Visual C# Intermediate

Sample manual - first two chapters

Manual 1110 - 123 pages –

TABLE OF CONTENTS (1 of 3)

TABLE OF CONTENTS (2 of 3)

TABLE OF CONTENTS (3 of 3)

CHAPTER 1 - USEFUL SHORT-CUT KEYS

1.1 The Best Short-Cut Keys in Visual Studio

To make sure that the most useful short-cut keys don't get buried in the rest of your courseware, this chapter summarises them.

Going to the definition of a variable or member

You can press the F12 key to go from a variable or method to its definition:

```
private void btnClick_Click(object sender,
{

    int firstNumber = 2;
    int lastNumber = 2;

    int answer = owlSum(firstNumber, lastN

    MessageBox.Show("The sum is " + answer

}
int owlSum(int a, int b)
{
    return a + b;
}
```

Pressing F12 with your cursor in this method will take you to its definition (even if this is in a different file).

```
private void btnClick_Click(object sender, EventArgs e)
{

    int firstNumber = 2;
    int lastNumber = 2;

    int answer = owlSum(firstNumber, lastNumber);

    MessageBox.Show("The sum is " + answer.ToString());

}
int owlSum(int a, int b)
{
    return a + b;
}
```

Pressing F12 here will take you to the definition of the variable (this won't necessarily be in the same procedure, although it is here).

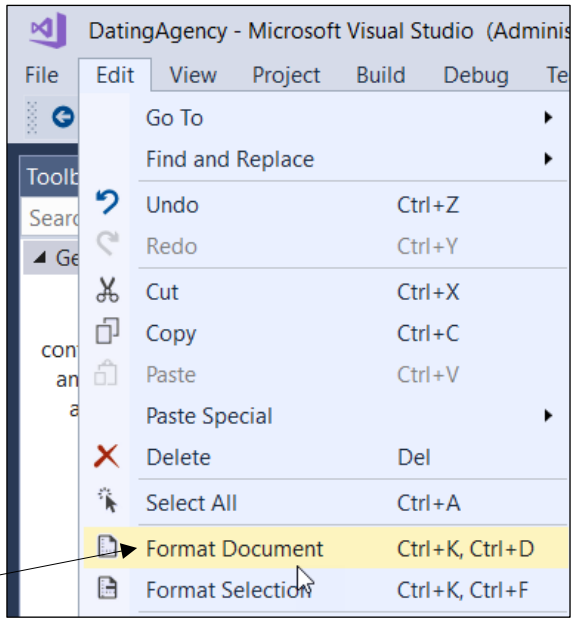
Going forward and backward using the keyboard

These two incredibly useful keys act like the < Back button in a browser:

Key	What it does
Ctrl + -	Takes you back to your last used location (whether in the same file or a different one), if necessary opening a window to show this.
Ctrl + Shift + -	Takes you forward (in effect, this counteracts what Ctrl + - does).

Auto-formatting text

This incredibly useful key combination tells Visual Studio to format code. To use it, press Ctrl + K followed by Ctrl + D :



The menu command that this shortcut key corresponds to.

Here's an example of an effect from pressing these keys:

```
namespace DatingAgency
{
    public partial class frmWiseOwl : Form
    {
        public frmWiseOwl()
        {
            InitializeComponent();

        }

    private void btnClick_Click(
        object sender, EventArgs e)
    {

    // get two numbers
    int firstNumber = 2;
    int lastNumber = 2;

    int answer = firstNumber + lastNumber;

    MessageBox.Show(
        "The sum is " + answer.ToString());

    }
}
```

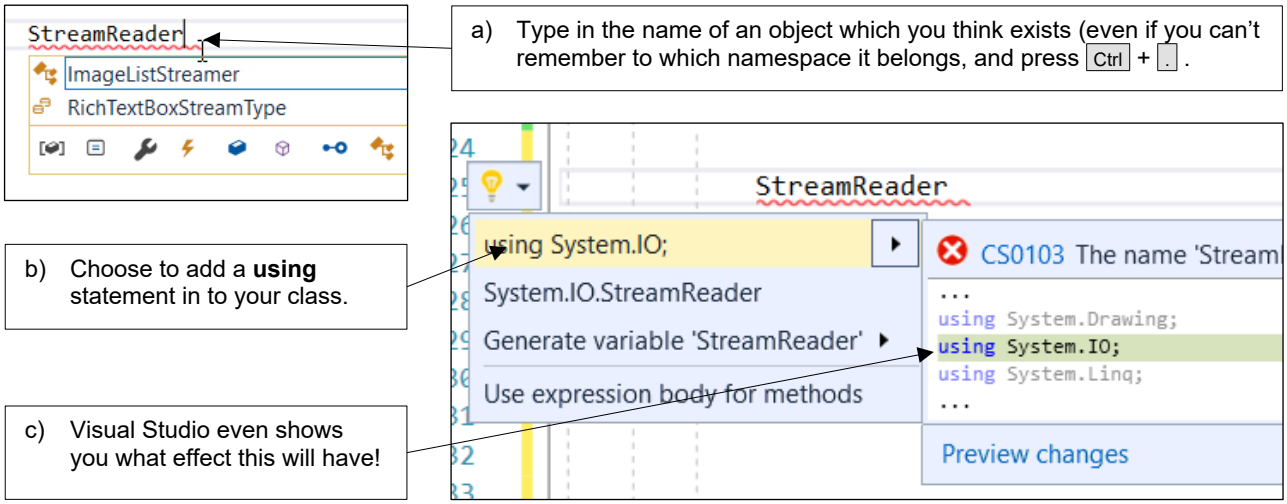
From disorder (the lines aren't indented well) …

```
public partial class frmWiseOwl : Form
{
    public frmWiseOwl()
    {
        InitializeComponent();

    }

    private void btnClick_Click(
        object sender, EventArgs e)
    {

        // get two numbers
        int firstNumber = 2;
        int lastNumber = 2;

        int answer = firstNumber + lastNumber;

        MessageBox.Show(
            "The sum is " + answer.ToString());

    }
}
```

… comes order (although most of the time Visual Studio will automatically indent code as you're typing it anyway).

Adding a Using statement

You can press Ctrl + . to reference a namespace:



CHAPTER 2 - DESIGNING CLASSES

2.1 Cats as Objects

Meet Niki! When the author was young, his family had a ginger tomcat. This section uses Niki to explain how classes work in programming.

Niki wasn't the sveltest of cats

Types, Classes and Objects

When Niki was born, God (whoever that may be) used a template to create a new cat object:

All cats follow the same basic template - 4 paws, whiskers, attitude – but there are also subtle differences between them which make each cat unique.

The cat template is a *class*, defining the rules that each cat object must follow. Niki was an *object* based on that class.

Wise Owl's Hint

An even more general word for a class is a type – more on this later in this courseware!

Instantiation and Termination

In any object's life, there are two main events:

Event	Technical name	Associated program to run
Birth	Instantiation	Constructor
Death	Disposal	Destructor

So when Niki was born, for example, God ran a *constructor* program to control what happened at the point at which Niki appeared in the world.

Properties

A cat (like any other object) has certain properties, each of which can be either *read-only*, *read-write* or *write-only*. Here are three examples:

Property	Type	Notes
Colour	*Read-only*	Once a cat has been *instantiated* (created), you can't change its colour (unless, of course, you temporarily dye it), so this is a read-only property. You can ask what colour a cat is, but you can't change this colour.
Mood	*Read-write*	It's usually obvious if a cat is unhappy (it arches its back or miaows), so you can read the value of this property. However, you can also change this property (feeding or kicking a cat are actions likely to change its mood immediately).
Wormed	*Write-only*	You can give a cat worming pills to worm it, but it's not possible to look at a cat and say whether it's been wormed or not – so this is a write-only property (you can change it, but not inspect it).

Methods

A *method* is something you do to an object. Here are some of the methods which you can apply to a cat object (and the *arguments* – or additional information – that you may need to supply):

Example method	Additional arguments
Feed	The amount and brand of food.
Stroke	The velocity of stroke and the area of the cat to which it is applied.
Kick	The area of the cat to which the kick should be applied.

The generic word for a property or a method is called a member *of the class.*

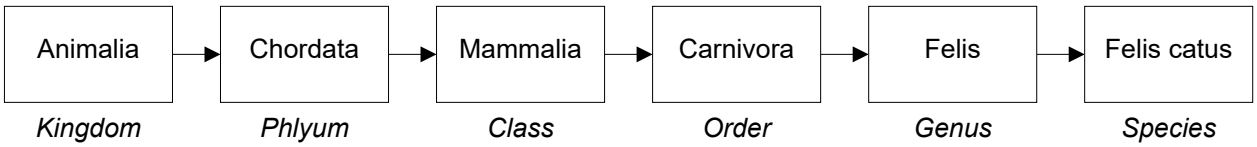
Encapsulation and Exposure

Cats *encapsulate* their logic, and only *expose* to the world certain methods and properties. Here are a couple of examples of *private* and *public* properties:

Member	Type	Scope	Notes
FurLength	*Property*	*Public*	You can look at a cat and see how long its fur is, so this is a property which the class exposes to the world.
LungCapacity	*Property*	*Private*	From a cat's point of view, its lung capacity is pretty important, but it's not something which is exposed to the world.

Inheritance

Domestic cats inherit from their species (*felis catus*), which in turn inherits from its genus, order, class, phylum and kingdom:

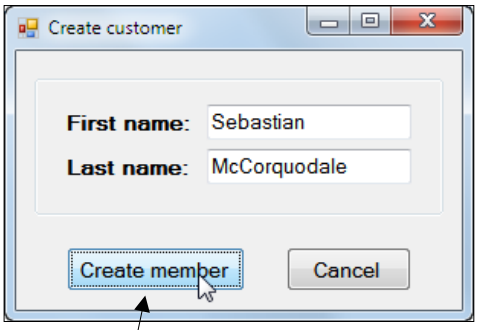


So (for example) the fact that a cat is warm-blooded is defined in the **Mammalia** class, and the domestic cat *inherits* (indirectly) from this *base class*.

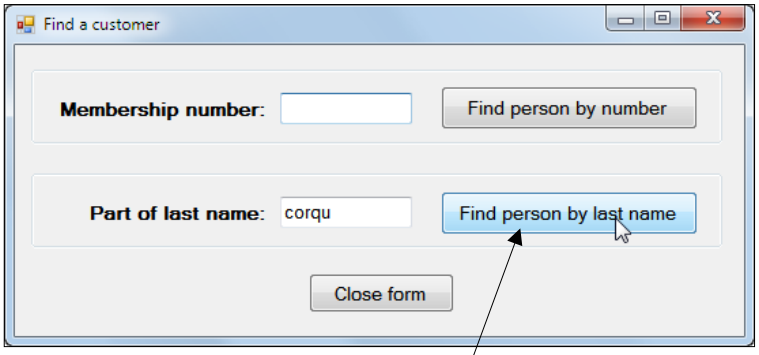
The Linnaeus classification of life (or taxonomy) is a perfect example of inheritance in action. It's not a coincidence that class is short for classification. Note that some animals can override *inherited characteristics (a platypus doesn't suckle its young, even though it's a mammal).*

2.2 Our Example – Dating Agency Customers

Imagine that you want to write a dating agency application (a very simple one!). Here are some of the forms you'll need:

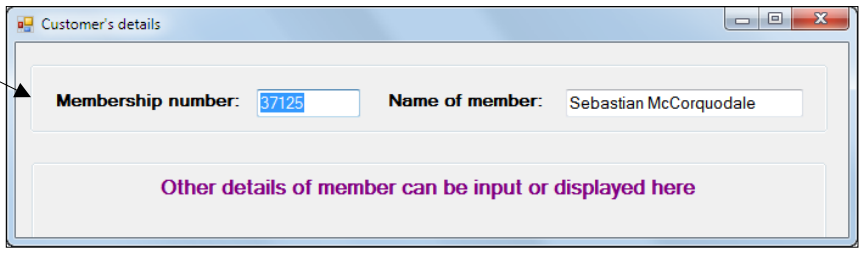


Whether you create a new customer by typing in their first and last name …



… or find an existing customer by typing in their membership number or a part of their surname …

… you'll then display their details in a separate form.



In real life, it's not names or numbers who join dating agencies: it's people. What object-orientated programming (*OOP*) allows you to do is to create and work with objects like this.

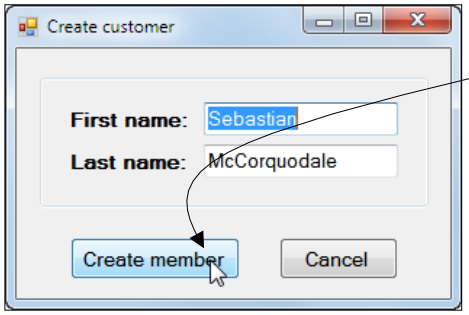
Our Customer Class

For our example we will create a *Customer* object (you could also create classes for dates, invoices, interests, events, matches and many more). Here are some suggested members:

Member	Type	Notes
FirstName	Read/write property	Set at the time a new object based on this class is instantiated, or created.
LastName		
FullName	Read-only property	Created by joining the first and last name together.
CustomerNumber	Read-only property	Assigned when this customer is first created
Greet	Method	Displays a message box on screen to say hello.

Envisaging how you will Consume a Class

The easiest way to design a class is to think how you'll *consume* it. Here's what our final code might look like:



When someone clicks on this button to create this member's details …

… our code should create a new **Customer** object, set his or her first and last name properties, then apply the **Create** method.

```
private void btnOK_Click(object sender, EventArgs e)
{
   // create a new customer
   Customer c = new Customer();

   // say what this person is called
   c.FirstName = txtFirst.Text;
   c.LastName = txtLast.Text;

   // add this person to list of customers
   c.Create();

}
```

What we do!

		Basic training	Advanced training	Systems / consultancy
Office	Microsoft Excel	✓	✓	✓
	VBA macros	✓	✓	✓
	Office Scripts	✓		
	Microsoft Access	✓	✓	✓
Business Intelligence	Power BI	✓	✓	✓
	Power Apps	✓		
	Power Automate / PAD	✓		
SQL Server	SQL	✓	✓	✓
	Reporting Services	✓	✓	✓
	Report Builder	✓	✓	✓
	Integration Services	✓	✓	✓
	Analysis Services	✓		
Coding	Visual C# programming	✓	✓	✓
	VB programming	✓	✓	✓
	DAX	✓	✓	✓
	Python	✓	✓	

WiseOwl Training