

**Name:**

**Course:** Introduction to .NET

**Title:** Introduction

**Instructor:** Bill Buchanan

 **NAPIER UNIVERSITY**  
EDINBURGH SCOTLAND



- Module 1** Introduction to Object-Orientation, Introduction to .NET, Overview of .NET Framework, .NET Components. C#. Introduction to Visual Studio Environment.
- Module 2** Variables and Types, Naming Variables, Using Built-In Data Types, Visual Studio Environment: Design Surface - dynamic help; Toolbox; Code Page; Properties Window; Solution Explorer.
- Module 3** Converting Data Types, Control structures: if, for, while, foreach, switch.
- Module 4** Afternoon – File handling – looking up details from help, Methods, Parameter passing, Variables and scope.
- Module 5** Review of Day 2, Converting Data Types, Control structures: if, for, while, foreach, switch.
- Module 6** Afternoon – Brief description of Arrays, Type Casts, Overview of System.Collections, ArrayLists and HashTables, Sorting, Enumerators.
- Module 7** Using Reference-Type Variables , Using Common Reference Types, The Object Hierarchy, Namespaces in the .NET Framework, Debugging techniques, breakpoints, line stepping, call stack, locals window.
- Module 8** Classes and Objects, Using Encapsulation, C# and Object Orientation, Defining Object-Oriented Systems, Deriving classes, Implementing classes, Private, public, internal and protected, Using Interfaces, Abstract classes, Using Constructors Initializing Data, Objects and Memory, Resource Management, Object Browser, Class View.
- Module 9** Methods and overloaded methods, Exceptions, Intellisense and overloaded methods, Enumerations and string → enumeration conversion.
- Module 10** Using Modules and Assemblies, XML (read only), Serialising classes.

Timetable:

	Mon 21 March	Tue 22 March	Wed 23 March	Thu 24 March	Fri 25 March	Mon 28 March	Tue 29 March	Wed 30 Mar	Thur 31 Mar	Fri 1 April
<b>Morning</b>	Mod 1	Mod 3		Mod 5				Mod 7	Mod 9	
<b>Afternoon</b>	Mod 2	Mod 4		Mod 6				Mod 8	Mod 10	

# 1 Introduction

## 1.1 Introduction

---

The first module provides a basic introduction to the .NET. It covers:

- Visual Studio Environment.
- Introduction to Object-orientation.
- .NET Framework.
- Benefits of C# over VB.
- .NET Components.
- .NET Languages.

.NET uses a newly developed language named C#, which has evolved through C++, and has adopted many of the object-oriented ideas from Java. The Visual Studio environment provides a complete system for many development areas, including:

- ◆ **Console applications.** These are created in a command window, and are excellent for the text-based programs, such as utility programs.
- ◆ **Windows applications.** This provides applications which use standard Windows elements, such as buttons, list boxes, data grids, and so on.
- ◆ **Windows services.** These are background processes which run silently can provide services to the user, such as supporting print facilities.
- ◆ **Web applications.** This include Web page design, which is enhanced with C# code which runs behind the page. This allows increased usability and interaction over normal Web pages. The standard Web page format used on Microsoft Web servers are ASP files.
- ◆ **Web services.** These are service programs which run on Web servers.
- ◆ **Mobile applications.** This supports the development for hand-held and mobile devices.

The .NET Framework is a standard software installation for Windows XP, Windows 2000 and Windows NT, and supports robust applications, and increased security. This can be used with the Visual Studio 2003 environment which provides an environment for the rapid development of applications. Software development, in the past, has been error prone, especially in that there was no guarantee that programs would run in a predictable way, especially as many of the resources required for the

program had to be determined once the application was executed. A key factor in Visual Studio 2003 is that it tries to aid the development with intelligent predicting on the syntax and usage of the system elements. These should lead to applications which are more robust, and which are free from run-time errors.

## 1.2 .NET Framework

---

The PC has slowly evolved with ever-increasing operating system components. Unfortunately, these additions make version control difficult. Microsoft has thus tried to overcome this by creating a common framework in which programs run. This framework should be easier to control and to update, where bug fixes can be updated with news downloads. Generally, Microsoft have further developed object-orientation, and integrated many of the new techniques that Java created, such as operating system/hardware independence. They have also, for the first time, binded the .NET Framework to the Windows environment, which should improve security and allow enhanced integration with system components, such as using Microsoft Word as an editor, or Microsoft Excel to create and process spreadsheets.

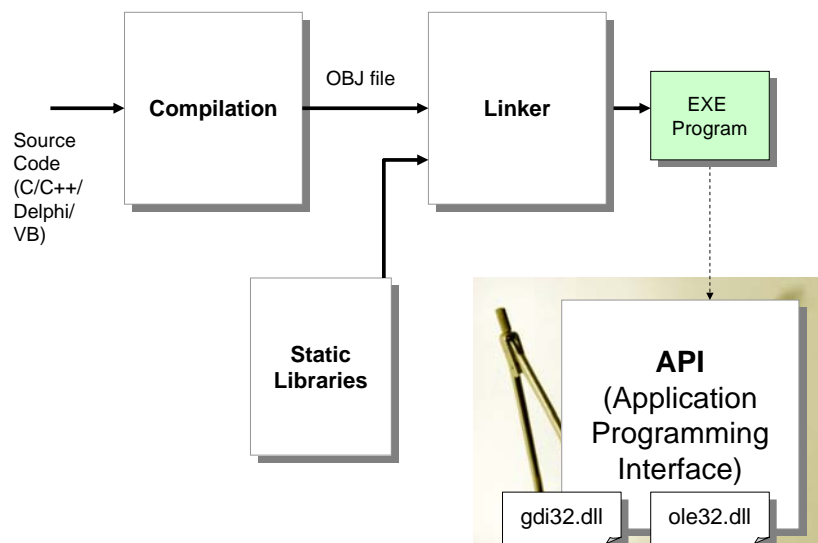
Figure 1.1 shows the typical steps taken in developing a program. Initially the source code is converted into object code (OBJ) using a **compiler**. This converts the code into a form which is matched to the hardware of the system, but cannot be run as it does not contain the basic elements of a program, such as routines which interface to the input/output. These elements are integrated with the **linker**, which takes all the created object code files, and searches in the static libraries for any code that is required to produce the executable program. The program can then access system routing through API (Application Programming Interface) calls, such as in creating Windows, or interfacing to networking functions. These API calls are typically contained in run-time libraries (DLL's), which contain the code which implements the require function.

### 1.2.1 Win32 API

The Win32 API library contains many routines:

- **Creating windows.**
- **Windows support functions.**
- **Message processing.**
- **Menus.**
- **Resources.**
- **Dialog boxes.**
- **User input functions.**
- **Memory management.**
- **GDI (graphical device interface).**
- **Bitmaps, icons and metafiles.**
- **Printing and text output.**
- **Painting and drawing.**
- **File I/O.**
- **Clipboard.** Support for public and

- private clipboards.
- **Registry.** Support for functions which access the Registry.
- **Initialization files.** Support for functions which access INI files.
- **System information.**
- **String manipulation.**
- **Timers.**
- **Processes and threads.**
- **Error and exception processing.**
- **Help files.**
- **File compression/decompression.**
- **DLLs.**
- **Network support** (NetBios and Windows sockets 1.1 APIs).
- **Multimedia support** (sound APIs).
- **OLE and DDE** (dynamic data exchange).
- **TrueType fonts.**



**Figure 1.1:** Traditional stages of program development

### 1.2.2 .NET Enhancements

Software development for the PC has developed through the use of languages such as C++, Delphi and Visual Basic. These languages are all focused on producing Windows-based programs using the x86 architecture. The computing industry, though, has moved over the last few years, with the advent of the Internet and Java. With Java, Sun Microsystems have tried to produce programs which could be run on any type of hardware. This has the advantage that the program can be produced for a range of hardware, including Apple-based computers and hand-held devices. There has also been a move toward using software routines which are not necessarily based on the host computer, and in integrating WWW services with programs. This integration is illustrated in Figure 1.2.

In general the current weaknesses of software development which have been overcome with .NET are:

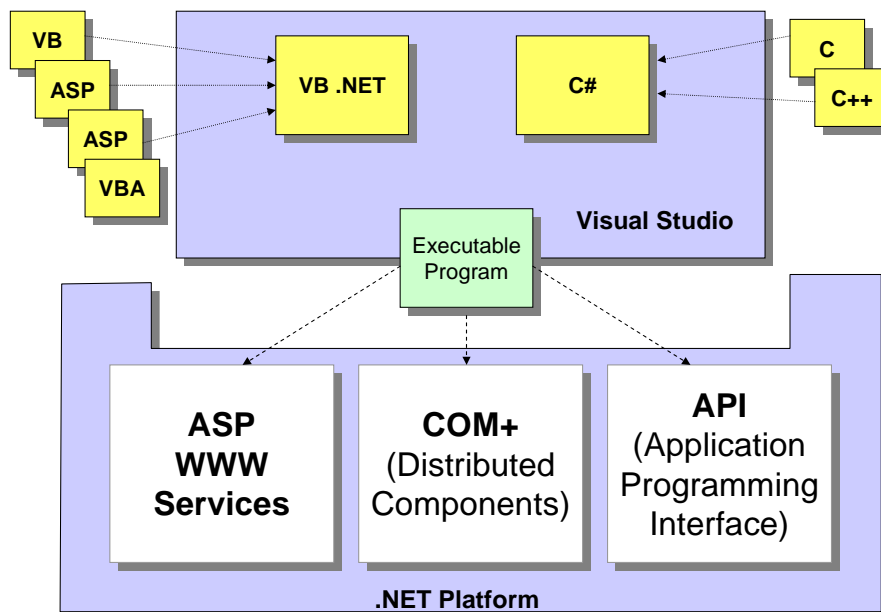
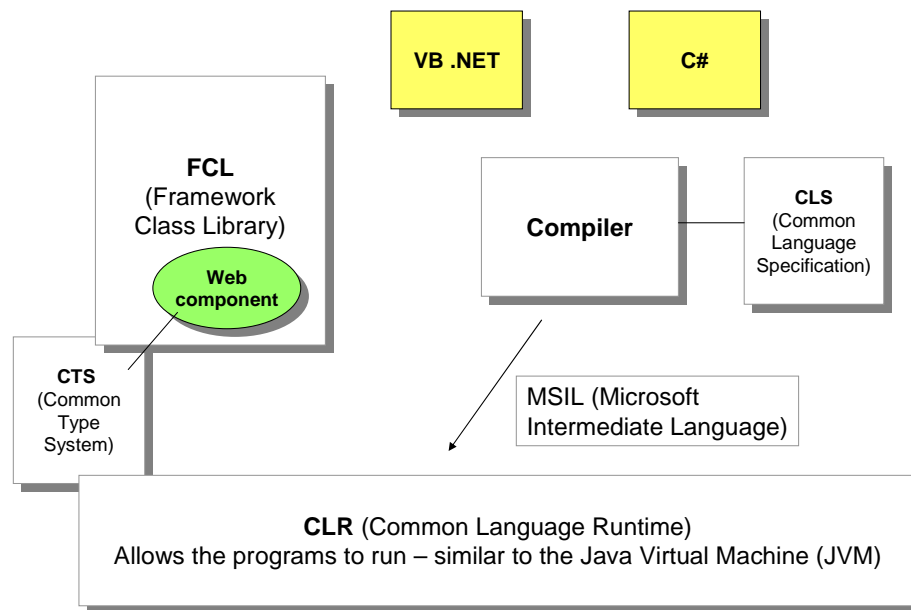


Figure 1.2: .NET Integration

- **Lack of support for different hardware.** .NET overcomes this in a similar way to Java in that it produces an intermediate code known as MSIL (Microsoft Intermediate Language) which when run with the CLR (Common Language Runtime) produces an executable program which matches the hardware.
- **Difficult to integrate different programming languages.** This is a problem with most software development environments, especially in representing the variables in the same way. .NET overcomes this by producing a standardized system for data representation, known as CTS (Common Type Representation). The compiler also uses CLS (Common Language Specification) to produce code which can be integrated with other programming languages. Along with this the .NET framework uses a FCL (Framework Class Library) which is a common set of classes which can be used with the different languages.
- **Lack of security integration.** .NET binds itself with the operating system so that it integrates better with the security controls of the system.
- **Poor version control of system components.** .NET improves this by supporting the installation of different versions of the .NET framework. Users can choose which of the version they install, and previous versions will be stored on the host.
- **Weak integration with the WWW/Internet.** .NET integrates WWW development with code development by integrating ASP with VB/C#.

In order to support these advancements, Microsoft has developed Visual Basic further and integrated it with ASP, to produce VB.NET. C++ has also been advanced to C# which incorporates many of the advanced features of Java. Figure 1.3 outlines some of the key features of the .NET framework.



**Figure 1.3:** .NET Framework

### 1.2.3 .NET Environment

DLL's are typically installed in the system folders of the host. It is thus difficult to keep track of new updates which enhance features or to fix bugs, as system files where often overwritten by new ones. One method that was used to support these updates was Direct X, which allowed software components to be registered onto a system (Figure 1.4). Thus, when a program required a service, such as network support, it would call the required DLL.

The problem of version control has now been reduced using the .NET framework. With this the key framework files are installed to a single folder. An example of this is shown in Figure 1.5. It can be seen that the single folder makes it easier to update with new versions. A key DLL is the Mscorlib.dll which contains many of the key elements of the .NET framework, such as for file I/O, system interfacing and security.

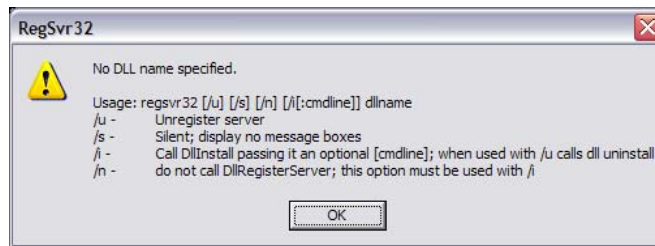


Figure 1.4: DLL registration

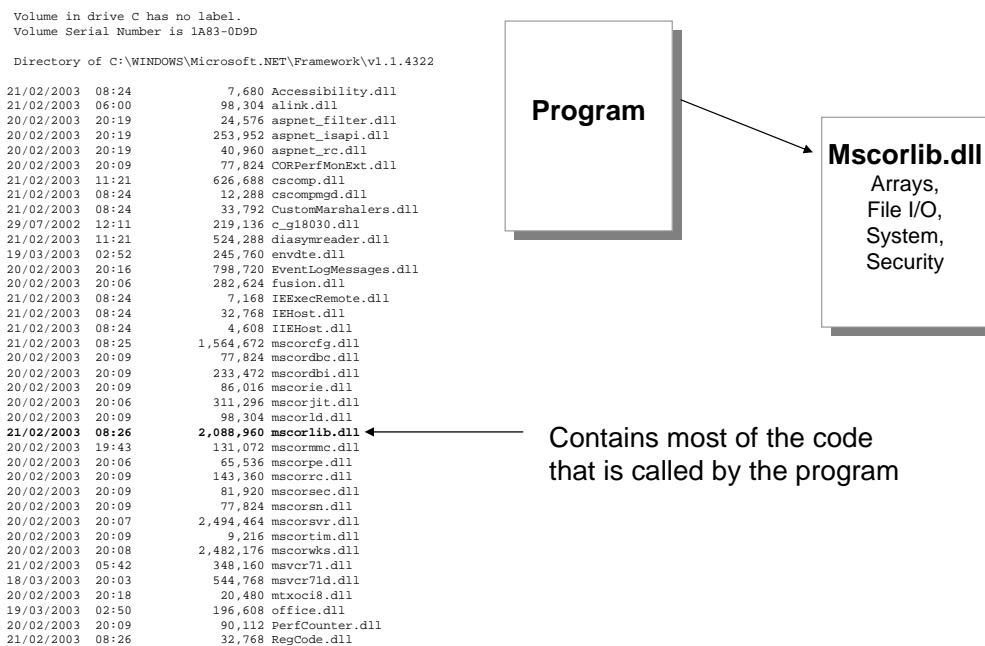


Figure 1.5: .NET framework installation folder

## 1.3 Visual Studio .NET Environment

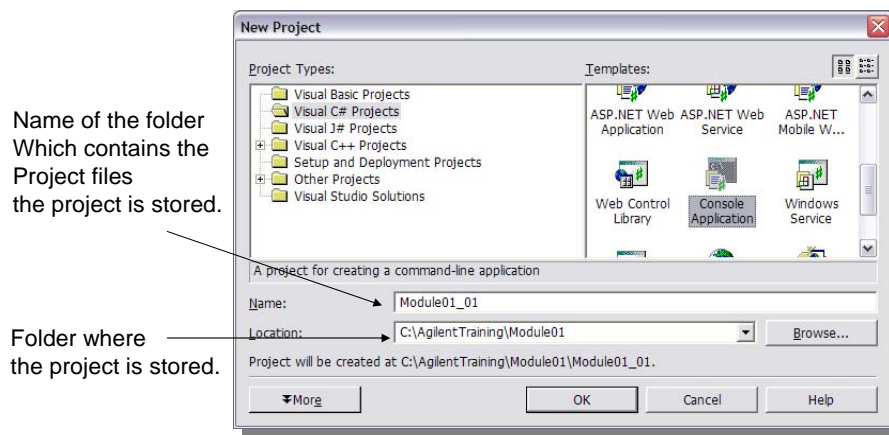
The Visual Studio .NET Environment provides an integration of the Visual Studio environment with new features, especially in automated code generation, and intelligent sensing of user requirements. It uses projects (VBP) or solutions (SLN) to create folders which contains the required elements of the program. In creating a new project (with **File**→**New**→**Project**), the user is then asked for the required application (Figure 1.6). This include a C# or a VB project. Once selected the user then selects a template for the project, such as:

- **Console application.** This uses a command line window for the output, and is useful in applications which require text-based information.



- **Windows application.** This uses type of application uses Windows for its interface, such as with text boxes and menus.

Figure 1.7 shows an example of the Visual Studio C# .NET environment. The project elements are stored in the Solution explorer window. The actual code is displayed in the Code and Text Editor window. For C# programs, the file name has a .cs extension. A useful feature of the environment is the editor's statement completion which aids the developer by completing the required statement, as well as given help on its syntax.



**Figure 1.6:** Creating a project

### 1.3.1 Solution files

The solution file contains of the required elements of a project. The information is contained in an sln file. In Figure 1.7, the solution file is named **Module01\_01.sln**, will the contents of the file is displayed in Figure 1.8. It can be seen that it defines the environment of the development system, and defines a project file, which is defined with a csproj file extension. In this case, the project file is named **Module01\_01.csproj**. The project file contains the references to all the required files that the solution uses. In this case, as shown in Figure 1.9, the files used are App.ico, AssemblyInfo.cs and Class1.cs. The C# code is contained in cs files, which are known as class files, and is illustrated in Figure 1.10.

Along with this, most projects contain an AssemblyInfo class file, an example of which is shown in Figure 1.11. .NET uses assembly to represent a single unit. An assembly, to be covered in a future unit, is a collection of files that appear as a single unit, such as a single DLL or an EXE. Along with class files, most projects contain an icon file, which is associated with the executable program. An example of an icon is shown in Figure 1.13.

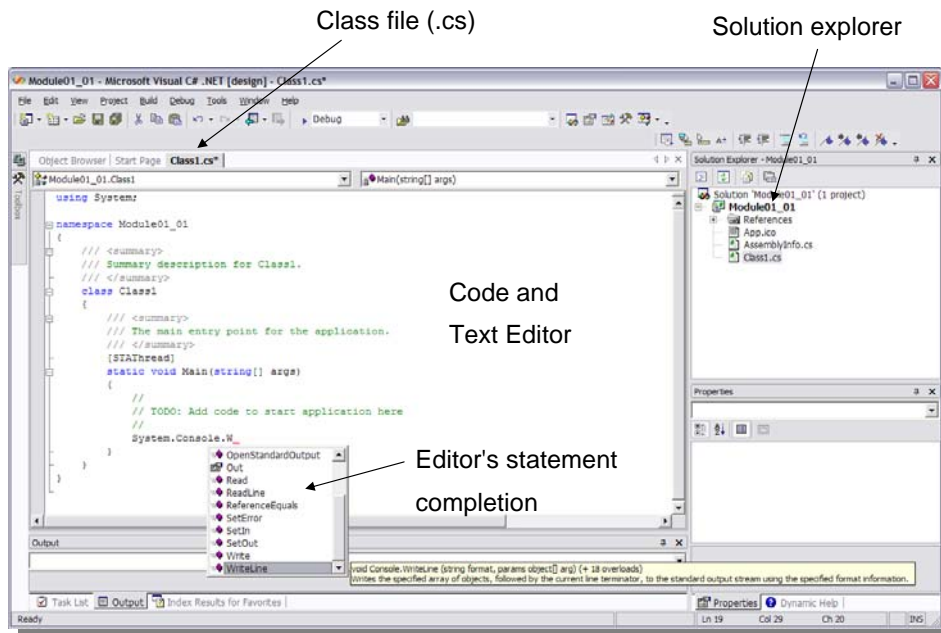


Figure 1.7: Visual Studio C# .NET environment

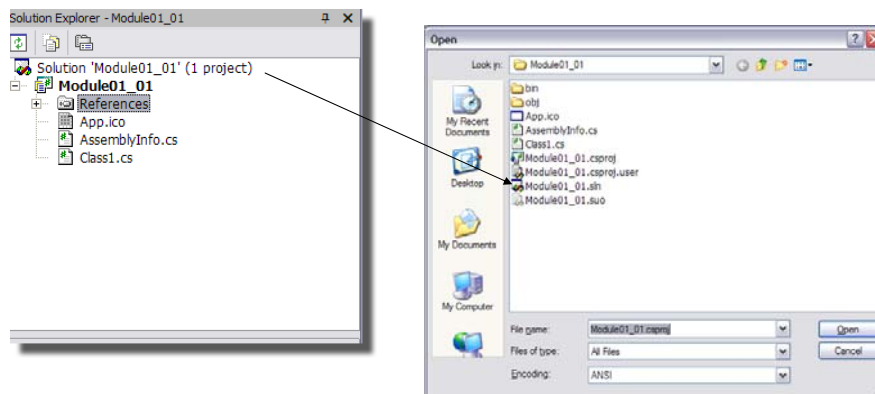
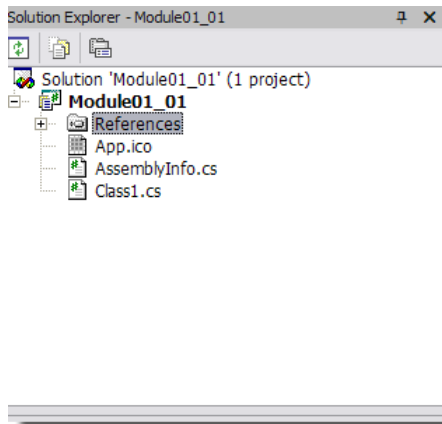


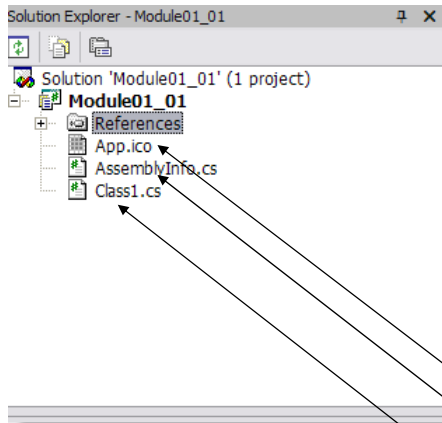
Figure 1.8: Example listing



```

Module01_01.sln
Microsoft Visual Studio Solution File, Format Version 8.00
Project("{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}") = "Module01_01",
  "Module01_01.csproj", "{1EA384DD-927E-4B94-ABB8-BA08DFAE5AB3}"
  ProjectSection(ProjectDependencies) = postProject
  EndProjectSection
EndProject
Global
GlobalSection(SolutionConfiguration) = preSolution
  Debug = Debug
  Release = Release
EndGlobalSection
GlobalSection(ProjectConfiguration) = postSolution
  {1EA384DD-927E-4B94-ABB8-BA08DFAE5AB3}.Debug.ActiveCfg =
  Debug|.NET
  {1EA384DD-927E-4B94-ABB8-BA08DFAE5AB3}.Debug.Build.0 =
  {1EA384DD-927E-4B94-ABB8-BA08DFAE5AB3}.Release.ActiveCfg =
  Release|.NET
  {1EA384DD-927E-4B94-ABB8-BA08DFAE5AB3}.Release.Build.0 =
  Release|.NET
EndGlobalSection
GlobalSection(ExtensibilityGlobals) = postSolution
EndGlobalSection
GlobalSection(ExtensibilityAddIns) = postSolution
EndGlobalSection
EndGlobal
  
```

Figure 1.9: Example sln file



```

Module01_01.csproj
<VisualStudioProject>
  <CSHARP
    ProjectType = "Local"
    ProductVersion = "7.10.3077"
    SchemaVersion = "2.0"
  >
  <Build>
    <Settings
      ApplicationIcon = "App.ico"
      AssemblyKeyContainerName = ""
      AssemblyName = "Module01_01"
      AssemblyOriginatorKeyFile = ""
      DefaultClientScript = "JScript"
      DefaultHTMLPageLayout = "Grid"
      DefaultTargetSchema = "IE50"
      DelaySign = "false"
      OutputType = "Exe"
      PreBuildEvent = ""
      PostBuildEvent = ""
      RootNamespace = "Module01_01"
      RunPostBuildEvent = "OnBuildSuccess" >
    <Config
      Name = "Debug"
      AllowUnsafeBlocks = "false"
    />
    <Config
      Name = "Release"
      AllowUnsafeBlocks = "false"
      BaseAddress = "285212672"
    />
  </Settings>
  <References>
</Build>
  <Files>
  <Include>
    <File RelPath = "App.ico" BuildAction = "Content" />
    <File RelPath = "AssemblyInfo.cs" SubType = "Code"
      BuildAction = "Compile" />
    <File
      RelPath = "Class1.cs" SubType = "Code"
      BuildAction = "Compile" />
  </Include>
  </Files>
</CSHARP>
</VisualStudioProject>
  
```

Figure 1.10: Project file

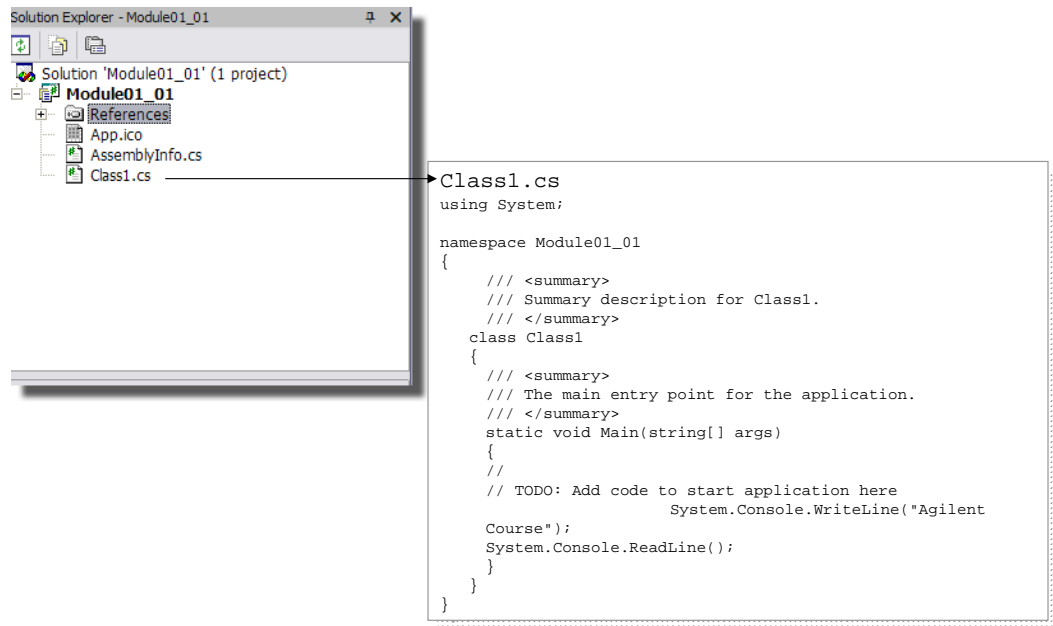


Figure 1.11: CS file

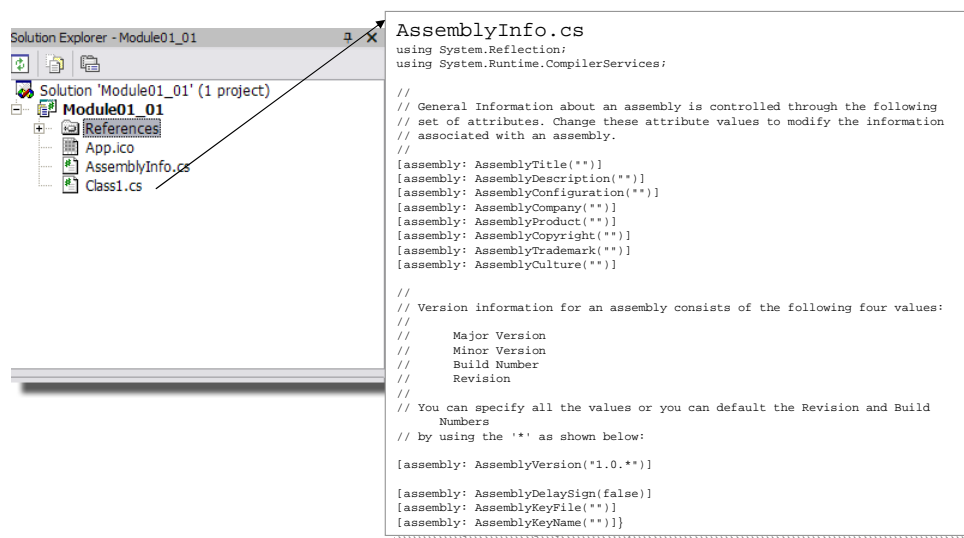


Figure 1.12: AssemblyInfo class file

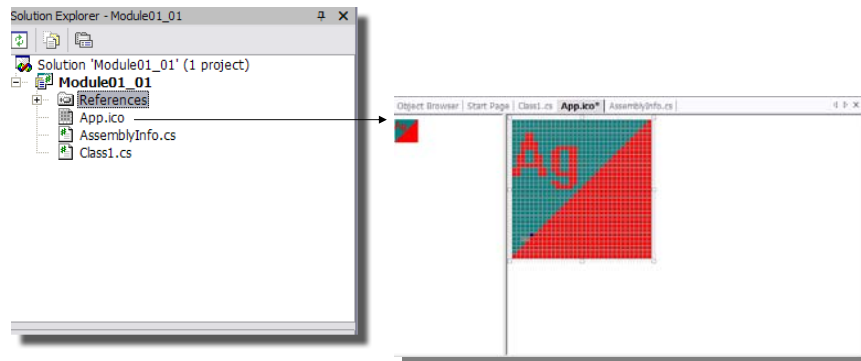


Figure 1.13: Icon file

## 1.4 Simple Console Application

In Visual Studio a new project is defined, based on one of the above, as shown in Figure 1.14. One of the basic projects is a Console Application, which typically supports user input with the `System.Console.ReadLine()` method, and supports output with the `System.Console.WriteLine()` method. A simple program is:

```
Program 1.1:  
using System;  
  
namespace ConsoleApplication1  
{  
    class Class1  
    {  
        static void Main(string[] args)  
        {  
            System.Console.WriteLine("This is my first program");  
            System.Console.ReadLine();  
        }  
    }  
}
```

which simply outputs a line of text, and waits for the user to press the <ENTER> key.

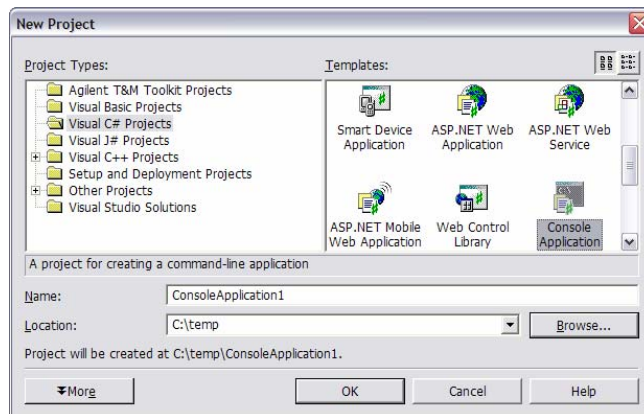


Figure 1.14: Console application

## 1.5 .NET Languages

The two main languages of .NET are VB.NET and C#. Example of their syntax are shown in Program 1.5 and 1.6. The gulf between VB and C++ was, at one time, a massive one, as C++ contained more error checking and had a strong syntax. With the advent of VB.NET, the gulf between VB and the new version of C++ (C#) has reduced. Both are fully object-oriented, and the weaknesses of VB have now been overcome. The true strength of C# is its simple and precise syntax which many professional developers prefer to VB.

📖 <b>Program 1.2:</b>	
<pre> \ VB.NET Code Dim j As Integer Dim prime As Boolean Dim i As Integer  For i = 1 To 100     prime = True      For j = 2 To (i / 2)         If ((i Mod j) = 0) Then             prime = False         End If     Next j     If (prime = True) Then         TextBox1.Text = TextBox1.Text &amp; "," &amp; Str(i)     End If Next i </pre>	
💻 <b>Sample Run 1.2</b>	
<p>1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97</p>	

**Program 1.3:**

```
// C# Code
int i, j;
bool prime;

for (i=0;i<100;i++)
{
    prime = true;

    for (j=2;j<=i/2;j++)
    {
        if ((i%j)==0) prime=false;
    }
    if (prime==true) textBox1.Text+=" " + Convert.ToString(i);
}

```

**Sample Run 1.2**

```
1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79,
83, 89, 97
```

Both languages have developed through different applications. VB has generally evolved through BASIC, and onto Windows programming. As it is relatively easy for novices to use it was integrated in Office applications, using VBA (Visual Basic for Applications). This allowed Office applications to use the power of VB in an Office environment. VB was also the basis of the server-side scripting language named ASP (Active Server Pages). For this VB.NET is generally considered as the best approach for Office and WWW-based applications.

C# has had a more nuts-and-bolts evolution, as C allowed the developers a great deal of freedom in their approach. For this the compiler generally *warned* the developer of possible violations, which the developer often ignored. Thus often led to run-time errors due to unforeseen bugs. C allows direct access to the memory. This door was finally closed with the advent of Windows NT, which barred direct access to memory. The next evolution of C was C++ which integrated the new object-oriented methods with C++. It was a missed opportunity, though, as it still allowed C programs to be written in the way they had always done. It was thus still a hybrid language. The rise of languages such as Java and the focus on mobility, WWW-based applications, and security caused Microsoft to re-examine their software development tools, in order that they could lead the market in promoting their operating system and applications. C# is thus aimed more at scientific/engineering applications, and allows for more flexibility, such as using memory pointers. There is also a great amount of code developed for many different applications, such as DSP, interfacing, and so on. Engineers and scientists tend to favour the minimal syntax of C# to the rather *basic* syntax of VB. The traditional weaknesses of VB, such as the lack of flexibility and in the usage of variables before they are declared, have gone, and only the core weakness of BASIC languages, itself, is the only weakness left.

## 1.6 Introduction to Object-orientation

---

We live in a world full of objects; so object-oriented programming is a natural technique in developing programs. For example, we have an object called a cup, and each cup has a number of **properties**, such as its colour, its shape, its size, and so on. It is efficient for us to identify it as a cup, as we know that cups should be able to hold liquid, and we will place our cup beside all the other cups that we have. If we were a cup designer then we could list all the possible properties of a cup, and for each design, we could set the properties of the cup. Of course, some of the properties might not actually be used, but for a general-purpose design, we would specify every property that a cup might have. For example, in a simple case we could have the following properties:

<b>Properties</b>	<b>Cup 1</b>	<b>Cup 2</b>	<b>Cup3</b>
Shape (Standard/Square/Mug)	Standard	Square	Mug
Colour (Red/Blue/Green)	Blue	Red	Green
Size (Small/Medium/Large)	Small	Large	Small
Transparency (0 to 100%)	100%	50%	25%
Handle type (Small/Large)	Small	Small	Large

Thus, we have three choices of shape (square, standard or mug), three choices of colour (red, blue or green), three choices in size (small, medium or large) and two choices of handle type (small or large). In addition, we can also choose a level of transparency of the cup from 0 to 100% (in integer steps). In object-oriented program, the collection of properties is known as a **class**. Thus, we could have a class for our cup which encapsulates all the design parameters for our cup. The instance of our class, such as Cup 1, Cup 2 and Cup 3, are known as **objects**. We can create many objects from our class. Along with this, there are certain things that we want to do with the cup, such as picking it up, painting it, or even dropping it. In object-orientation, these are known as methods, and are the functions that can be allowed to operate on our objects.

Program 1.4 shows an object-oriented example of the cup, where a class named `Cup` is created, of which an instance is named `cup`. A full description on this program will be discussed in a later module. It uses variables, such as `Shape`, `Colour`, and `Size` to define the property of an object.



**Program 1.4:**

```

using System;

namespace ConsoleApplication2
{
    public class Cup
    {
        public string Shape;
        public string Colour;
        public string Size;
        public int Transparency;
        public string Handle;

        public void DisplayCup()
        {
            System.Console.WriteLine("Cup is {0}, {1}", Colour, Handle);
        }
    }
    class Class1
    {
        static void Main(string[] args)
        {
            Cup cup = new Cup();
            cup.Colour = "Red";           cup.Handle = "Small";
            cup.DisplayCup();
            cup.Colour = "Green";       cup.Handle = "Small";
            cup.DisplayCup();
            System.Console.ReadLine();
        }
    }
}

```

**Sample Run**

```

Cup is Red, Small
Cup is Green, Small

```

In the following example, we create a class named `Circuit`, of which we create a new instance of it named `cir`. The class then has two methods, named `Serial` and `Parallel`.

**Program 1.5:**

```

using System;

namespace ConsoleApplication2
{
    public class Circuit
    {
        public string name;

        public double Parallel(double r1, double r2)
        {
            return((r1*r2)/(r1+r2));
        }
        public double Series(double r1, double r2)
        {
            return(r1+r2);
        }
    }
}

class Class1

```

```

{
    static void Main(string[] args)
    {
        double v1=100,v2=100;
        double res;

        Circuit cir = new Circuit();

        cir.name="Circuit 1";
        res=cir.Parallel(v1,v2);
        System.Console.WriteLine("[{0}] Parallel resistance is {1} ohms",
            cir.name,res);

        cir.name="Circuit 2";
        res=cir.Series(v1,v2);
        System.Console.WriteLine("[{0}] Series resistance is {1} ohms ",
            cir.name,res);

        System.Console.ReadLine();
    }
}

```

#### Sample Run

```

[Circuit 1] Parallel resistance is 50 ohms
[Circuit 2] Series resistance is 200 ohms

```

In this case, we have used a single object (`cir`). Of course, we could have created two objects, with:

```

Circuit cir1 = new Circuit();
Circuit cir2 = new Circuit();

cir1.name="Circuit 1"; res1=cir1.Parallel(v1,v2);
cir2.name="Circuit 2"; res2=cir2.Series(v1,v2);

```

Finally, for this section, Program 1.6 shows an example of a complex number class, where a complex number object is created ( $x$ ), which is made up of two components ( $x.real$  and  $x.imag$ ). The class defines two methods: `mag()` and `angle()`, which calculate the magnitude and the angle of the complex number, using:

$$z = x + jy$$

$$|z| = \sqrt{x^2 + y^2}$$

$$\langle z \rangle = \tan^{-1}\left(\frac{y}{x}\right)$$

#### Program 1.6:

```

using System;
namespace ConsoleApplication2
{
    public class Complex
    {
        public double real;

```

```

public double imag;

public double mag()
{
    return (Math.Sqrt(real*real+imag*imag));
}
public double angle()
{
    return (Math.Atan(imag/real)*180/Math.PI);
}
}
class Class1
{
    static void Main(string[] args)
    {
        string str;
        double mag,angle;
        Complex r = new Complex();

        System.Console.Write("Enter real value >>");
        str=System.Console.ReadLine();
        r.real = Convert.ToInt32(str);

        System.Console.Write("Enter imag value >>");
        str=System.Console.ReadLine();
        r.imag = Convert.ToInt32(str);

        mag=r.mag();
        angle=r.angle();

        System.Console.WriteLine("Mag is {0} and angle is {1}",mag,angle);
        System.Console.ReadLine();
    }
}
}

```


Sample Run	
Enter real value >>	3
Enter imag value >>	4
Mag is	5 and angle is 53.130102354156

### 1.6.1 Using the environment

The C# code is displayed in the Editor windows, as illustrated in Figure 1.15. The classes within the file are displayed within the Types pull down menu item. It can be seen, that, in this case, that the classes are named:

- ConsoleApplication2.ArrayExample02().
- ConsoleApplication2.test().

These are defined with the namespace of ConsoleApplication2 (namespaces will be covered in a following section). Once within the class the members of the class are shown in the right-hand pull down menu. The main graphics used to display the members are:

 - Method within a class. If it has a padlock, it is a private method.

- 🔒 - Variable within a class. If it has a padlock, it is a private variable.
- 🏠 - Property of a class. Figure 1.16 shows an example of this.

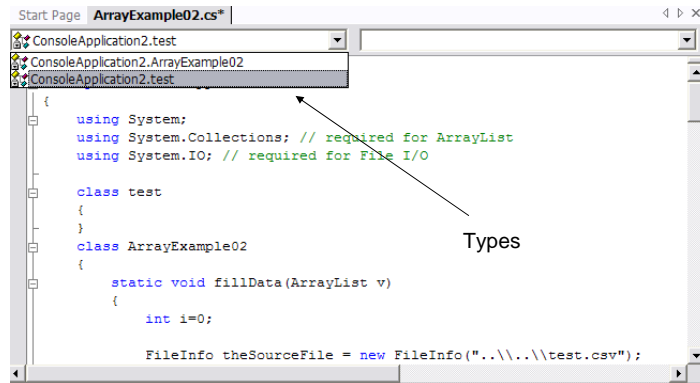


Figure 1.15: Types

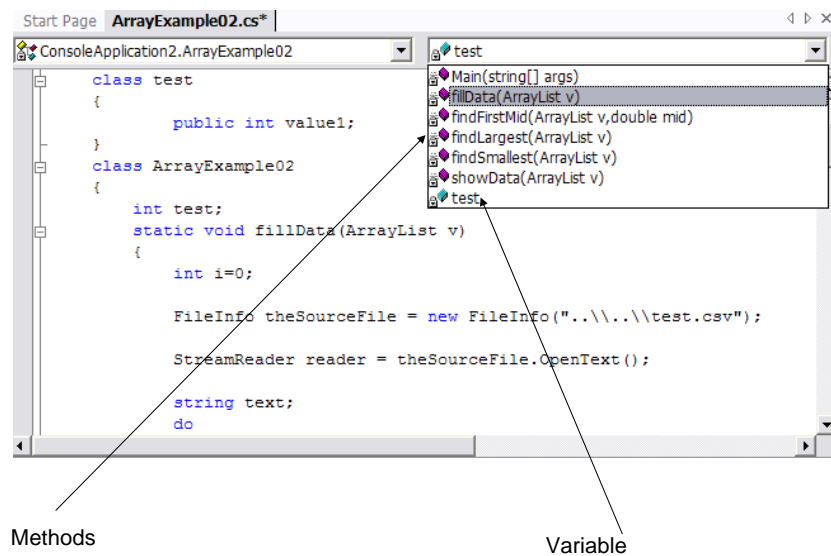


Figure 1.16: Class members

## 1.6.2 Viewing objects

Along with viewing the classes, it is possible to view the complete hierarchy of the objects in the solution. This includes the hierarchy above the classes that have been developed. Figure 1.17 shows an example of the object browser, and Figure 1.18 shows the browsing of the objects within the core library (mscorlib). In this case it shows the objects within Systems.Collections.

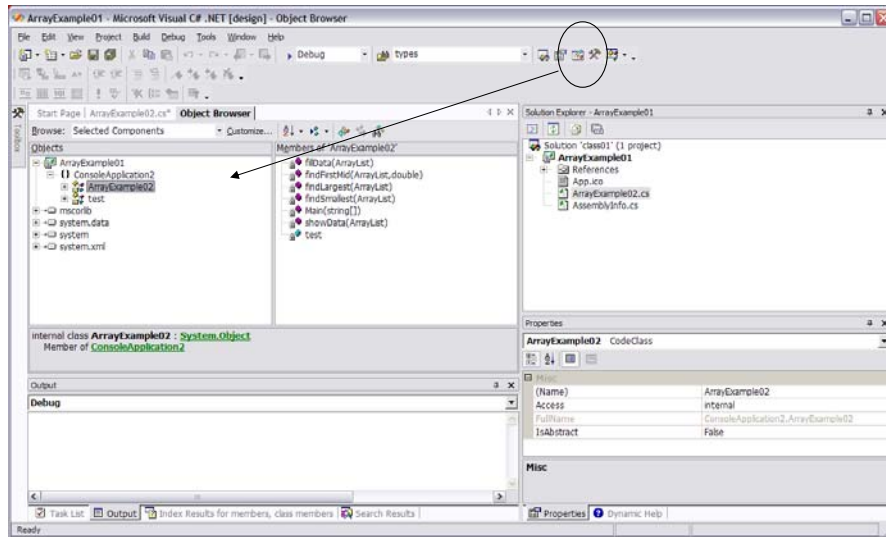


Figure 1.17: Object browsing

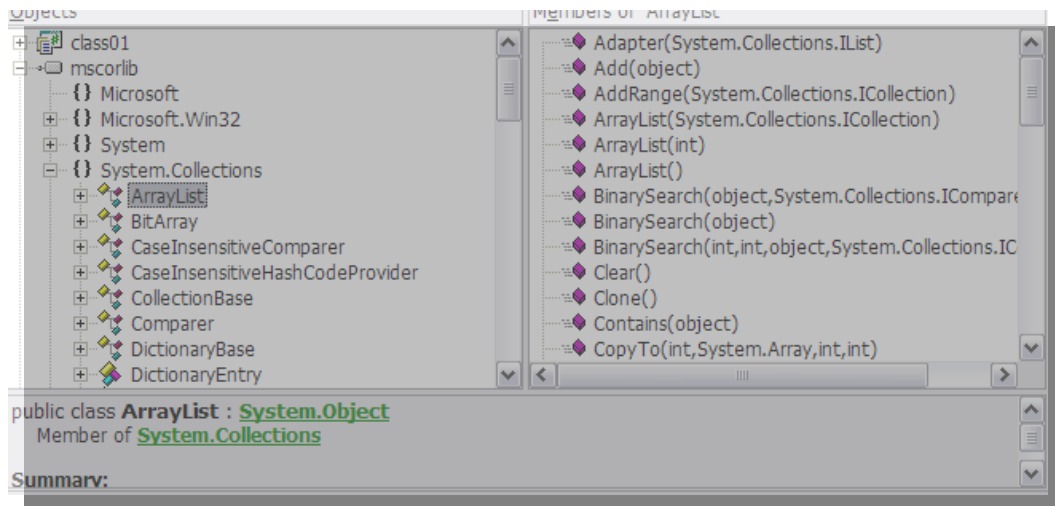


Figure 1.18: Object browsing for other classes

## 1.7 Basic elements of a C# program

C# follows its parents in having a minimal number of keywords. These are:

abstract	event	new	struct
as	explicit	null	switch
base	extern	object	this
bool	false	operator	throw
break	finally	out	true

byte	fixed	override	try
case	float	params	typeof
catch	for	private	uint
char	foreach	protected	ulong
checked	goto	public	unchecked
class	if	readonly	unsafe
const	implicit	ref	ushort
continue	in	return	using
decimal	int	sbyte	virtual
default	interface	sealed	volatile
delegate	internal	short	void
do	is	sizeof	while
double	lock	stackalloc	
else	long	static	
enum	namespace	string	

C#, as the other C languages is case sensitive, thus the keywords must use the lowercase format. The main elements of a C# program are given next.

```

using System;
namespace ConsoleApplication2
{
    public class Complex
    {
        public double real;
        public double imag;

        public int val { set {} get {} };

        public double mag()
        {
            return (Math.Sqrt(real*real+imag*imag));
        }
        public double angle()
        {
            return (Math.Atan(imag/real)*180/Math.PI);
        }
    }
}
class Class1
{
    static void Main(string[] args)
    {
        Complex r = new Complex();
        string str;
        double mag, angle;

        System.Console.WriteLine("Enter real value >> ");

        str=System.Console.ReadLine();
        r.real = Convert.ToInt32(str);

        System.Console.WriteLine("Enter imag value >> ");
    }
}

```

**using.** Imports types defined in other namespaces.

**namespace.** Defines a unique name for the objects. In this case the objects would have the name of:  
 ConsoleApplications2.Complex()  
 ConsoleApplicaitions2.Class1()

**Main().** This is the entry point into the program, and defines the start and end of the program. It must be declared inside a class, and must be static.

```

        str=System.Console.ReadLine();
        r.imag = Convert.ToInt32(str);

        mag=r.mag();
        angle=r.angle();

        System.Console.WriteLine("Mag is {0} and angle is {1}",mag,angle);
        System.Console.ReadLine();
    }
}
}

```

### 1.7.1 Namespace

For the namespace, the full name of the Complex() class, its full name would be:

```
ConsoleApplication2.Complex()
```

Thus we could have used:

```
ConsoleApplication2.Complex r = new ConsoleApplication2.Complex()
```

but as we are in the same namespace it is possible to use the relative form.

### 1.7.2 using

The `using` keyword is used imports types from other namespaces. Figure 1.19 shows types that are imported for `System.Console.WriteLine()`. It is important to import the required namespace for the classes that are required. Examples include:

<b>System:</b>	Array, Boolean, Byte, Char, Convert, DateTime, Double, Enum, Int16, Int32, Int64, Math, Random, String, Void
<b>System.Collections:</b>	ArrayList, BitArray, Hashtable, Queue, Stack.
<b>System.IO:</b>	BinaryReader, BinaryWriter, File, Stream, StreamWriter, StreamReader

```
System.Console.WriteLine("Enter real value >> ");
```

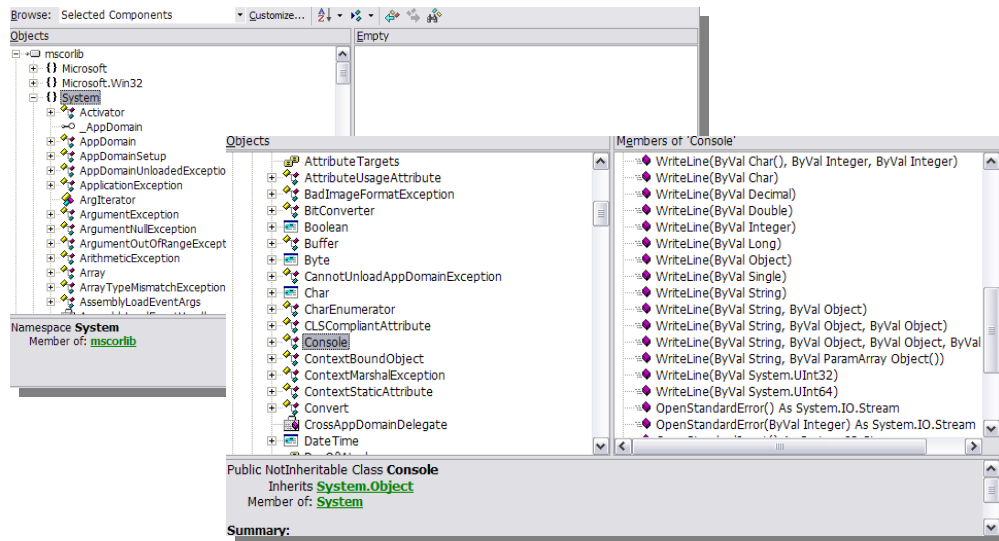


Figure 1.19: System namespace