

# OOP – Creating a class: Properties

A class is the template or blue print of an object. It defines the characteristics of the object and the actions the object can take.

Previously we created classes with public variables:

```
public string strFirstName;
```

The above line allows strFirstName to be accessed from outside of the class. Meaning that any other file can read and write to the strFirstName variable from outside of the class.

One of the benefits of OOP is encapsulation. Encapsulation prevents direct access to data this makes the data more secure and therefore reliable.

The recommendation is to create all variables as private and grant access to the variables via properties (also called accessor methods).

## Properties

The properties of a class define the data associated with the class. For example a product class may have a productName, unitPrice and a productID. Each object created from the class can have a different set of values for these properties.

Following best practice creating properties requires 2 steps. First you create a private variable to retain the property value called a field. You make the variable private so that it cannot be directly accessed by any code outside of the class.

Next you create a property statement. The property statement defines the property and the accessors used to get and set the property. The set accessor, sometimes called the setter, sets the property's value. The get accessor, sometimes called the getter, returns the property's value.

This technique encapsulates the property by providing access to it only through the accessors. You can write code in the accessors to validate data, perform formatting or any other business logic.

```
private string _StudentNumber;

public string StudentNumber
{
    get
    {
        return _StudentNumber;
    }
    set
    {
        _StudentNumber = value;
    }
}
```

This allows access to the private variable \_StudentNumber. Access to this variable is only possible via the property. Some checks could be added to the property like for example a student number may need to be 6 digits in length.

### Activity 1. Student class

Student.cs & testStudent.aspx

Create the following class and test it.

```
public class Student
{
    private string _FirstName;
    private string _LastName;
    private string _StudentNumber;

    //properties
    public string FirstName
    {
        get
        {
            return _FirstName;
        }
    }
}
```

```

    }
    set
    {
        _FirstName = value;
    }
}

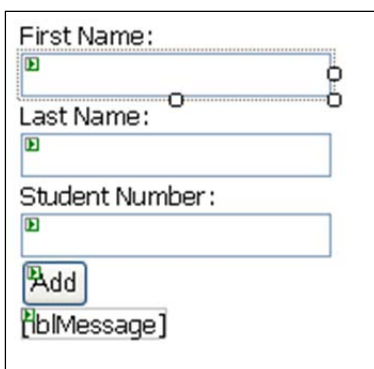
public string LastName
{
    get
    {
        return _LastName;
    }
    set
    {
        _LastName = value;
    }
}

public string StudentNumber
{
    get
    {
        return _StudentNumber;
    }
    set
    {
        _StudentNumber = value;
    }
}

public Student()
{
    //
    // TODO: Add constructor logic here
    //
}
}

```

To test the class create the following web form:



The screenshot shows a web form with three text input fields. The first field is labeled 'First Name:', the second 'Last Name:', and the third 'Student Number:'. Below these fields is a button labeled 'Add' and a label '[blMessage]'.

Code behind:

```

public partial class testStudent : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void btnAdd_Click(object sender, EventArgs e)
    {
        //create a new instance of the student class
    }
}

```

```

Student objStudent = new Student();

//allocate the values
objStudent.FirstName = txtFirstName.Text;
objStudent.LastName = txtLastName.Text;
objStudent.StudentNumber = txtStudentNumber.Text;

//display the values to the label
lblMessage.Text = objStudent.FirstName + " " + objStudent.LastName + "
" + objStudent.StudentNumber;
}
}

```

## Activity 2. Person class

Person.cs1 & testPerson.aspx

```

/// <summary>
/// Summary description for Person
/// </summary>
public class Person
{
    private int _Age;
    private string _Name;
    private string _EyeColour;

    public Person()
    {
        //
        // TODO: Add constructor logic here
        //
    }

    public int Age
    {
        get
        {
            return _Age;
        }
        set
        {
            _Age = value;
        }
    }

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

    public string EyeColour
    {
        get
        {

```

```

        return _EyeColour;
    }
    set
    {
        _EyeColour = value;
    }
}

```

Test it by creating a web form containing 2 labels named lblPerson1 and lblPerson2. Add the following code to the code behind:

```

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

    objPerson1.Name = "Sally Brown";
    objPerson1.Age = 20;
    objPerson1.EyeColour = "Blue";

    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;
    objPerson2.EyeColour = "Brown";

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

```

The above file will create 2 instances of the person class. The instances are called objPerson1 and objPerson2. As both objects are created from the same class (template) they both automatically get the Name, Age and EyeColour attributes.

### Activity 3. Modify the Age property

person.cs2

The age property could be set up so that it only allows values between 0 and 130.

Rewrite the class so that the property now looks like this:

```

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");
        }
    }
}

```

```
}
```

If age is not set correctly it will throw an exception.

Test this out by trying to assign a value less than 0 or greater than 130.

## Property types

---

Properties can be set up to be read only or write only or both read and write. If you don't specify anything (as we have above) it will allow you to both read and write.

To set up a property to be read only leave out the set section. To set up a property as write only leave out the get section.

```
public string EyeColour
{
    get
    {
        return _EyeColour;
    }
}
```

Notice there is no more set section to this property.

Now if you try and compile your code you will get an error on the line: `objPerson1.EyeColour = "Blue";`  
"Property eyeColour is read only"