct.
         Console.WriteLine(person1.Details());
         Console.WriteLine(person2.Details());
         // Keep the console window open in debug mode.
         Console.WriteLine("Press any key to exit.");
         Console.ReadKey();
      }
```

```
class Person
   {
      public string Name;
      public int Age;
      // Copy constructor.
      public Person(Person previousPerson)
      {
         Name = previousPerson.Name;
         Age = previousPerson.Age;
      }
      public Person(string name, int age)
      {
         Name = name;
         Age = age;
      }
      public string Details()
      {
         return Name + " is " + Age;
      }
   }
```

Private Constructor

- A private constructor is a special instance constructor. It is generally used in classes that contain static members only. If a class has one or more private constructors and no public constructors, other classes (except nested classes) cannot create instances of this class.

```
class Mathmatic
   {
      public static double pi = 3.14;
      private Mathmatic(){}
   }
```

```
static void Main(string[] sr)
      {
         Mathmatic p = new Mathmatic(); // Error

         Console.WriteLine(Mathmatic.pi);
      }
```

Class vs Struct

- The main differences between Class and Struct are:

1)

The ***class objects*** copy by reference.

The ***struct objects*** copy by value.

Class copy by reference

```
class Point
  {
    private int x, y;
    public int X { get { return x; }set { x = value; } }
    public int Y { get { return y; }set { y = value; } }
    public Point(int x, int y)
    {
      this.x = x;
      this.y = y;
    }
  }
```

```
static void Main()
    {
      Point a = new Point(10, 10);
      Point b = a;
      Console.WriteLine(a.X);
      Console.WriteLine(b.X);
      a.X = 15;
      Console.WriteLine(a.X);
      Console.WriteLine(b.X);
    }
```

Struct

```
struct Point
   {
      private int x, y;
      public int X { get { return x; }set { x = value; } }
      public int Y { get { return y; }set { y = value; } }
      public Point(int x, int y)
      {
         this.x = x;
         this.y = y;
      }
   }
```

```
static void Main()
      {
         Point a = new Point(10, 10);
         Point b = a;
         Console.WriteLine(a.X);
         Console.WriteLine(b.X);
         a.X = 15;
         Console.WriteLine(a.X);
         Console.WriteLine(b.X);
      }
```

Link to other class

```
class student
{
    public string name, email;
    public int birthYear;
    public subject[] subjects;
    public student(string s)
    { name = s; }
}
```

```
class subject
{
    string name;
    int[] marks;
    string teacher;
    public subject(string n)
    { name = n; }
}
```

ENUM

Enum is a constant List

Enum index is an integer

Such as Weekdays, only lists that never change.

Such as:

Blood Types [A+, A-, …]

Directions [North, South, …]

Weather [Hot, Cold, …]

Enum

```
enum Weekdays { Sunday, Monday, Tuesday ,
                Wednesday, Thursday     };
class Employee
{
    Weekdays offDay;
    public Employee()
    {
        offDay = Weekdays.Sunday;
    }
}
```

```
public enum Season
    {
        Spring,
        Summer,
        Autumn,
        Winter
    }
```

```
public static void Main()
        {
            Season a = Season.Autumn;
            Console.WriteLine($"Integral value of {a}
             is {(int)a}");  // output: Integral value of
             Autumn is 2
            var b = (Season)1;
            Console.WriteLine(b);  // output: Summer
        }
```

Destructor

- destructor (finalizer) is used to destroy objects of class when the scope of an object ends. It has the same name as the class and starts with a tilde.

```
class Person
{
    public Person()
    {
       Console.WriteLine("Constructor    called.");
    } // destructor
    ~Person()
    {
        Console.WriteLine("Destructor called.");
}
```

- When we create an object of the Person class, the constructor is called. After the scope of the object ends, object p1 is no longer needed. So, the destructor is called implicitly which destroys object p1.

Features of Destructors

There are some important features of the C# destructor. They are as follows:

We can only have one destructor in a class.

A destructor cannot have access modifiers, parameters, or return types.

A destructor is called implicitly by the Garbage collector of the .NET Framework.

We cannot overload or inherit destructors.

We cannot define destructors in structs.