


Awarding
Great British
Qualifications



Designing and Developing Object-Oriented Computer Programmes

Topic 5:

Object Orientation (1)

1

Scope and Coverage

This topic will cover:

Object Orientation


The Theory of Object Orientation

Classes and Objects

Using an Object

Methods

Constructors




2

Learning Outcomes

By the end of this topic students will be able to:

- Understand the structure of object orientation;*
- Create and make use of objects and classes;*
- Employ encapsulation for data protection;*




3

Title of Topic: Topic 1 - 1.4

Object Orientation - 1

- We have been using object orientation all the way through this unit, but we have not yet actually taken the time to look at what objects are in theory and in practice.
- These serve as the fundamental building blocks of modern OO programming languages – hardly surprising when we consider that OO stands for Object Orientation.




4

Title of Topic: Topic 1 - 1.5

Object Orientation - 2

- The concept can be quite difficult to understand at the beginning.
- Object orientation involves breaking a problem down into separate components in a way that is contradictory to how we would approach such a thing in our minds.
- Despite this, object orientation is a powerful programming paradigm and the tools it provides can lead to the development of richer and more elegant programming solutions than is possible with other, earlier development frameworks.



5


Title of Topic: Topic 1 - 1.6

The Theory of Object Orientation - 1

- Consider the following code:

```
namespace WindowsFormsApplication5
{
    public partial class FormClassExample : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
        }
    }
}
```



6

Title of Topic: Topic 1 - 1.7

The Theory of Object Orientation - 2

- We haven't made a big deal of it, but what we're doing here is creating a **class**.
- In object oriented programs, a class is a blueprint from which we derive instances of an **object**.
- Consider the chair on which you are (perhaps) sitting – there may be many chairs like this, but they all stem from a central design.
- That design is the class, and the specific chair you are sitting on is an object.

NCC

7

Title of Topic: Topic 1 - 1.8

The Theory of Object Orientation - 3

- There may be variations in objects – you may have chairs with the same design that have different colours, sizes, materials, and so on.
- It is our job as programmers to write classes that are sufficiently flexible that they can adapt to the real world as needed.

NCC

8

Title of Topic: Topic 1 - 1.9

The Theory of Object Orientation - 4

- When we create the class above, what we are doing is building a template for a Windows application.
- When we run the program, this class gets **instantiated** into an object.
- Usually we only have one of these, but there would be nothing to stop us having two, three, or three hundred objects **instantiated** from that simple class.


NCC

9

Title of Topic: Topic 1 - 1.10

The Theory of Object Orientation - 5

- This is a difficult concept to understand without a concrete example, so let's look at that.
- For this, we'll need to add a new kind of file to our project.
- We do that from the **solution explorer** – a part of the IDE we haven't had much cause to do anything with up until now.
- We add a new class by right clicking on the project name, choosing 'add' and picking class:




10

Title of Topic: Topic 1 - 1.11

The Theory of Object Orientation - 6

- This in turn will bring up the new item dialog.
- Click 'class' and give it a name. In this case, Person.cs
- Once we've done this, we'll get an empty class that we're going to expand so that it can be used to hold a set of useful data elements.



11


Title of Topic: Topic 1 - 1.12

The Theory of Object Orientation - 7

- The code we'll get is the following:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ObjectOrientationExample
{
    class Person
    {
    }
}
```




12

Title of Topic: Topic 1 - 1.13

The Theory of Object Orientation - 8

- The namespace here uses the name we give our project, it doesn't matter if it doesn't match what we have here.
- There will be more on this in the next few slides.
- The key element is what's within the braces there – the class Person portion of the code.
- We're about to make that do something very powerful.




13

Title of Topic: Topic 1 - 1.14

The Theory of Object Orientation - 9

- A quick definition of **Namespaces**; they are used to provide a "named space" in which your application exists.
- They're used to provide the C# compiler a context for all the named information in your program, such as variable names.
- Without namespaces, you wouldn't be able to make a class named Public, as .NET already uses one in its System namespace.




14

Title of Topic: Topic 1 - 1.15

The Theory of Object Orientation - 10

- The purpose of namespaces is to solve this problem.
- Namespace releases hundreds of names defined in the .NET Framework for your applications to use, along with making it so your application doesn't occupy names for other applications.
- So the short of it is - namespaces exist to resolve ambiguities a compiler wouldn't otherwise be able to do.




15

Title of Topic: Topic 1 - 1.16

Classes and Objects - 1

- As discussed above, the class is the template and the object is a specific instance of that template.
- The class tells the object what data and functions it should have, and the object tells the program what **state** the data has.
- In other words, a class tells an object what it should be, and the object tells the program what it is.
- For our purposes, a person might have a name and an age.




16

Title of Topic: Topic 1 - 1.17

Classes and Objects - 2

- That's the information we store about a person. It's only when we make an object that we give specific values to those fields.
- Let's see how:

```
class Person
{
    private String name;
    private int age;
}
```




17

Title of Topic: Topic 1 - 1.18

Classes and Objects - 3

- Don't worry at the moment about what the **private** part of this means just yet– just remember it should always be there when we add data elements.
- What we've done here is create two **attributes** that will be used to define what information an object might contain.
- Think here of an object as a way to create your own data-types that are made up of collections of other data types.



18

Title of Topic: Topic 1 - 1.19

Classes and Objects - 4

- While it's not wholly accurate, it's useful to think of a class as a new data type, of our own design, that has many different drawers in which we might store information.
- This is a technique known as **encapsulation** – storing the data in a class along with the methods that are going to act upon this data.

NCC

19

Title of Topic: Topic 1 - 1.20

Classes and Objects - 5

- However, this information currently isn't accessible to any other part of the program.
- For that, we need to add in some **accessor** functionality.
- In some languages this is done by creating a **set** and **get** function for each variable we have, like so:

```
void setAge(int a)
{
    age = a;
}

int getAge()
{
    return age;
}
```

NCC

20

Title of Topic: Topic 1 - 1.21

Classes and Objects - 6

- This works perfectly well, but the .NET framework has a better system – **properties**. These let us hook our objects up to the usual IntelliSense that we see when we type code into the IDE. To create a property, we provide the following code:

```
public int Age
{
    get
    {
        return age;
    }

    set
    {
        age = value;
    }
}
```


NCC

21

Title of Topic: Topic 1 - 1.22

Classes and Objects - 7

- Note here that we use **public** rather than **private**.
- This is true of properties and methods, but not of attributes.
- We'll talk about why later in the unit.
- We can add a property for each variable we want to expose to the rest of our program.
- Adding in properties for both our attributes will give us a class that looks like this:



22

Title of Topic: Topic 1 - 1.23


Classes and Objects - 8

```
namespace ObjectOrientationExample
{
    class Person
    {
        private string name;
        private int age;
        public int Age
        {
            get
            {
                return age;
            }

            set
            {
                age = value;
            }
        }

        public string Name
        {
            get
            {
                return name;
            }

            set
            {
                name = value;
            }
        }
    }
}
```




23

Title of Topic: Topic 1 - 1.24

Classes and Objects - 9

- The code inside **set** and **get** here can be manipulated too, to change the way the object behaves when we try to query or assign values.
- This can be a powerful technique and we will have cause to see it in the future.
- Now we've got a class of our own – a Person.
- This is now a thing we can use in the rest of the program.




24

Title of Topic: Topic 1 - 1.25

Using an Object - 1

- Now we've got a class of our own – a Person.
- This is now a thing we can use in the rest of the program.
- First of all, we're going to create what is called a **class wide variable** – this sits outside of any of our methods, which means it's available to all of them.
- We define this underneath the class definition:

```
public partial class frmClassExample : Form
{
    Person me;
```




25

Title of Topic: Topic 1 - 1.26

Using an Object - 2

- Here, we're saying 'we have a variable called **me** that is of type **Person**'.
- We wrote that Person class ourselves, so C# accepts it without question.
- In a very real sense, we're taking part in actively constructing the language of C#.




26

Title of Topic: Topic 1 - 1.27

Using an Object - 3

- Next, we need to **instantiate** this object. We do this using the **new** keyword, and usually we do it in the load event of the form.

```
private void frmClassExample_Load(object sender, EventArgs e)
{
    me = new Person();
}
```



27

Title of Topic: Topic 1 - 1.28

Using an Object - 4

- And then finally, we use the click event of our button to actually store the values in the object we created.

```
private void cmdStore_Click(object sender, EventArgs e)
{
    String name = txtName.Text;
    int age = Int32.Parse(txtAge.Text);

    me.Name = name;
    me.Age = age;
}
```

NCC

28

Title of Topic: Topic 1 - 1.29

Using an Object - 5

- Now, we have the name and age from the textboxes stored happily in our object.
- We can prove it too. Let's add a second button called cmdDisplay:

```
private void cmdDisplay_Click(object sender, EventArgs e)
{
    txtName.Text = me.Name;
    txtAge.Text = "" + me.Age;
}
```

NCC

29

Title of Topic: Topic 1 - 1.30

Using an Object - 6

- When you run this program, type in a name and age.
- Then press the store button, and then delete the text you had put into the textboxes.
- When you press the display button you'll see both pieces of data get restored.


NCC

30

Title of Topic: Topic 1 - 1.31

Methods - 1

- In addition to providing attributes for our class, we can also provide **methods**.
- Methods are functions that exist within a class, and normally permit us to act upon the data within.
- As noted above, storing the data and the methods that act upon that data together is known as **encapsulation**.




31

Title of Topic: Topic 1 - 1.32

Methods - 2

- Consider the following scenario:
- You've got to write a program that tracks the names and ages of several people, and also permits people to add to and subtract from their ages as necessary.
 - Upon instruction the age should increase, or decrease by a set value.
- Think how we might do that using the tools we discussed before this chapter – we'd need to keep track of two variables for each person, work out how much to increase or decrease ages by, work out which person to do that for and hope that nothing falls out of sync.




32

Title of Topic: Topic 1 - 1.33

Methods - 3

- However, with object orientation we have the object acting as the glue keeping the name and age of someone together.
- All we need to do is provide a set of methods that permit someone to modify the contents.



33

Title of Topic: Topic 1 - 1.34

Methods - 4

- We can do this like so:

```
class Person
{
    private String name;
    private int age;


    public void adjustAge(int amount) {
        age = age + amount;
    }

    public int Age
    {
        get
        {
            return age;
        }

        set
        {
            age = value;
        }
    }

    public string Name
    {
        get
        {
            return name;
        }

        set
        {
            name = value;
        }
    }
}
```




34

Title of Topic: Topic 1 - 1.35

Methods - 5

- Putting this into practise you can go back into your form and create: *(This can be seen in the student guide)*

1. A textbox to store the amount to adjust ages (txtAmount)
2. Make the program use three objects rather than one
3. Add a textbox to indicate which object we wish to work with (txtWhich)
4. Add a button to permit the adjustment (cmdAdjust)




35

Title of Topic: Topic 1 - 1.36

Constructors - 1

- We have one last thing to cover before we're done with our initial discussion of objects.
- Isn't it awkward to have to store all the details in each person before we can do anything else with the program?
- Wouldn't it be useful if we could give the objects some initial starting values? Well, we can!




36

Title of Topic: Topic 1 - 1.37

Constructors - 2

- We can provide each object with something called a **constructor** – a method that’s executed when we create the class with the new keyword.
- Let’s say we wanted to provide the object with a name and an age when created – we’d add the following constructor to our class definition:

```
public Person(String n, int a)
{
    name = n;
    age = a;
}
```




37

Title of Topic: Topic 1 - 1.38

Constructors - 3

- The thing that defines a constructor is that it has exactly the same name as the class itself (in this case, Person) and does not have a return type.
- Once we provide a constructor like this we no longer create a Person using an empty parameter list.
- Instead, we either need to always provide a name and age, or provide a **second** constructor that works with no parameters:




38

Title of Topic: Topic 1 - 1.39

Constructors - 4

```
public Person(String n, int a)
{
    name = n;
    age = a;
}

public Person()
{
    name = "Unset";
    age = 21;
}
```




39

Title of Topic: Topic 1 - 1.40

Constructors - 5

- This is called **method overloading** and is performed by giving a method the same name as one that already exists, but a different **method signature**.
- The method signature is made up of the order and type of parameters passed into the function. Now, the following will work:

```
private void FormClassExample_Load(object sender, EventArgs e)
{
    p1 = new Person("Michael", 100);
    p2 = new Person("Pauline", 20);
    p3 = new Person();
}
```




40


Title of Topic: Topic 1 - 1.41

Constructors - 6

- Run the program with this code and you'll see that we don't need to store data to begin with
- We can give ourselves some starting test data that makes the program easier to work with.



41



Awarding
Great British
Qualifications

Topic 5: Object Orientation (1)

Any Questions?

42
