

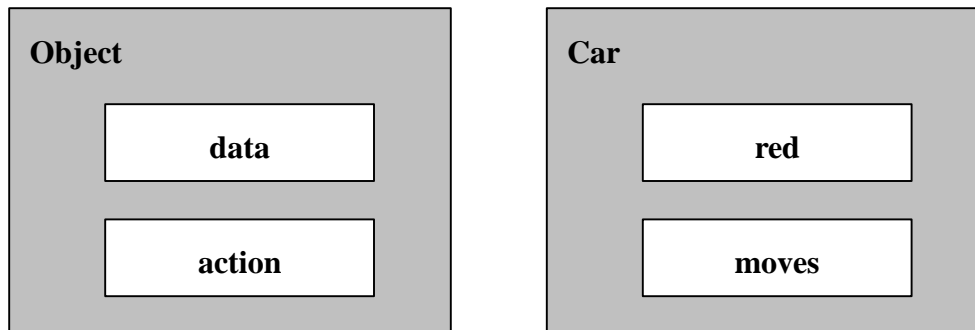
## C# PROGRAMMERS GUIDE LESSON 1

File:	CsGuideL1.doc
Date Started:	May 8,2001
Last Update:	Aug 10, 2006
Version	2003-2005

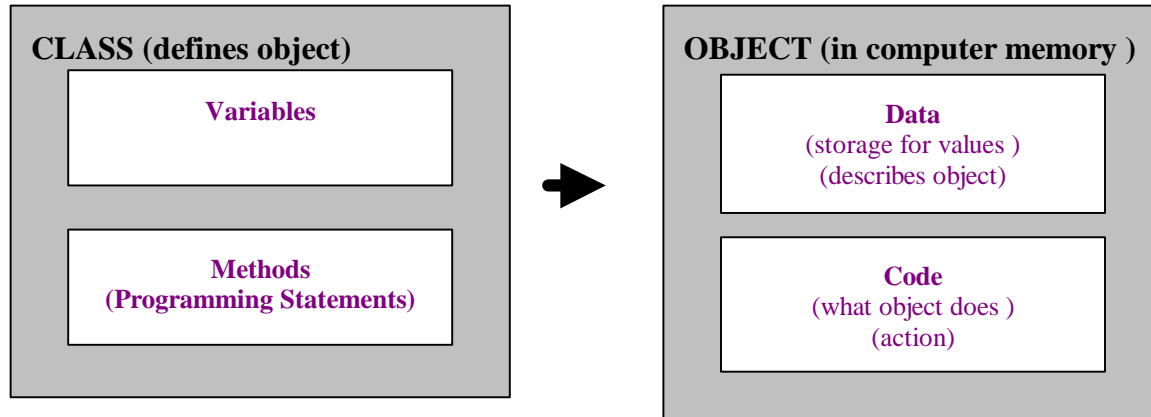
# C#

### INTRODUCTION

This manual introduces the C# programming language techniques and concepts. C# is a new language designed for general and Internet programming. C# is derived from C and C++. Now you get one great language that has the best features of C++ and C#. C# uses the **Object Oriented Programming** (OOP) approach. This approach uses a programming language to describe an object. What is an Object ? An Object is an everyday thing. A car, house, table or even a person. Every object has an appearance and does something (**action**). A car has a shape, a color and moves, People have names and address. The object oriented programming approach, wants to represent everyday objects in a programming language. How can it do this? A programming language can store the appearance of an object as **data** in the computer memory. The programming language also has **programming statements** to perform the action that an object does. Programming Statements are **instructions** telling the computer what the object does.



An object has two important components associated with it **data** and **code**. When a computer stores information about an object, the stored information is known as **data**. The data is the information that describes the objects and is stored in the computer memory. The location where the data is stored is represented by a **variable** having a name. Programming statements are used to describe what an object does and is translated into **code**. Code is used to run on the computer to perform some action or operation for the object. The code is the translated instructions telling the computer what to do. Programming statements ARE grouped together into what is known as a **method**. A method is a collection of programming statements that are defined for the object. Methods get translated into code and runs on your computer. Before you can have an object you need a definition of the object. A **class** is used to define an object. The class lists all the methods and variables needed by the **object**. A **class** defines an object having **data** and **code properties**. Think that a class is the drawings or plans to build a house. When the house is made then it is an object.



Objects are created in the computer memory from the class definitions. The class definitions tell what data and methods your object needs. Memory in the computer is allocated for the variables defined in the class definition. It is these created objects that use the computer memory for **data** when you run your C# program. The Object Oriented Programming approach will enable you to organize your programs so that your programming tasks will be much easier to accomplish. What is the difference between a class and an object? A class is the definition. An object is allocated memory for the variables defined in the class. An object has methods to do operations on the data defined in the class.

### Instruction approach

This manual teaches by using the **analogy** approach. This approach makes it easier for you to understand new concepts by comparing the operation to common known principles. We also use the **seeing** approach, the words we use, are carefully chosen so that you get a picture of what is happening. **Visualization** is a very powerful concept to understanding programming. Once you get the picture of what is happening, then the programming concepts are much easier to understand and accomplish. It is very important to visualize and see things as a picture rather than trying to understand words by reading and memorizing textbooks. Pictures are made up of words. This document does not cover all details of C# programming but acts as a guide so that you can get started right away with C# programming. The main purpose of this document, is to introduce, and teach you C# programming and Object Oriented Programming techniques. You should have a textbook as a companion to look up important terms for clarification or for more in depth study. Good textbooks are:

Title	Author(s)	Publisher
Programming C#	Jesse Liberty	O'Reilly
Computing with C#	Art Gittleman	Jones and Bartlett

## PURPOSE OF PROGRAMMING LANGUAGES

The purpose of a programming language is to direct a computer to do what you want it to do. In most cases of beginners the computer tells the programmer what to do. Good programmers are in control of the computer operation. A computer program lets the user enter data from the keyboard, perform some calculation to solve some problem and then displays the results on the computer screen. The user can store input and output data on a data file for future use.

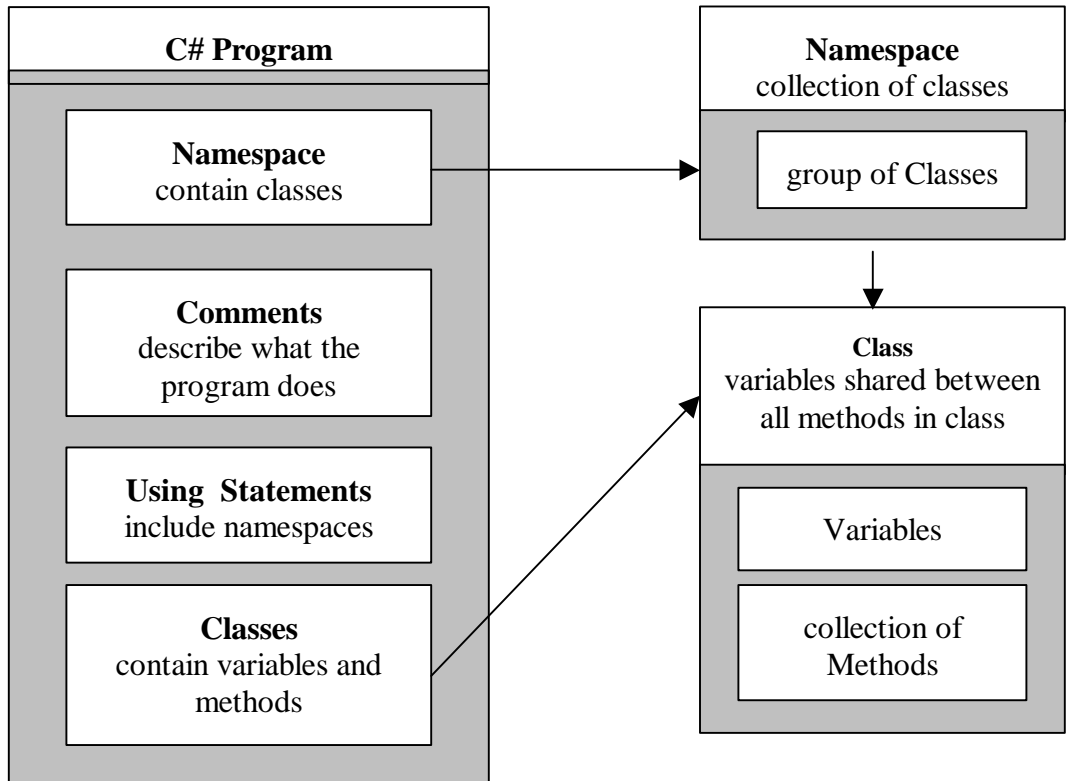
## LESSONS AND PROJECTS

There are 12 lessons and 1 project in the C# course. You should understand all the material and do all the exercises in each lesson proceeding before proceeding to the next Lesson. You must have working Project to pass this course. Grading is based on the exercises and project. (P) Pass, (G) Good and (E) Excellent. Excellent is awarded to students with outstanding programs that involve creativity and works of genius. Good is awarded to students that have exceptional working programs. Pass is awarded to students that have minimal working programs. Important concepts and C# keywords are introduced in **bold**. Keywords are the reserved words in a programming language that make up the Language definitions. Language definitions are in *purple italics* and programming statement examples are in *blue* and comments describing the programming statements are in *green*.

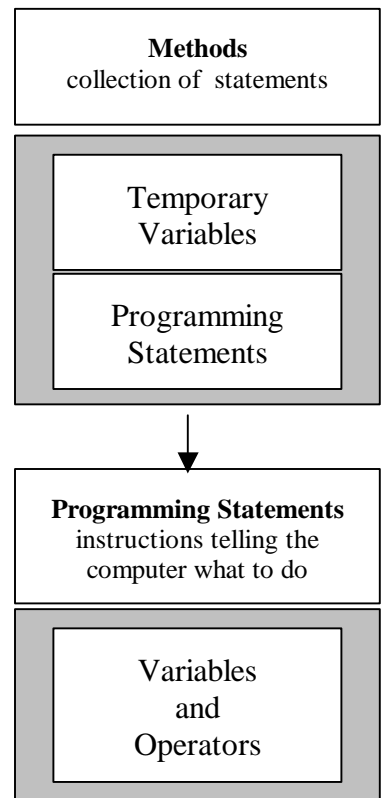
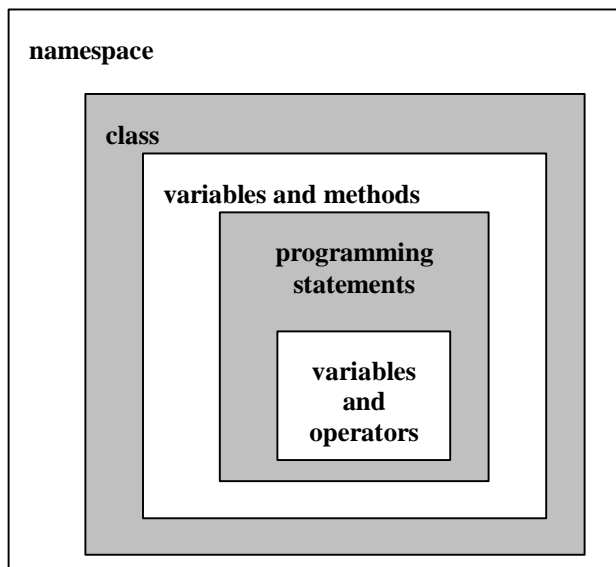
## LESSON 1 C# PROGRAM COMPONENTS AND VARIABLES

### C# PROGRAM FORMAT AND COMPONENTS

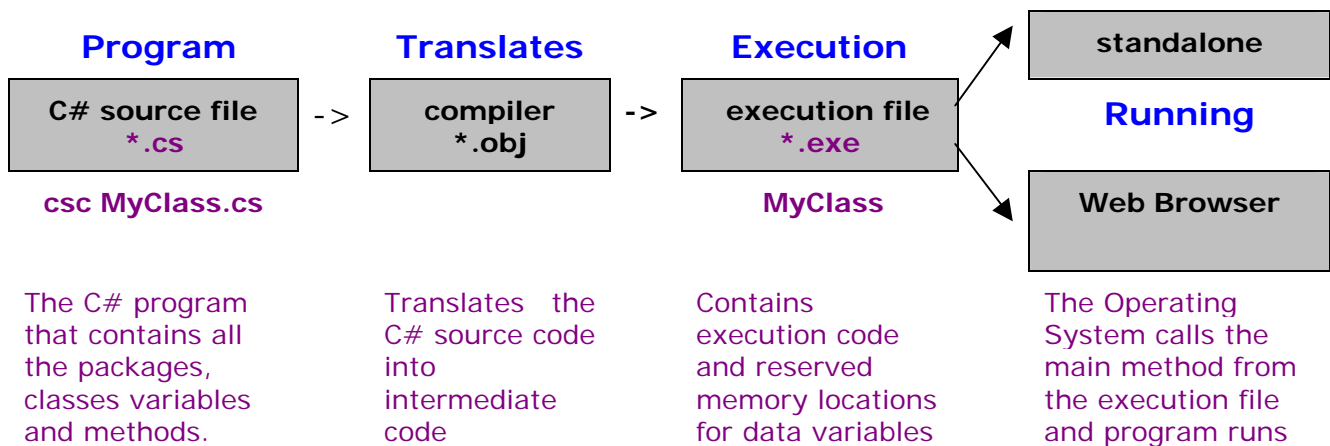
A C# program is made up programming components usually arranged in a predefined format. By following a proven format, your programs will be more organized and easier to read. A C# program contains **namespaces**, **using statements**, **classes**, **methods**, **programming statements**, **variables** and **operators**. **Namespaces** are used to group **classes** together that have something in common. **Using** statements are used to let you use classes not listed from your program located in an external class library. A class library contains classes grouped together in a **namespace**. The **namespace** identifies the group of classes. **A Class** declares and defines the **variables** and **methods** needed by an **object**. **A class** defines an **object**. **Variables** are used to **store** and **retrieve** data values. **Methods** are what the object uses to perform the tasks it needs to do. **Methods** are also made up of **variables** and programming **statements**. Programming **statements** are instructions used to direct the flow of program execution. **Statements** are made up of **variables** and **operators**. **Operators** perform operations on variables like addition and subtraction. Think that a C# program has many different layers each depending on each other: A C# program is an organized collection of components.



You can think that a C# program is a onion with many layers. The inner layer depends on the outer layer.

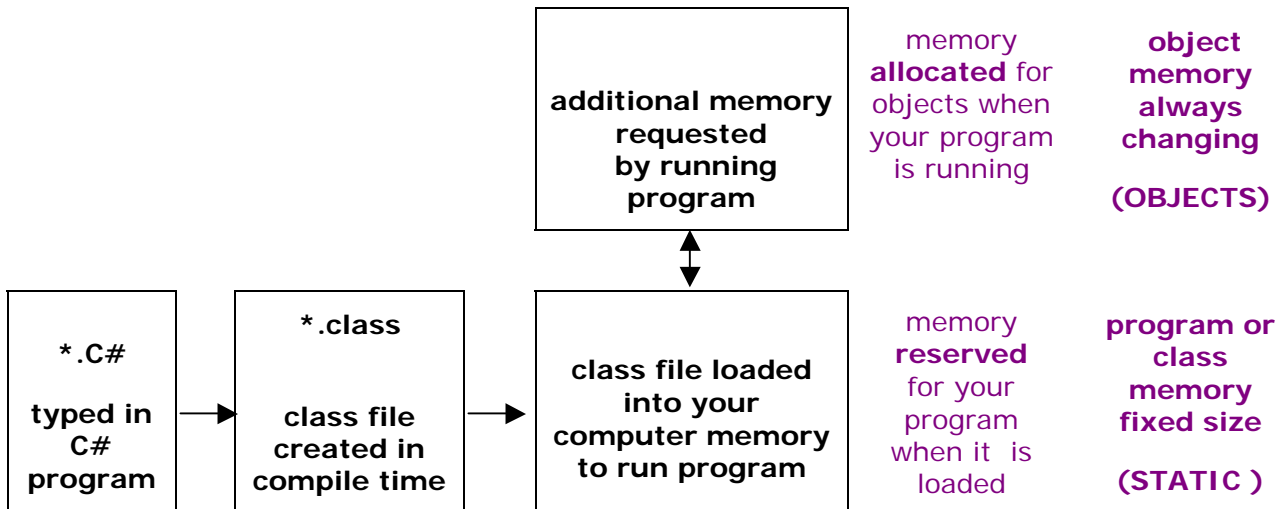


We will study each component individually. You do not need to understand every thing right now. The most important thing is that you are introduced to all the **terminology** and the **components** that make up a C# program. Analogy to a C# program is what cities and towns are made up of. Towns are made up of neighborhoods, neighborhoods are made up of streets, streets are made up of homes, homes are made up of people etc. A minimal C# program has only one class. A minimal class has one method called the **main method**. The **main method** is the first method to run in a C# program. When a C# program **runs** it means it is **executing** programming statements contained in a method. You run a C# program by executing a class file that contains a main method. I call this class the "**running class**". The class that starts your C# program. The C# program is stored in a file having the "**\*.cs**" extension. The file that has the class with the main program in it should have the same name as the class name. Each class is usually kept in a separate file having the same name as the class. The file name and class name is case sensitive meaning upper and lower case characters are considered different. Before a C# program can run it must be compiled to an execution file having the **\*.exe** extension. A **compiler** is used to convert your program into execution code. The C# compiler is available in Visual studio 7.0. It has both GUI Graphics User Interface and command line tools.



### Compile time and run time

**Compile time** is the time when the compiler is compiling your program into an execution code file. During compile time the compiler is checking your program for errors. If you have errors in your program you have to fix them and then re-compile. During compile time the compiler will convert your C# programming statements into execution code and **reserve memory** locations for your declared data variables and compiled programming statement code. When your program runs, the class execution file is loaded into your computer. The operating system gives you some memory to run your program. The size of the memory depends on the size of your execution file. We call this memory **program** or **class memory** and is said to be **reserved** memory. It is at a fixed size and never changes. Some methods are static meaning they can be used directly from the class execution code. When a method is static it can be accessed directly by using the class name. When a method is not static this means only an object created from the class definition can access the method. When your program is running it may need additional memory for objects called **object memory**. The additional memory comes from unused computer memory and is said to be **allocated** memory. Objects are created in computer memory during run time from the class definitions defined in your program. What is the difference between compile time and run time ? What is the difference between a class and an object ?



Why do we need additional memory when a program is running ? A good example is a telephone book program. When someone writes the program they do not know how many telephones entries the book is going to have. For every person needed in the telephone book there must be enough memory for the persons name, address and telephone number. When the program first runs it will ask the person how many entries the telephone book will have. As soon as the person enters the number the program must go to the computer memory and get enough memory for 100 entries. 100 entries may not be enough, you could have thousands of people wanting telephones. Alternatively you could just add additional memory when you add entries to your telephone book. You allocate memory in run time when you do not know how much memory you need. You reserve memory in compile time when you know how much memory you need. If you knew how many people you have in your telephone book then you could reserve all memory in compile time. You will soon learn how to reserve memory in compile time and allocate memory in run time.

## C# PROGRAM COMPONENTS

We will now discuss in depth all the components of a C# program. In C# you will see curly brackets like these { } all the time. The open curly { bracket means to begin something. The closed curly bracket } means to end something. They are used to group statements together in a method or methods in a class.

### Comments

All programs need comments to explain to the user how the program works, what the variables represent and to define the purpose of classes and methods etc. You may place comments anywhere in your C# program. The compiler ignores comments. They are just there for the people who read your C# program. A comment starts with a `/*` and ends with a `*/`.

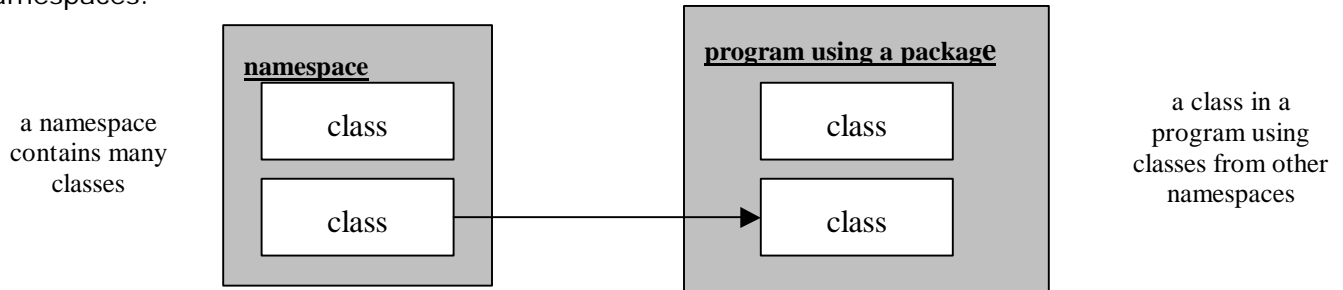
```
/* My first C# comment */
```

A comment may also start with a `///` . The end of the comment is assumed to be the end of the line.

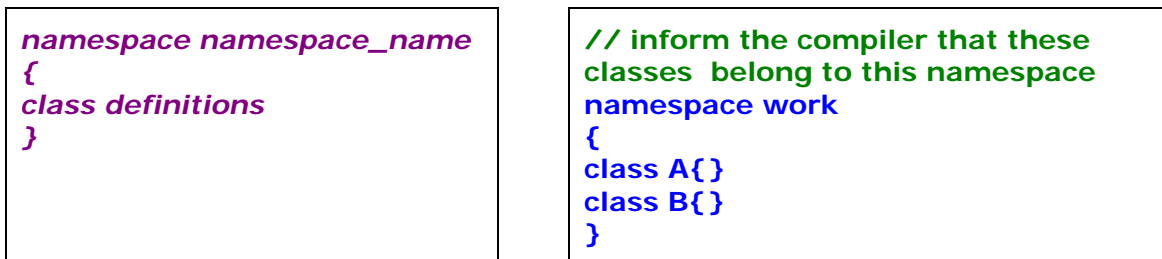
```
/// My first C# comment
```

## Namespaces

Having your own Namespaces are optional and used mostly in large programs involving many classes. Namespaces allow you to group a collection of classes together that have a common purpose. A namespace includes many class and a class may also use classes from other namespaces.



To make your class belong to a namespace you just put the name of the namespace at the beginning of your C# source code file. All classes in this file will now belong to your namespace. Other programs can use your classes identified by your namespace named **work**.



You must be careful in using namespaces of the same name. Some C# development tools will overwrite classes having the same name with the same named package without warning. You will lose all your original code. Always make sure all your namespace names are unique.

### using the net framework system classes

The net framework is a collection of classes enclosed in individual namespaces that allows you to run your programs on the Internet and on the computer. There is a namespace called **System** that has classes that let you write data to the screen and get data from the keyboard. The using statement

```
using namespace;
```

```
using System; // use classes from system namespace
```

is used to include C# system classes contained in the System namespace. The system class has many methods so that you can easily get your C# program up and running quickly. The purpose of the system classes is to relieve the programmer of all burdens. The System class has input/output classes that allows you to get data from a keyboard or file system, send information to the computer screen, store data on an output file, make a connection to the internet etc. You will get used to all the system packages in C# because you are always forced to use them.

The **compiler** needs to know that you want to use the system classes contained in the system namespace when it **compiles** your program. If you just want to use one class in this package then you can specify it by name.

```
using System.Data;
```

## using your own namespaces

If you have your own classes in a name space. To use your own namespace you must also use the using statement

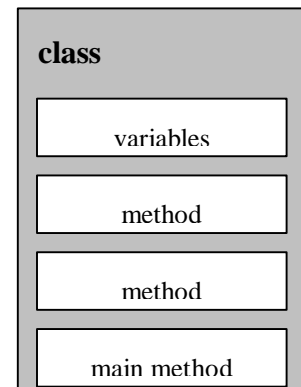
```
using user_namespace_name;
using work; // use all classes in work name space
```

If you just want to use a specific class of a namespace then you just specify which class you want:

```
using namespace_name . class_name;
using work.A; // just use the calculate class from work namespace
```

## Classes

A **class** defines all the **variables** and **methods** that a **object** will need. A class is defined by a **level of accessibility**, the **class** keyword, the **class name** and the required **variables** and **methods**. The open curly bracket { means to begin the class the closed curly bracket } means to end the class definition. Classes are defined in a program. A class contains a main method. The main method is the first method to execute when the class is executed. When we say a class executes we mean the main method is executed.



Here is the class definition (left) and an example class (right)

```
using statements
accessibility class
class_name
```

```
{
  variables
  methods
}
```

```
// example class
using System;
public class L1p1
```

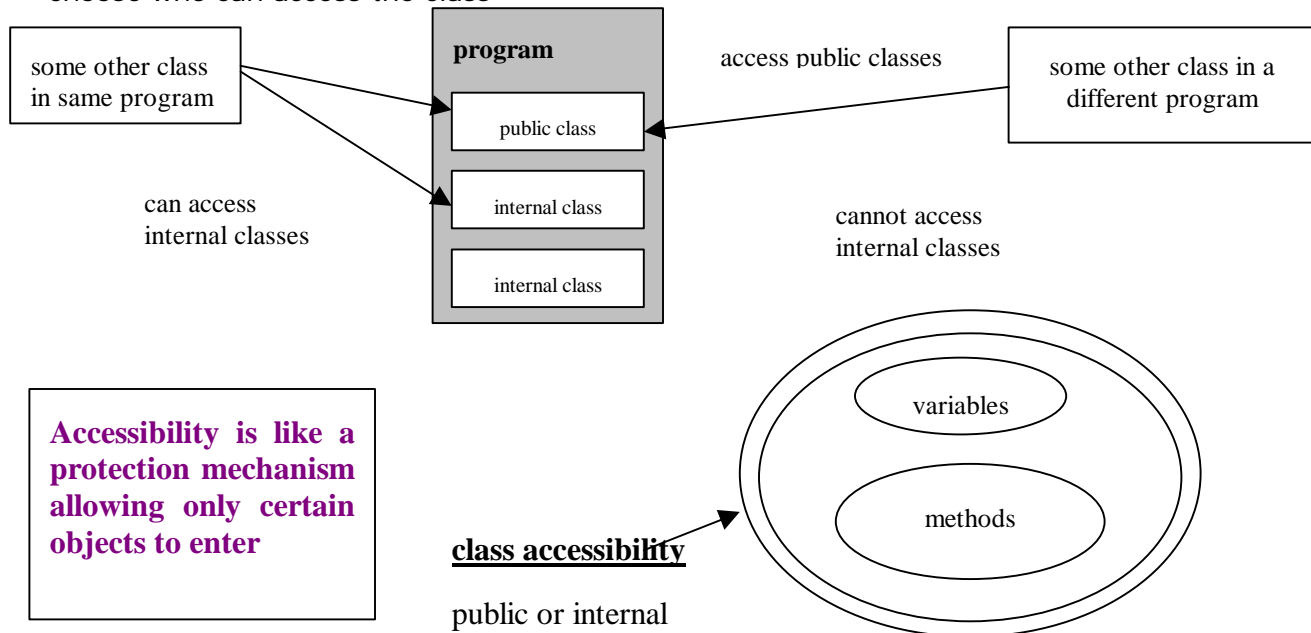
```
{ // begin class
  // main method
  public static void Main ()
  { // begin method
    Console.WriteLine("Hello");
  } // end method
} // end class
```

## visibility of classes

The **level of visibility** states who can use this class. Visibility for classes can be **public, or internal**:

class visibility	description
<b>public</b>	means everybody can access this class
<b>internal</b>	access limited to this program

If the accessibility is not specified the default accessibility is assumed to be internal. The visibility of our example class is public. Accessibility is like a protection mechanism where the programmer can choose who can access the class



When the **class** keyword is used it means the following block of code is to be a class definition. All classes need a name so that the programmer and compiler can identify it. The name of our example class is L1P1. The variables and methods belonging to a class are enclosed by curly brackets { }. The variables in a class are shared between all methods in the class. This is why classes were invented so that we could have a collection of methods all sharing the same variables. This is a highly desirable thing to do, it allows your programs to be highly organized and stops data corruption. Data corruption is avoided because we now know which methods access the data. All classes start with a CAPITAL letter where methods start with a lower case letter. You need to define a class before you can make an object of it. A class definition is like the recipe to bake a cake. You need a recipe before you can bake a cake. When a cake is baked it is ready to be eaten. The cake is the object. Obviously you cannot eat a recipe! When your program is running objects are constructed in the computer memory from your class definitions defined in your program.

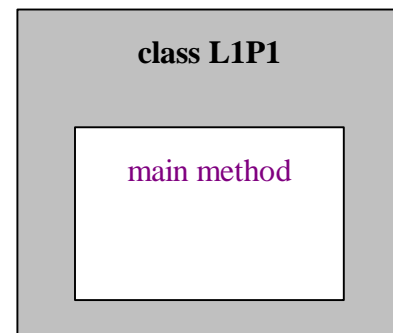
## class operation

A class contains **variables** and **methods**. Just like a cake needs **ingredients** and **instructions** to bake the cake. Variables are used to store and retrieve data. The variables in a class are shared between all methods in the class and retain their value. This is the main reason we have classes is that we want a bunch of methods to share the same variables! A class also defines all the methods needed for the operation of the class. Methods contain variables and **programming statements**. Variables in a method are used for temporary storage and retrieval of data, where the programming statements are the instructions telling the computer what to do. The variables in a methods do not retain their value and are said to be temporary. Variables in a method only contain values when the method is running. After a method is finished running the variables disappear. We will study methods in future lessons. In this lesson we will concentrate on variables.

### LESSON 1 PROGRAM 1

A simple C# program only needs a **class** and a **main method**. The main method is the first method to be executed when the C# program first runs. **Each class can have its own main method**. All class names in C# start with a CAPITAL letter, where method names start with a lower case letter. The following C# program is an example class called L1p1 and is **public** meaning everybody can use it. Our example class just has a main method and prints out the message "Hello" to the screen.

```
using System;
// example class
public class L1p1
{
    // main method
    public static void Main ()
    {
        Console.WriteLine("Hello");
    }
}
```



Program output:

```
hello
```

The L1p1 class is the "running class" because it is the class we call to run the program. It makes sense that the running class is public. Why ? Our main method just puts the word "hello" on the screen using the following programming statement:

```
Console.WriteLine("Hello"); // print "hello" on computer screen
```

The Console class is defined in the namespace System belonging to the Net framework, You need to use the **using** statement to access the Console class in the System namespace.

## LESSON 1 EXERCISE 1

Type the above program into your C# compiler and run it. If you are using the IDE make a project called Lesson1 and a class called L1p1. For those people using the command line, type the program into an editor, the file will be called L1p1.cs. From the command line of your computer type:

```
csc L1p1.cs
```

To execute type:

```
L1p1
```

Your program should now run. The file name should be identical to the class name. Change the above program so that it print's out "Hello" and your **name** on the computer screen. Call your class L1p1 and your C# file L1ex1.cs. You will now need the following commands to compile and run your program.

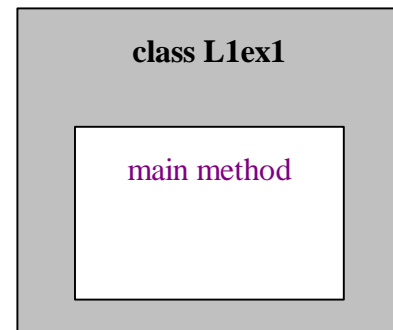
```
csc L1ex1.cs
```

```
L1ex1
```

### main method

When you run your program, the main method is the first method to be called from the running class. For Every class that you want to run you must have a main method.

```
using System;  
// example class  
public class L1ex1  
  
    {  
        // main method  
        public static void Main ()  
  
            {  
                Console.WriteLine("Hello");  
            }  
    }  
}
```



We will now discuss and analyze the Main method in detail. Every method has a **header** that declares the method.

<i>method accessibility</i>	<i>modifier</i>	<i>return data_type</i>	<i>method name</i>	<i>parameter(s) (parameter type and name)</i>
public	static	void	Main	( )

The header declares:

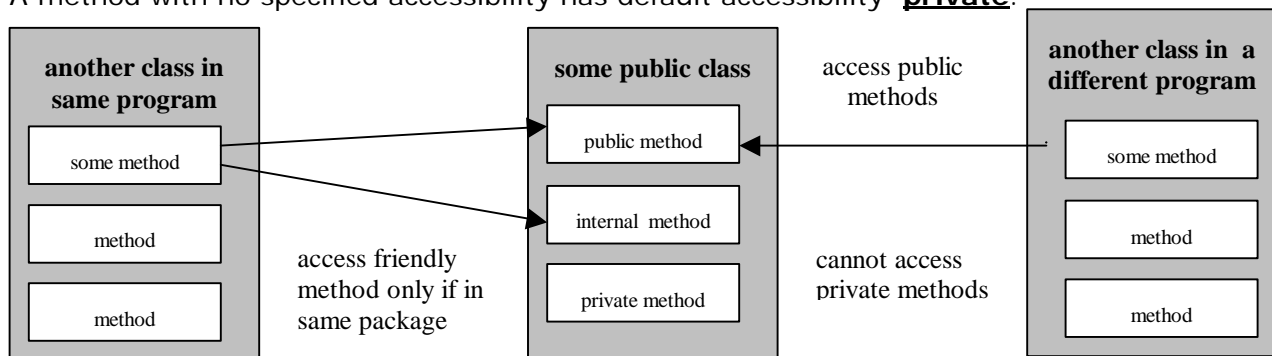
- (1) who can access the method (method accessibility)
- (2) modifiers features of this method
- (3) values the method returns
- (4) the name of the method
- (5) what values a method receives.

### (1) method accessibility

Methods have accessibility **public**, **internal** and **private**.

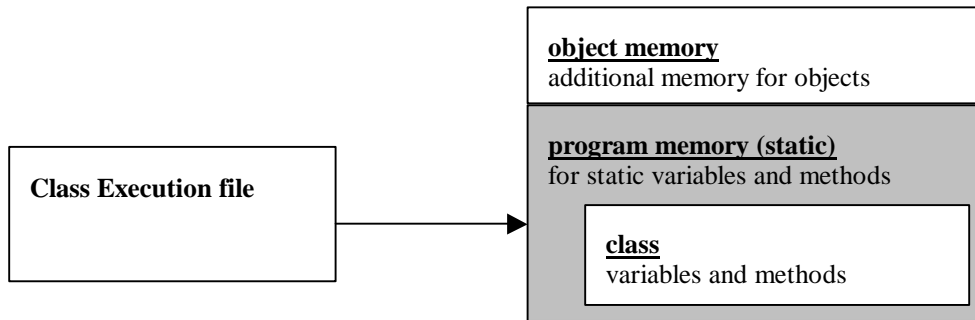
method visibility	description
<b>public</b>	means everybody can access this class
<b>internal</b>	access limited to the classes defined in this program
<b>private</b>	access limited to the class that defined the method

A method with no specified accessibility has default accessibility **private**.



## (2) modifiers

**static** means the code of the main method is readily available to use. You do not need to make an object of the class to use static variables and methods. The variables and methods are placed in program memory when the class is loaded and is ready to be used. Program memory is for classes the rest of the computer memory can be used for objects. Objects are created in computer memory when your program is running.



## (3) values the method returns

The purpose of a method is to receive a value, do a calculation and return a result. A method may or not return a value. The return data type **void** is used to indicate the main method does not return a value. The **return data type** is the type of data the method will return. **void** means return no data type. We will study data types very shortly.

## (4) method receives values

All method names end with round brackets () to distinguish the method name from an variable name. Values are sent to a method through a **parameters**. Parameters are declared inside the round brackets of the method, so the method can receive values. The values sent to the main method are known as **arguments**. Our main method has no parameters so it cannot receive any values. We can write a main method that has parameters to receive values.

```

using System;
// example class
public class L1p2

{
    // main method receive values from the command line
    public static void Main (string[] args)

    {
        Console.WriteLine("Hello " + args[0]);
    }
}
  
```

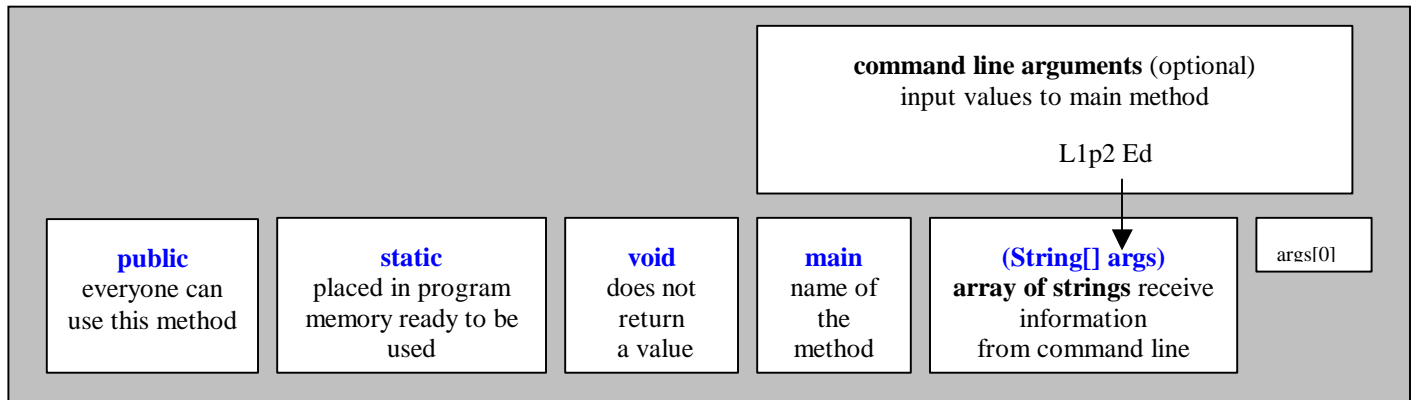
Now we have a parameter: **string[] args**. The **parameter type** is **string[]** and the parameter **name** is **args**. The parameter type tells you what kind of data you want to receive, The parameter name **arg** represents the received data. **Arg** represents an array of String objects that will let you send data values to the main method. Where do these values come from ? You send the data to the

main method through the parameter list by including extra information on the command line when you run your C# program. An example is when you include a file name when you want to run your C# program on the computer.

**L1p2 Ed**

Hello Ed

Ed will be placed into an array of Strings referred to by the parameter args.



Do not be too worried about arguments and parameters now, you just need to know that you can send information to the main method from the command line when you run your C# program. The String object type will be explained in this lesson. Arrays and methods will be explained in detail in future lessons. Without the main method the program can not run. Why must the main method be made public ? Why is the main method static ? Try the above program your computer on the command line call it L1p2.cs.

## Constants

Numbers like (0-9) and letters like ('a'- 'Z') are known as constants. Constants are **hard coded** values also called **literals** and are assigned to variables that store data. There are many different types of constants.

<b>character:</b>	'A'	<b>numeric:</b>	9	<b>decimal:</b>	10.5
<b>real:</b>	1.05e1	<b>string:</b>	"hello	<b>boolean:</b>	true or false

Numbers represent numeric value whereas letters are assigned numeric values for identification from a chart known as the ASCII table. The letter 'A' is assigned the numeric value 65. Letters constants have **single quotes** 'a' around them and are known as characters. What is the difference between '9' and 9 ? Numbers may be grouped together to make larger numbers like 1234. Decimal numbers may be represented as 10.5 or as 1.05e1 known as a real literal. **e** means exponent 1.05

\* 10<sup>2</sup>. Characters maybe grouped together to make messages and enclosed by **double quotes**: "hello". Groups of letters are known as strings. What is the difference between "1234" and 1234 ? What is the difference between a character and a string ? Boolean constants represent true and false values.

### backslash characters

The backslash codes allow you to specify new line or tab in your character strings like: "hello\n". Characters are now represented **by uni-code**. With uni-code you can represent many characters of different languages. Unicode numbers are represented by the hexadecimal numbering system that you will study shortly. All hexadecimal numbers start with 0x and have numbers in the range 0 to 9 and A to F. The following is a hexadecimal number: 0x005c There are special characters that control how data is presented on the screen or printer and are represented as **back slash** codes.

code	description	unicode encoding	code	description	unicode encoding
\b	backspace	0x0008	\0	end of string terminator	0x0000
\n	new line	0x000A	\\	backslash	0x005C
\r	carriage return	0x000D	\v	vertical tab	0x000B
\t	horizontal tab	0x0009	\a	bell	0x0007
\"	double quotation mark	0x00022	\f	form feed	0x000C
\'	single quotation mark	0x0027			

### number representation

All memory is made up of **bits** having values 0 and 1. Eight bits are grouped together into a byte.

1 byte = 8 bits

0	1	1	0	1	0	1	0
---	---	---	---	---	---	---	---

Data types use 1 to many bytes. Data types representing small numbers use few bytes of memory where data types representing large numbers use many bytes of memory. Numbers in C# are **signed** meaning they can represent negative and positive numbers or unsigned meaning all numbers are positive only. For signed numbers the right most significant bit (MSB) indicates if the number is positive or negative.

For a **positive signed** number the MSB is 0 for a negative signed number the MSB is 1.

positive signed number

0	1	1	0	1	0	1	0
---	---	---	---	---	---	---	---

For a **negative signed** number the MSB is 1 for a negative signed number the MSB is 1.

negative signed number

1	1	1	0	1	0	1	0
---	---	---	---	---	---	---	---

### range of a number

The smallest negative value to the largest positive value a number can represent is known as the **range** of a number. We need to know the range of a number before we can use it or we will get inaccurate results.

### range of a signed number

We use the following formula to calculate the range of the signed number from the number of bits N the number has.

$$- 2^{(N-1)} \text{ to } (2^{(N-1)} - 1)$$

**N is the number of bits in the number.**

The range of a number is calculated for an 8 bit signed number where:

**N=8**

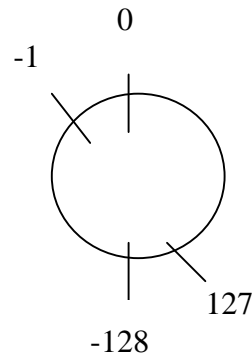
$$- 2^{(8-1)} \text{ to } (2^{(8-1)} - 1) = - 2^7 \text{ to } 2^7 - 1 = -128 \text{ to } 127$$

This means a signed number that has 8 bits can represent values between -128 and 127. Why is the negative number magnitude greater than the positive number magnitude ?

**The positive numbers start at 0 and end at 127**

**The negative numbers start at -1 and end at -128**

**Each has 128 numbers**



### range of a unsigned number

We use the following formula to calculate the range of the unsigned number from the number of bits N the number has.]

$$0 \text{ to } (2^N - 1)$$

**N is the number of bits in the number.**

The range of a number is calculated for an 8 bit signed number where:

**N=8**

$$0 \text{ to } 2^8 - 1$$

$$0 \text{ to } 2^8 - 1$$

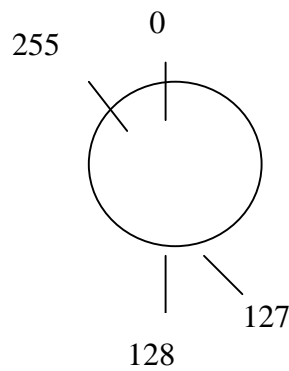
$$0 \text{ to } 255$$

This means a unsigned number that has 8 bits can represent values between 0 to 255

**The positive numbers start at 0 and end at 127**

**The negative numbers start at -1 and end at -128**

**Each has 128 numbers**



## VARIABLES AND DATA TYPES

**Variables** let you store and retrieve data in a computer memory represented by a variable name like **x**. The **variable name** lets you identify a particular memory location. The compiler reserves the memory address automatically for you when it compiles your program. A variable is like a bank account. You can put money in and take money out. The particular place (location) where your money is located is identified by the bank account number. Variables have different **data types**, just as a bank account would represent many different **currencies**. Each variable must be declared with a **data type**. The data type indicates what kind of data the variable is to represent. C# has many data types for you. The following chart lists the C# data types and an example of the data it is to represent.

C# data types				signed range
type	bytes	N bits	example	- $2^{\text{exp}(N-1)}$ to $2^{\text{exp}(N-1)}-1$
object	4	32	any value	represents location of object
string	4	32	"hello"	represents location of string
sbyte	1	8	45	-128 to 127
short	2	16	1234	-32,768 to 32,767
int	4	32	123467	-2147483648 to 2147483647
long	8	64	123456788L	-9223372036854775808 to 9223372036854775807
byte	1	8	45 45U	0 to 255
ushort	2	16	1234 1234U	0 to 65535
uint	4	32	1234567 1234567U	0 to 4294967295
ulong	8	64	123456788U 123456788UL	0 to 18446744073709551615
float	4	32	34.56f	$1.5E-45$ to $3.40282347E+38$
double	8	64	23.2333333D	$5.0E-324$ to $1.79769313486231570E+308$
boolean	1	8	true or false	0=false 1 = true
char	2	16	'A' (unicode)	16 bit character set for all world languages
decimal	28	96	1.655434M	28 digit decimal number $1.0E-28$ to $7.9E28$

The **object** type can represent any data type. Strings represent string data, decimal represent precise decimal numbers. All unsigned data types start with u except byte. Long constants are represented by L. Double constants are represented by D. Million is represented by M. We need different Data types because we want to **minimize** the amount of data memory space we use. The data type also indicates the type of data the memory allocated for the variable is to represent

## float and double number representation

Float and double data types are used to represent large numbers known as floating point numbers stored in a small memory space. Floating point numbers have a **fraction (mantissa)** part and an **exponent part** and represented in decimal point notation 24.56f or exponent notation 3.456E04f. You will notice that float constants have a 'f' at the end of them. In C# you need to distinguish a float constant from a double constant. It is the exponent that allows the floating point number to represent a large value in a small memory space. The following is a 32 bit floating point format known as a **float**, where the sign, exponent and fractional parts are stored separately.

1	8	23	32 bit floating point number ( <b>float</b> )
<b>sign bit</b>	<b>exponent</b>	<b>fraction (mantissa)</b>	$(-1)^{\text{sign}} * 2^{\text{exp}} * (\text{exponent}-127) * (\text{mantissa})$
-	12	.12365	$1.2365 * 10^{12}$ (example)

The following is a 64 bit floating point format known as a **double**, where the sign, exponent and fractional parts are stored separately. Double has greater precision over float because it stores more bits in their exponent and fraction.

1	16	47	64 bit floating point number ( <b>double</b> )
sign bit	exponent	fraction (mantissa)	$(-1)^{\text{sign}} * 2^{\text{exp}} * (\text{exponent}-127) * (\text{mantissa})$
-	56	.12365654	$1.2365654 * 10^{56}$ (example)

Variables store data

## Declaring variables

When you declare a variable you are telling the compiler to **reserve** a memory space location in the computer memory for your data when your program runs. The variable name is used to represent the location in the computer memory where the data value is stored. Declaring a variable is like opening a bank account. The name of the variable is like the bank account number. The bank account number tells you where your money is in the bank. The bank account number identifies the location where your money is in the bank. It is just like the variable in your program identifying the locations in the computer memory where your data value is stored. To declare a variable you specify the data or object type and the variable name. A data type is like the currency of the money stored in the bank account. You declare a variable by specifying the data type and a name for identification. Variable names should be short and meaningful. Every variable must be declared with a data type and a name. The data type tells your program what kind of data the computer memory is supposed to represent. The variable name tells your program the location in the computer memory where your data value is stored. Declared variables are initialized with default values. All integer 0, floats and decimal 0.0 and string and objects to null. null means no known location.

*data\_type* *variable\_name;*

**int y;** // declare variable y of data type int

**char ch;** // declare variable c of data type char.

**double d;** // declare variable d of data type double

default values

y	0
ch	0
d	0.0

You may declare more than one variable at a time if it is the **same data type**, by separating the variable names by commas.

*data\_type* *variable\_name\_list;*

**int i, j, k;** // declare a variable's i, j and k having data type integer

uninitialized variables

i	0
j	0
k	0

### declaring and initializing variables

You may assign values to variables when they are declared. This is called **initializing**. It is like opening a bank account with an initial deposit.

*data\_type* *variable\_name = value;*

**int x = 5;** // declare x and initialize to the value 5

**decimal d = 10.5M;** // declare d and initialize to the value 10.5  
// (M means a decimal type)

x	5
d	10.5

You can declare many variables and initialize them all at once if they are of the **same data type**, by separating the variable names and initializations by commas.

*data\_type* *variable\_name\_and\_initialization\_list;*

**int m=1, n=2, p=3;**

m	1
n	2
p	3

## String Objects

Character strings are groups of letters enclosed by double quotes:

**"message"**

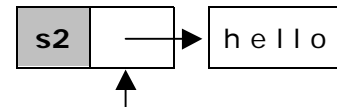
It is easy to use character strings in C# because C# has the built in String class. Character strings in C# are represented by the String class. The String class lets you do operations on strings. These operations may be assigning character strings or joining two character strings together. A character string is made up of characters. A character is a single letter in the string. A character is denoted by single quotes. like the letter 'a'. There is a big difference between 'a' and "a". 'a' is a character where as "a" is a character string with only 1 character. When you declare a String variable as follows:

```
string s; // declare variable s having a object type of String
```



The String variable s will store the location of a String object. s is known as a reference to a string variable. Right now variable s does not refer to any String object and is **null**. The best thing to do is to declare a String variable and assign a String object to it. The String object will store the character string. A string variable refers to a String object rather than contains the character string itself.

```
string s2 = "hello"; // declare and initialize s to "hello"
```



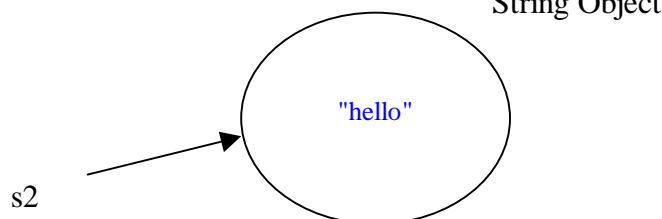
A String is an object that stores a character string.

s has location of string object

The string object contains the character string "hello". Refers means the string variable is not the String object itself but contains the **location** where the String object is located in the computer memory.

```
string s2 = "hello";
```

The variable s contains the location of the string object.



## Assignment statements

Assignment statements using the "=" assignment operator allow you to assign values to existing declared variables. The expression on the right hand side is assigned to the variable on the left hand side of the assignment "=" operator. The variable gets the new value and the old value is over written. Values are expressions. An expression may be a constant, numeric, String data value's or other variable with an assigned value. Once a variable is declared it does not need to be re-declared to assign a new value to it. Its like you do not need to re-open a bank account when you want to make deposits and withdraws. Before you can assign a value to a variable, the variable must be previously declared. When you assign a variable to another variable the value of the variable (not the variable name) on the right hand side is assigned to the variable on the left hand side.

Assignment statements have the form:

```
variable_name = expression;
```

```
x = 3;           // variable x gets the value of 3
```

```
y = x;           // variable y gets the value of x which happens to be 3
```

```
ch = 'A';        // variable ch gets the character constant 'A'
```

```
s = "today";     // the String object s gets the character string "hello"
```

<b>x</b>	<b>3</b>	<b>(x)</b>
<b>y</b>	<b>3</b>	
<b>ch</b>	<b>'A'</b>	
<b>s</b>		

→ **today**

## Printing out variable values to screen

The `Console.WriteLine()` statement is used to print values and messages to the computer screen. The Console class is defined in the System namespace. `WriteLine()` is a method of the Console class. The `WriteLine()` method causes your printed message to end with a new line **after** the message is printed not before. If you do not want a new line after your message use the `Write()` method instead.

```
Console.WriteLine("hello"); // print a message to the screen, with out a new line
```

```
Console.WriteLine("hello"); // print a message to the screen
```

```
hellohello
hello
```

```
Console.WriteLine("hello"); // print a message to the screen
```

To print out a variable value you just include the variable name.

```
Console.WriteLine(x); // print a variable value to the screen
```

```
3
```

## Printing messages and values

There are two ways to print a message and a variable value to the screen. Using the + operator of the String class or using place holders {0}. I like using the + operator rather than using place holders. Using the + operator of the string class is quite easy and fast.

```
Console.WriteLine("the value of x is: " + x );
```

```
the value of x is: 3
```

Using place holders seems confusion and a lot of work. To print a message and a variable value to the screen using placeholders you list the message in double quotes and use a placeholder {0} to substitute the formatted value of the variables to be printed. You close the string and after a comma you list the variable name. Messages are surrounded with double quotes and variables are not. The {0} is represent where you want the formatted value of the variable to appear in the message.

```
Console.WriteLine("the value of x is: {0}" , x );
```

```
the value of x is: 3
```

placeholder

variable

### LESSON 1 EXERCISE 2

Write a simple C# program that has only one class and a main method. In the main method declare **five** variables of different data types. and initialize these variables with values. Use the **Console.WriteLine** statement and place holders to print the values of the variables on the screen. Next assign values to your variables and print out the new values. Call your class L1ex2 and your C# file L1ex2.cs.

### Getting values from the keyboard.

You can use the **Console.ReadLine()** method to get string line from the keyboard.

```
s = Console.ReadLine(); // get string from keyboard
```

When you get values from the keyboard they are in a string format. You need to convert the string to the variable data type receiving the value. You need to use Methods from the **Convert** class listed as follows:

<b>ToBoolean</b>	Returns a Boolean value indicating whether the value of this instance of <b>String</b> is <b>true</b> or <b>false</b> .
<b>ToByte</b>	Returns an 8-bit unsigned integer
<b>ToChar</b>	Converts a non-empty string of length one to a Char.
<b>ToDateTime</b>	Returns the specified string converted to DateTime format.
<b>ToDecimal</b>	Converts a string to a Decimal.
<b>ToDouble</b>	Converts a string to a Double.
<b>ToInt16</b>	Converts a specified string to an Int16 in the specified base.
<b>ToInt32</b>	Converts a string to an Int32.
<b>ToInt64</b>	Converts a string to an Int64 value.
<b>ToSByte</b>	Returns an 8-bit signed integer

These methods are **static** meaning you can use them directly by specifying the class name as follows:

```
int x = Convert.ToInt32(Console.ReadLine()); // get number from keyboard
```

Here we use the **ToInt32** method from the **Convert** class to convert a string to a integer. (**Int32** means a 32 bit integer value)

Here is an example reading a number from a keyboard

```
// Example reading a number from keyboard
public class L1p3
{
    // main method receive values from the command line
    public static void main ()
    {
        // ask user to enter a number
        Console.WriteLine("enter a number: ");
        // get number from keyboard
        int x = Convert.ToInt32(Console.ReadLine());
        // write number to screen
        Console.WriteLine("The number is : {0}" , x);
    }
}
```

### LESSON 1 EXERCISE 3

Write a simple C# program that has only one class and a main method. In the main method declare **five** variables of different data types. Ask the user to enter values for each variable. Use the `Console.WriteLine()` statement to print the values of the variables on the screen. Call your class `L1ex3` and your C# file `L1ex3.cs`.

**IMPORTANT**

You should use all the material in all the lessons to do the questions and exercises. If you do not know how to do something or have to use additional books or references to do the questions or exercises, please let us know immediately. We want to have all the required information in our lessons. By letting us know we can add the required information to the lessons. The lessons are updated on a daily bases. We call our lessons the "living lessons". Please help us keep our lessons alive.

**E-Mail all typos, unclear test, and additional information required to:**

[courses@cstutoring.com](mailto:courses@cstutoring.com)

**E-Mail all attached files of your completed exercises to:**

[students@cstutoring.com](mailto:students@cstutoring.com)

**Order your next lesson from:**

<http://www.cscourses.com/cs.htm>

This lesson is copyright (C) 2006 by The Computer Science Tutoring Center "cstutoring"  
This document is not to be copied or reproduced in any form. For use of student only