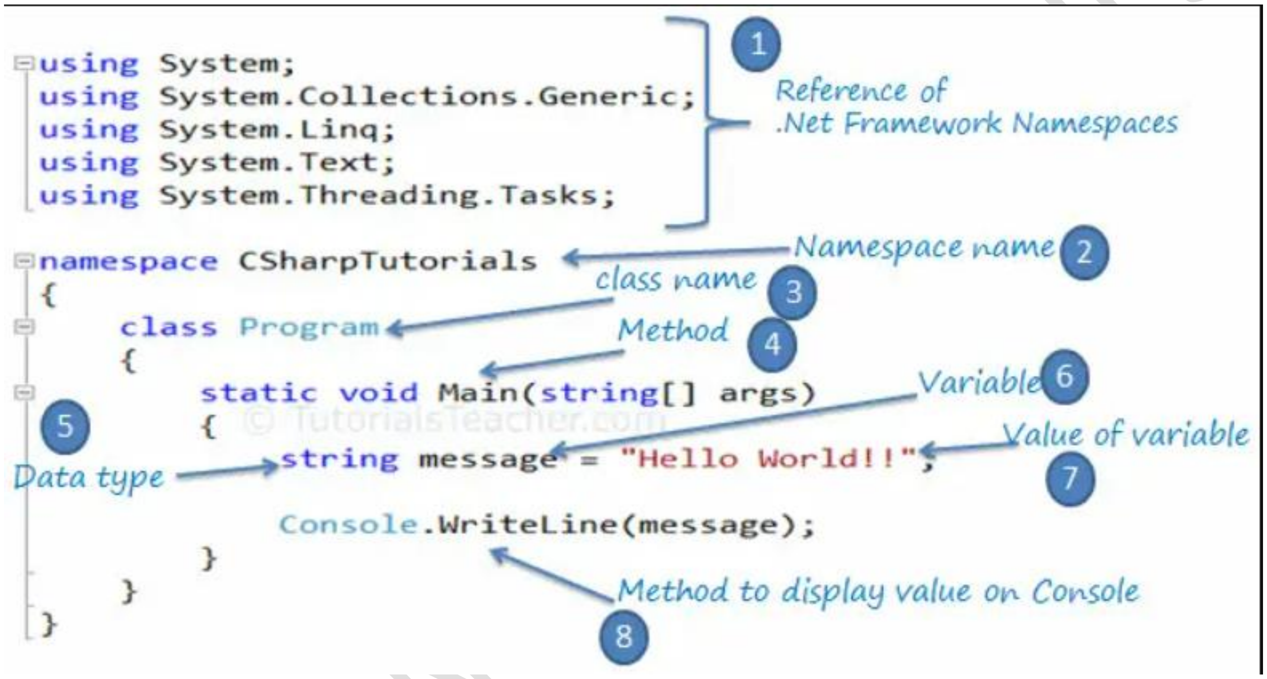
First Program

Create a simple console application in C# and understand the basic building blocks of a console application.

We will create a console application to work with C#.



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

namespace CSharpTutorials
{
    class Program
    {
        static void Main(string[] args)
        {
            string message = "Hello World!!";

            Console.WriteLine(message);
        }
    }
}
```

Let's understand the above C# structure.

- Every .NET application takes the reference of the necessary .NET framework namespaces that it is planning to use with the `using` keyword, e.g., `using System.Text`.
- Declare the namespace for the current class using the `namespace` keyword, e.g., `namespace CSharpTutorials.FirstProgram`
- We then declared a class using the `class` keyword: `class Program`
- The `Main()` is a method of `Program` class is the entry point of the console application.
- `String` is a data type.
- A `message` is a variable that holds the value of a specified data type.

- `"Hello World!!"` is the value of the message variable.
- The `Console.WriteLine()` is a static method, which is used to display a text on the console.

Note:

Every line or statement in C# must end with a semicolon (;).

Compile and Run C# Program

To compile it and run it by pressing Ctrl + F5 or clicking the Run button or by clicking the "Debug" menu and clicking "Start Without Debugging".

C# Namespace

Namespaces play an important role in managing related classes in C#.

The .NET Framework uses namespaces to organize its built-in classes.

For example, there are some built-in namespaces in .NET such as System, System.Linq, System.Web, etc. Each namespace contains related classes.

A namespace is a container for classes and namespaces. The namespace also gives unique names to its classes thereby you can have the same class name in different namespaces.

In C#, a namespace can be defined using the namespace keyword.

Example: Namespace

```
namespace School
{
    // define classes here

}
```

The following namespace contains the `Student` and `Course` classes.

Example: Namespace

```
namespace School
{
    class Student
    {

    }

    class Course
    {

    }
}
```

Classes under the same namespace can be referred to as `namespace.classname` syntax. For example, the `Student` class can be accessed as `School.Student`.

Example: Refer a Class with Namespace

```
namespace CSharpTutorials
{
    class Program
    {
        static void Main(string[] args)
        {
            School.Student std = new School.Student();

            School.Course cs = new School.Course();
        }
    }
}
```

To use classes under a namespace without the fully qualified name, import the namespace with the `using` keyword at the top of C# class file.

Example: Namespace

```
using System; //built-in namespace
using School;

namespace CSharpTutorials
{
    class Program
    {
        static void Main(string[] args)
        {
            Student std = new Student();
        }
    }
}
```

A namespace can contain other namespaces. Inner namespaces can be separated using (.).

Example: Namespace

```
namespace School.Education
{
    class Student
    {

    }
}
```

In the above example, the fully qualified class name is `School.Education.Student`.

Beginning with C# 10, you can declare a namespace for all types defined in that file without wrapping classes inside curly braces `{ .. }`, as shown below.

Example: C# 10 Namespace

```
namespace School.Education

class Student
{

}
```

C# Keywords

C# contains reserved words that have special meaning for the compiler. These reserved words are called "keywords". Keywords cannot be used as an identifier (name of a variable, class, interface, etc.).

Keywords in C# are distributed under the following categories:

Modifier Keywords

Modifier keywords are specific keywords that indicate who can modify types and type members. Modifiers allow or prevent certain parts of programs from being modified by other parts.

Modifier keywords
abstract
async
const
event
extern
new
override
partial
readonly
sealed
static
unsafe
virtual
volatile

Access Modifier Keywords:

Access modifiers are applied to the declaration of the class, method, properties, fields, and other members. They define the accessibility of the class and its members.

Access Modifiers	Usage
public	The Public modifier allows any part of the program in the same assembly or another assembly to access and its members.
private	The Private modifier restricts other parts of the program from accessing the type and its members. Only the same class or struct can access it.
internal	The Internal modifier allows other program code in the same assembly to access the type or its member default access modifiers if no modifier is specified.
protected	The Protected modifier allows codes in the same class or a class that derives from that class to access th its members.

Statement Keywords

Statement keywords are related to program flow.

Statement Keywords
if
else
switch
case
do
for
foreach
in
while
break

Statement Keywords
continue
default
goto
return
yield
throw
try
catch
finally
checked
unchecked
fixed
lock

Method Parameter Keywords

These keywords are applied to the parameters of a method.

Method Parameter Keywords
params
ref
out

Namespace Keywords

These keywords are applied with namespace and related operators.

Namespace Keywords
using
. operator
:: operator
extern alias

Operator Keywords

Operator keywords perform miscellaneous actions.

Operator Keywords
as
await
is
new
sizeof
typeof
stackalloc
checked
unchecked

Access Keywords

Access keywords are used to access the containing class or the base class of an object or class.

Access keywords
base
this

Literal Keywords

Literal keywords apply to the current instance or value of an object.

Literal Keywords
null
false
true
value
void

Type Keywords

Type keywords are used for data types.

Type keywords
bool
byte
char
class
decimal
double
enum
float
int
long
sbyte
short

Type keywords
string
struct
uint
ulong
ushort

Contextual Keywords

Contextual keywords are considered as keywords, only if used in specific contexts. They are not reserved and so can be used as names or identifiers.

Contextual Keywords
add
var
dynamic
global
set
value

Contextual keywords are not converted into blue color (default color for keywords in visual studio) when used as an identifier in Visual Studio. For example, var in the below figure is not in blue, whereas the color of this is the blue color. So var is a contextual keyword.

Query Keywords

Query keywords are contextual keywords used in LINQ queries.

Query Keywords
from

Query Keywords
where
select
group
into
orderby
join
let
in
on
equals
by
ascending
descending

Use Keyword as Identifier

As mentioned above, a keyword cannot be used as an identifier (name of the variable, class, interface, etc.). However, they can be used with the prefix '@'. For example, the class is a reserved keyword, so it cannot be used as an identifier, but @class can be used as shown below.

```
public class @class
{
    public static int MyProperty { get; set; }
}

@class.MyProperty = 100;
```

C# Class and Objects

A class is like a blueprint of a specific object that has certain attributes and features.

For example, a car should have some attributes such as four wheels, two or more doors, steering, a windshield, etc.

It should also have some functionalities like start, stop, run, move, etc. Now, any object that has these attributes and functionalities is a car.

Here, the car is a class that defines some specific attributes and functionalities. Each individual car is an object of the car class. You can say that the car you are having is an object of the car class.

Likewise, in object-oriented programming, a class defines some properties, fields, events, methods, etc. A class defines the kinds of data and the functionality their objects will have.

Define a Class

class can be defined by using the class keyword. Let's define a class named 'Student'.

Example: Define a Class

```
class Student
{

}
```

A class can contain one or more constructors, fields, methods, properties, delegates, and events. They are called class members. A class and its members can have access modifiers such as public, private, protected, and internal, to restrict access from other parts of the program.

Let's add different members to the `Student` class.

Field

A class can have one or more fields. It is a class-level variable that holds a value. Generally, field members should have a private access modifier used with property.

Example: Field

```
class Student
{
    public int id;

}
```

Property

A property encapsulates a private field using setter and getter to assign and retrieve underlying field value.

Example: Property

```
class Student
{
    private int id;

    public int StudentId
    {
        get { return id; }
        set { id = value; }
    }
}
```

Example: Property in C#

```
private int id;

public int StudentId
{
    get { return id; }
    set {
        if (value > 0)
            id = value;
    }
}
```

Method

A method can contain one or more statements to be executed as a single unit. A method may or may not return a value. A method can have one or more input parameters.

Syntax

```
[access-modifier]   return-type   MethodName(type   parameterName1,   type
parameterName2,...)
{


}
```

The following defines the `Sum` method that returns the sum of two numbers.

Example: C# Method

```
public int Sum(int num1, int num2)
{
    var total = num1 + num2;

    return total;
}
```

The following method doesn't return anything and doesn't have any parameters. The return type is `void`.

Example: C# Method

```
public void Greet()
{
    Console.Write("Hello World!");
}
```

The following defines the `GetFullName()` method in the `Student` class.

Example: Method

```
class Student
```

```
{
    public string FirstName { get; set; }
    public string LastName { get; set; }

    public string GetFullName()
    {
        return FirstName + " " + LastName;
    }
}
```

Constructor

A constructor is a special type of method which will be called automatically when you create an instance of a class. A constructor is defined by using an access modifier and class name `<access-modifier> <class-name>(){ }`.

```
class Student
{
    public Student()
    {
        //constructor
    }
}
```

- A constructor name must be the same as a class name.
- A constructor can be public, private, or protected.
- The constructor cannot return any value so cannot have a return type.
- A class can have multiple constructors with different parameters but can only have one parameterless constructor.
- If no constructor is defined, the C# compiler would create it internally.

Objects of a Class

You can create one or more objects of a class. Each object can have different values of properties and field but methods and events behaves the same.

In C#, an object of a class can be created using the `new` keyword and assign that object to a variable of a class type. For example, the following creates an object of the `Student` class and assign it to a variable of the `Student` type.

Example: Create an Object of a Class

```
Student mystudent = new Student();
```

You can now access public members of a class using the `object.MemberName` notation.

Example: Access Members of a Class

```
Student mystudent = new Student();
mystudent.FirstName = "Steve";
mystudent.LastName = "Jobs";

mystudent.GetFullName();
```

You can create multiple objects of a class with different values of properties and fields.

Example: Create Multiple Objects of a Class

```
Student mystudent1 = new Student();
mystudent1.FirstName = "Steve";
mystudent1.LastName = "Jobs";

Student mystudent2 = new Student();
mystudent2.FirstName = "Bill";
mystudent2.LastName = "Gates";
```

C# Variables

Variables in C# are the same as variables in mathematics.

```
<data type> <variable name> = <value>;
```

The followings are naming conventions for declaring variables in C#:

- Variable names must be unique.
- Variable names can contain letters, digits, and the underscore `_` only.
- Variable names must start with a letter.
- Variable names are case-sensitive, `num` and `Num` are considered different names.
- Variable names cannot contain reserved keywords. Must prefix `@` before keyword if want reserve keywords as identifiers.

C# is the strongly typed language. It means you can assign a value of the specified data type. You cannot assign an integer value to string type or vice-versa.

Example: Cannot assign string to int type variable

```
int num = "Steve";
```

Variables can be declared first and initialized later.

Example: Late Initialization

```
int num;
num = 100;
```

A variable must be assigned a value before using it, otherwise, C# will give a compile-time error.

Error: Invalid Assignment

```
int i;
int j = i; //compile-time error: Use of unassigned local variable 'i'
```

The value of a variable can be changed anytime after initializing it.

Example: C# Variable

```
int num = 100;
num = 200;
Console.WriteLine(num); //output: 200
```

Multiple variables of the same data type can be declared and initialized in a single line separated by commas.

Example: Multiple Variables in a Single Line

```
int i, j = 10, k = 100;
```

Try it

Multiple variables of the same type can also be declared in multiple lines separated by a comma. The compiler will consider it to be one statement until it encounters a semicolon `;`.

Example: Multi-Line Declarations

```
int i = 0,
    j = 10,
    k = 100;
```

Try it

The value of a variable can be assigned to another variable of the same data type. However, a value must be assigned to a variable before using it.

Example: Variable Assignment

```
int i = 100;

int j = i; // value of j will be 100
```

Try it

Variables can be used in an expression and the result of the expression can be assigned to the same or different variable.

Example: Variable & Expression

```
int i = 100;

int j = i + 20; // j = 120

i = 200;
j = i + 20;// j = 220

i = 300;
```

```
Console.WriteLine("j = {0}", j);// j = 220
```