



Visual C# Intermediate

Sample manual - first two chapters



TABLE OF CONTENTS (1 of 3)

1	USEFUL SHORT-CUT KEYS	Page
1.1	The Best Short-Cut Keys in Visual Studio	5
	<i>Going to the definition of a variable or member</i>	5
	<i>Going forward and backward using the keyboard</i>	5
	<i>Auto-formatting text</i>	6
	<i>Adding a Using statement</i>	7

2	DESIGNING CLASSES	Page
2.1	Cats as Objects	8
	<i>Types, Classes and Objects</i>	8
	<i>Instantiation and Termination</i>	8
	<i>Properties</i>	9
	<i>Methods</i>	9
	<i>Encapsulation and Exposure</i>	10
	<i>Inheritance</i>	10
2.2	Our Example – Dating Agency Customers	10
	<i>Our Customer Class</i>	11
	<i>Envisaging how you will Consume a Class</i>	11

3	CREATING CLASSES	Page
3.1	Creating a Class	13
3.2	Namespaces	13
	<i>Example of a Namespace</i>	14
	<i>The Using Statement</i>	14
	<i>Removing Unused Using Statements</i>	14
	<i>Giving Aliases to Namespaces</i>	15
	<i>Our DatingAgency Namespace</i>	15
3.3	Creating a Constructor	15
	<i>Syntax of a Constructor</i>	16
	<i>Example of a Constructor</i>	16
3.4	Fields and Properties	17
	<i>Creating Fields</i>	18
	<i>Properties</i>	18
	<i>Refactoring (encapsulating) fields</i>	18
	<i>The Quickest and Best Way to Create Properties</i>	20
	<i>Properties which Perform Other Logic</i>	21
3.5	Methods	21
	<i>Void Methods</i>	22
	<i>Methods which Return Values</i>	22
	<i>Choosing between a Property and a Method</i>	23
3.6	Static Properties and Methods	23
	<i>Example of a Static Property</i>	24
	<i>Example of a Static Method</i>	24

4	FORMS AND CLASSES	Page
4.1	Anatomy of a Form	26
	<i>Partial Classes</i>	26
	<i>The Form's Constructor</i>	27
	<i>Drawing a Form</i>	27
4.2	Instantiating Forms	28
	<i>Setting the Start-Up Form</i>	29
	<i>Showing a Form during an Application</i>	29

5	OVERLOADING	Page
5.1	Overloading	30
5.2	Creating Overloaded Methods	30
	<i>Consuming the method</i>	31
	<i>Creating our FIND method</i>	31
5.3	Overloading Constructors	32

6	INHERITANCE	Page
6.1	The Concept	34
6.2	Existing Classes in .NET	34
6.3	Inheriting from Existing Classes	35
	<i>The code for this example</i>	36
6.4	Creating your own Hierarchy	37
	<i>Our example – arranging dates between customers</i>	38
	<i>How to inherit classes – syntax</i>	38
	<i>Calling the base constructor</i>	39
	<i>Creating protected members and fields</i>	39
6.5	Overriding Properties	40
	<i>The problem: duplicate member name</i>	41
	<i>Two possible solutions: NEW versus OVERRIDE</i>	41
	<i>Using NEW to solve our problem</i>	41
	<i>Using OVERRIDE and VIRTUAL to solve our problem</i>	42
6.6	Overriding Methods	42
	<i>Calling the BASE method</i>	43
6.7	Sealed Classes and Members	43
	<i>Sealing classes</i>	44
	<i>Sealing methods and properties</i>	44
6.8	Abstract Classes and Members	44

TABLE OF CONTENTS (2 of 3)

7	VALUE AND REFERENCE	Page
7.1	Types of Memory (Stack and Heap)	46
7.2	Types of Variables	46
	<i>Direct Variables</i>	47
	<i>Indirect Variables</i>	47
7.3	Boxing and Unboxing	47
	<i>System.Object</i>	48
	<i>Boxing</i>	48
	<i>Unboxing</i>	48
7.4	Passing by Value and Reference	49
	<i>Arguments are passed by value by default</i>	50
	<i>Passing arguments by reference using REF or OUT</i>	50

8	ENUMERATIONS	Page
8.1	Creating and Using Enumerations	51
8.2	Customising Enumerations	51
	<i>Enumeration Aliases</i>	52
	<i>Changing Enumeration Integer Values</i>	52
	<i>Changing the Enumeration Data Type</i>	52
8.3	Looping over Enumerations	52

9	STRUCTURES	Page
9.1	Overview of Structures	54
9.2	Differences between Structures and Classes	54
	<i>Structures are Value Types</i>	55
	<i>Other Differences</i>	55
9.3	Familiar Structures!	56

10	ARRAYS	Page
10.1	Arrays	58
	<i>Creating single-dimensional arrays</i>	58
	<i>Populating arrays and retrieving items</i>	58
	<i>Looping over arrays</i>	58
	<i>Multi-dimensional arrays</i>	59

11	LISTS	Page
11.1	Overview of Lists	60
	<i>An Example of a List</i>	60
11.2	Working with Lists	60
	<i>Creating a List</i>	61
	<i>Adding Items to a List</i>	61
	<i>Counting the Items in a List</i>	61
	<i>Displaying All of the Items in a List (FOR EACH)</i>	62
	<i>Removing Items from a List</i>	62
	<i>Finding items in a list</i>	62
	<i>Lambda Expression Syntax for Find Methods</i>	63
11.3	Getting a Subset of a List	64
	<i>Method 1: Using FindAll</i>	64
	<i>Getting a Subset of a List – Method 2: Using GetRange</i>	64
11.4	Joining and Splitting String Lists	64

12	STACKS AND QUEUES	Page
12.1	Queues	66
12.2	Stacks	66

13	DICTIONARIES	Page
13.1	Key/Value Pairs	68
13.2	Our Example – the Customer Class	69
13.3	Working with Dictionaries	70
	<i>Creating a Dictionary</i>	70
	<i>Adding to a dictionary</i>	70
	<i>Removing from a Dictionary</i>	71
	<i>Accessing Dictionary Values</i>	71
	<i>Determining if a key exists</i>	71
13.4	Looping over dictionary items	72
	<i>Looping by Key Value Pair</i>	72
	<i>Looping by Key Only</i>	72
	<i>Looping by Value Only</i>	72
13.5	pe	72

14	DATA TABLES	Page
14.1	Overview of Data Tables	74
	<i>Referencing System.Data</i>	74
	<i>How data tables work</i>	74
14.2	Working with Data Tables	75
	<i>Creating a Data Table</i>	75
	<i>Adding rows</i>	75
	<i>Looping over rows to retrieve data</i>	76

TABLE OF CONTENTS (3 of 3)

15	GETTING STARTED WITH LINQ	Page
15.1	What is LINQ?	77
	<i>Referencing the LINQ Namespace</i>	77
15.2	Anatomy of a LINQ Query	77
15.3	Implicit and Explicit Variable Types	78
	<i>LINQ queries are compiled</i>	79
	<i>Implicit variable types</i>	79
	<i>The case for explicit variable types</i>	79
15.4	Examples for Different Enumerable Sets	80

16	LINQ SYNTAX	Page
16.1	Our Example	83
16.2	The SELECT keyword	83
	<i>Transformations</i>	84
16.3	Projections using Anonymous Types	84
	<i>Creating anonymous types</i>	85
	<i>Using LINQ to project data onto anonymous types</i>	85
	<i>Using anonymous types to merge data</i>	86
16.4	Taking and Skipping	87
16.5	Forcing Query Execution	88
16.6	Ordering a Sequence (ORDERBY)	89
16.7	Filtering (WHERE)	90
	<i>Calling methods in where clauses</i>	92
16.8	Adding Expressions (LET)	92
	<i>Example: listing primes</i>	93

17	TYPES OF DATA MODEL	Page
17.1	Our Example	94
	<i>The 3 Types of Data Model</i>	94
17.2	Code First Models	95
	<i>Creating the Table Classes</i>	95
	<i>Creating the Database Class</i>	95
	<i>Creating the Database</i>	96
	<i>Viewing the Database</i>	96
17.3	The Model First Approach	97
17.4	Database First	98
17.5	Thoughts on which Approach to Use	99
	<i>Do you have a Database?</i>	99
	<i>How do you Change your Database?</i>	99
	<i>Do you like Wizards?</i>	100
	<i>Our Recommendation: Database First</i>	100

18	DATABASE FIRST MODELS	Page
18.1	Creating a Model	101
	<i>Step 1 - Creating the Database</i>	101
	<i>Step 2 - Adding a New Item</i>	101
	<i>Step 3 - Choosing the Model Type and Connection</i>	102
	<i>Step 4 - Choosing the EF Version</i>	103
	<i>Step 5 - Choosing the Entities for your Model</i>	103
	<i>Step 6 - Saving your Model</i>	104
18.2	Updating Models	105

19	LINQ AND ENTITY FRAMEWORKS	Page
19.1	Getting Data with LINQ	106
	<i>Creating a Data Context</i>	106
	<i>Selecting Data using LINQ</i>	106
	<i>Using Relationships</i>	107
19.2	Changing Data	108
	<i>Inserting Rows</i>	108
	<i>Deleting Rows</i>	109
	<i>Editing Rows</i>	109
19.3	Returning Anonymous Types	110
19.4	Working with Stored Procedures	111
	<i>Changing Stored Procedures</i>	112
19.5	Partial Classes and Entity Frameworks	113
	<i>Creating a partial class</i>	113
	<i>Partial classes don't work with LINQ</i>	114
19.6	Joining Tables	115

20	GROUPING IN LINQ AND EF	Page
20.1	Basic Grouping	116
	<i>How grouping works</i>	116
20.2	Grouping into Ranges	117
	<i>Grouping films by their initial letter</i>	118
	<i>Grouping customers by their decade of birth</i>	118
20.3	Grouping Into and Ordering	119

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).

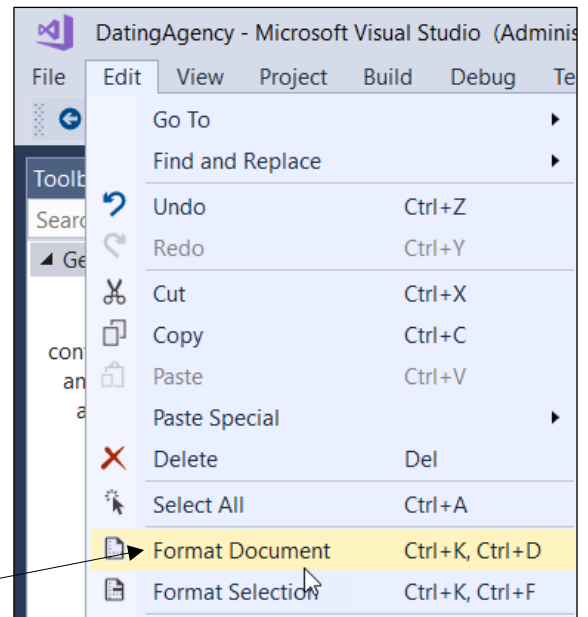
Going forward and backward using the keyboard

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;

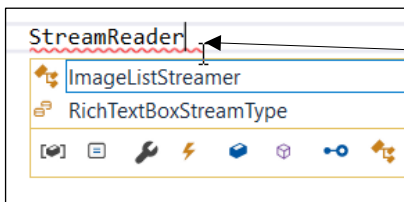
        int answer = firstNumber + lastNumber;

        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

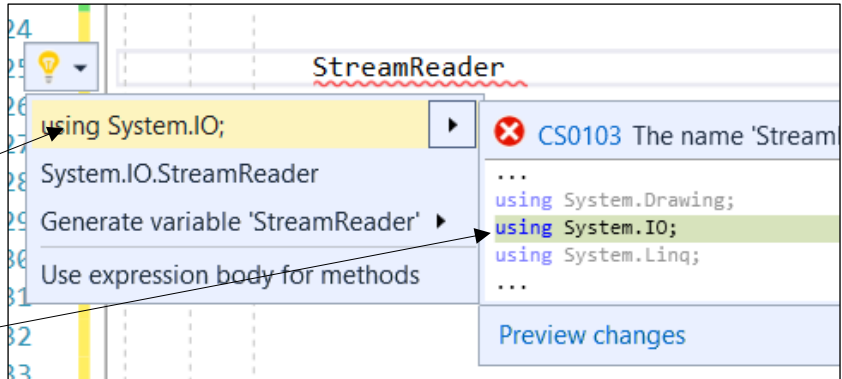
You can press **Ctrl** + **.** to reference a namespace:



a) Type in the name of an object which you think exists (even if you can't remember to which namespace it belongs, and press **Ctrl** + **.** .

b) Choose to add a **using** statement in to your class.

c) Visual Studio even shows you what effect this will have!



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.



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
<i>Birth</i>	Instantiation	Constructor
<i>Death</i>	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	<i>Read-only</i>	Once a cat has been <i>instantiated</i> (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	<i>Read-write</i>	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	<i>Write-only</i>	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
<i>Feed</i>	The amount and brand of food.
<i>Stroke</i>	The velocity of stroke and the area of the cat to which it is applied.
<i>Kick</i>	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.

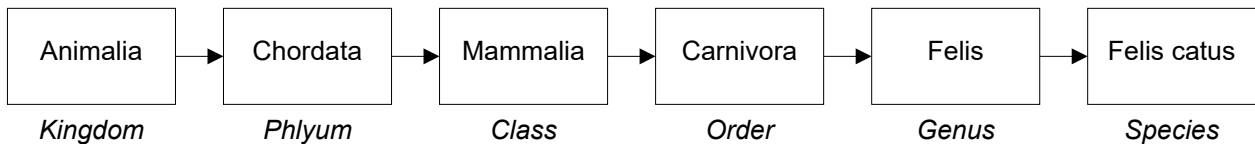
Encapsulation and Exposure

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
<i>FurLength</i>	<i>Property</i>	<i>Public</i>	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.
<i>LungCapacity</i>	<i>Property</i>	<i>Private</i>	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.

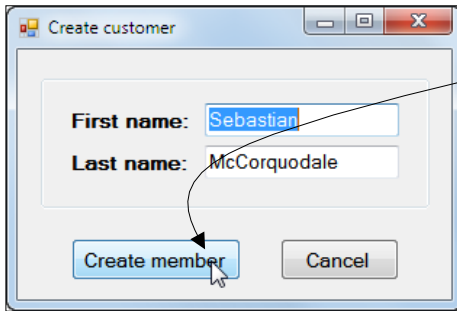
Our Customer Class

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
<i>FirstName</i>	Read/write property	Set at the time a new object based on this class is instantiated, or created.
<i>LastName</i>		
<i>FullName</i>	Read-only property	Created by joining the first and last name together.
<i>CustomerNumber</i>	Read-only property	Assigned when this customer is first created
<i>Greet</i>	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

