OOP – Constructors and Methods

Contents

Constructors

A constructor is a built in method in a class that the .NET runtime executes when an object is first instantiated (created). You can add code in the constructor to perform any initialisation operations for a new object. A constructor has the same name as the class. When an object is created (instantiated) the constructor is automatically called and any code inside it is executed.

⇘ Example:

```
Person objPerson1 = new Person();
```

When the above line is executed the .NET runtime calls the constructor in the person class and executes any code in it.

⇘ Example:

```
public Person()
{
        _EyeColour = "Brown";
}
```

The above line would create all instances of the class with _EyeColour set to brown.

Activity 1. Creating a constructor

person.cs1 & testPerson.aspx

Create the following class:

```
public class Person
{
    private int _Age;
    private string _Name;
    private string _EyeColour;

    public Person()
    {
        _EyeColour = "Brown";
    }

    public int Age
    {
        get
        {
            return _Age;
        }
        set
```

```
        {
            if ((value >= 0) && (value <= 130))
            {
                _Age = value;
            }
            else
            {
                throw new Exception("Age cannot be less than 0 or greater than
130");
            }
        }
    }

    public string Name
    {
        get
        {
            return _Name;
        }
        set
        {
            _Name = value;
        }
    }

    public string EyeColour
    {
        get
        {
            return _EyeColour;
        }
  set
        {
            _EyeColour = value;
        }
    }
}
```

To test the class create a web form with 2 labels named lblPerson1 and lblPerson2. Add the following code to the codebehind:

```
protected void Page_Load(object sender, EventArgs e)
{
        Person objPerson1 = new Person();
        Person objPerson2 = new Person();

        objPerson1.Name = "Sally Browm";
        objPerson1.Age = 20;

        lblPerson1.Text = objPerson1.Name;
        lblPerson1.Text += " is " + objPerson1.Age.ToString() + " years old
and has ";
        lblPerson1.Text += objPerson1.EyeColour + " eyes";

        objPerson2.Name = "John Smith";
        objPerson2.Age = 24;

        lblPerson2.Text = objPerson2.Name;
        lblPerson2.Text += " is " + objPerson2.Age.ToString() + " years old
and has ";
        lblPerson2.Text += objPerson2.EyeColour + " eyes";
```

```
}
```

Passing arguments

You can pass data into the constructor by defining parameters (arguments). The constructor is then called a parameterized constructor.

When you create a new instance of a class you then need to pass in a value for the parameter to the constructor.

You can define multiple constructors for a class. This is called overloading and is covered later in these notes.

Activity 2. Passing Parameters

person.cs2 & testPerson.aspx

```
public Person(string strEyeColour)
{
        _EyeColour = strEyeColour;
}
```

Now when testing the class you will need to pass the eye colour to the constructor:

```
Person objPerson1 = new Person("Blue");
Person objPerson2 = new Person("Brown");
```

Methods

The methods of a class define the behaviour and functionality associated with the class. Methods are implemented as functions. Methods define the logic in your application (what your application does). You create a method in a class for each set of business identified for the class during design phase. If you are using UML for your design the methods are identified by your sequence diagrams.

When creating a method choose a good name for your method. Choose a name that describes the purpose of the method. For example if the purpose of the method is to add and item to a shopping cart then you should name it something like AddItemToCart().

Methods can have parameters. Parameters define the data that is passed into the method. Use a good naming convention for your parameter names.

To create good methods make sure the method has a single purpose and is no longer than one page. If a method is long, break it up into several methods. This makes each method much easier to build and maintain.

Activity 3. Adding a method

person.cs3 & testPerson2.aspx

We will add a method to the person class.

```
public string walk()
{
        string strMessage;

        strMessage = Name + ": You are now walking forwards";

        return strMessage;
}
```

The above method returns a value of type string. To test this method we need to create a web page with 2 labels names lblPerson and lblWalk. From the code behind add the following code:

```
protected void Page_Load(object sender, EventArgs e)
{
        Person objPerson1 = new Person("Blue");
        string strReturnMessage;

        objPerson1.Name = "Sally Browm";
        objPerson1.Age = 20;

        lblPerson1.Text = objPerson1.Name;
        lblPerson1.Text += " is " + objPerson1.Age.ToString() + " years old
and has ";
```

```
        lblPerson1.Text += objPerson1.EyeColour + " eyes";

       //call the walk method
       //this method returns a string
        strReturnMessage = objPerson1.walk();

        //display message to label
        lblWalk.Text = strReturnMessage;
}
```

Passing arguments

You can supply arguments (parameters) to methods.

Overloading

There may be times when you want to have different methods that carry out the same function, but implemented differently. You could define different method names, but a better way is to use overloaded methods. An overloaded method is a method that has the same name but a different signature. The signature of a method is the parameter list.

A method can have any number of overloads, as long as each has a different signature. The signature is different if either the number of parameters is different or the data types are different or both.

Method overloading

Overloaded methods can be useful when you have different blocks of code executed for a similar action.

Activity 4. Overloading the walk method

person.cs4 & testPerson4.aspx

```
public string walk()
{
        string strMessage;

        strMessage = Name + ": You are now walking forwards";

        return strMessage;
}

public string walk(string direction)
{
        string strMessage;

        if (direction == "Back")
        {
            strMessage = Name + ": You are now walking backwards";
        }
        else
        {
            strMessage = Name + ": You are now walking forwards";
        }

        return strMessage;
}

public string walk(int direction)
{
        string strMessage;

        if (direction < 0)
        {
            strMessage = Name + ": You are now walking backwards";
```

```
        }
        else
        {
            strMessage = Name + ": You are now walking forwards";
        }

        return strMessage;
}
```

To test these methods you need to call each one. If no parameter is passed it will execute the 1st walk method, if a string value is passed the 2nd and if an integer is passed the 3rd.

Constructor overloading

A constructor is actually a special method. You can have more than 1 constructor for a class just as you can have more than one method in a class with the same name.

Activity 5. Constructor overloading

Modify the person class so that it has 2 constructors.

Person.cs5 & testperson5.aspx

```
public Person(string strEyeColour)
{
        _EyeColour = strEyeColour;
}

public Person()
{
        _EyeColour = "Brown";
}
```

The first constructor has a string parameter passed to it to set the eye colour and the second does not have any parameter passed to it.

To test the 2 different constructors create a web page with 2 labels on it named lblPerson1 and lblPerson2.

Add the following code to the codebehind file:

```
protected void Page_Load(object sender, EventArgs e)
{

        Person objPerson1 = new Person("Blue");
        Person objPerson2 = new Person();

        objPerson1.Name = "Sally Browm";
        objPerson1.Age = 20;


        lblPerson1.Text = objPerson1.Name;
        lblPerson1.Text += " is " + objPerson1.Age.ToString() + " years old
and has ";
        lblPerson1.Text += objPerson1.EyeColour + " eyes";

        objPerson2.Name = "John Smith";
        objPerson2.Age = 24;

        lblPerson2.Text = objPerson2.Name;
        lblPerson2.Text += " is " + objPerson2.Age.ToString() + " years old
and has ";
        lblPerson2.Text += objPerson2.EyeColour + " eyes";
}
```

Activity 6. Create a class

Create a student class with the following properties:

First name, last name, student number, status of enrolment (True or false) and course number.

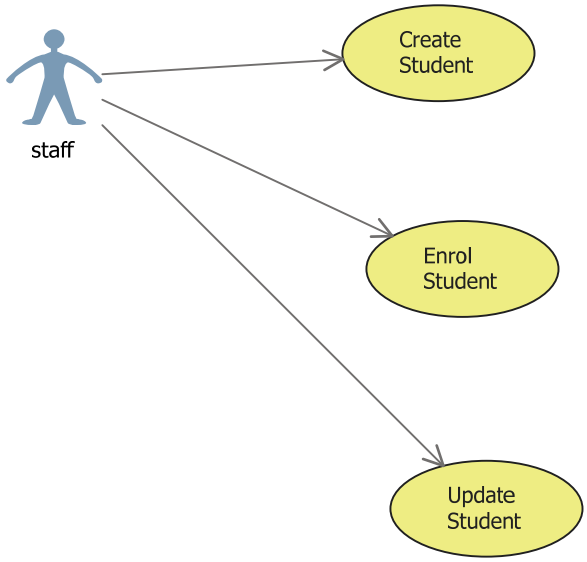
When a student is created they may be created as already enrolled or not yet enrolled (the default setting). Provide 2 constructors one that takes in an argument of type string containing their existing student number. This constructor will allocate the student number property to the one passed in and set the enrolment status to true. The other constructor has no input parameters and sets the enrolment status to false.

Create a method that enrols them into a course by passing the course number into the method. This method will then set the course number property and set the enrolment status to true.

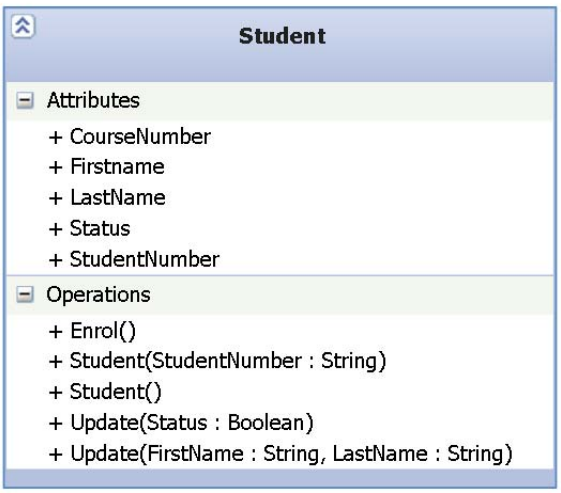
Create another 2 methods called update which can update their first and last name if 2 strings are passed in or update their enrolment status if a Boolean value is passed in.

Make sure you test the class.

Use case diagram:



Class Diagram:



References

C# for programmers: second edition by Harvey M Deitel and Paul J. Deitel.