Chapter 4: Class and Object

C# is based on the C++ programming language. Hence, the C# programming language has in-built support for classes and objects. Class is nothing but an encapsulation of properties and methods that are used to represent a real-time entity.

For an example, if you want to work with employee's data in a particular application.
The properties of the employee would be the ID and name of the employee. The methods would include the entry and modification of employee data. All of these operations can be represented as a class in C#. In this chapter, we will look at how we can work with classes and objects in C# in more detail.

What are classes and objects

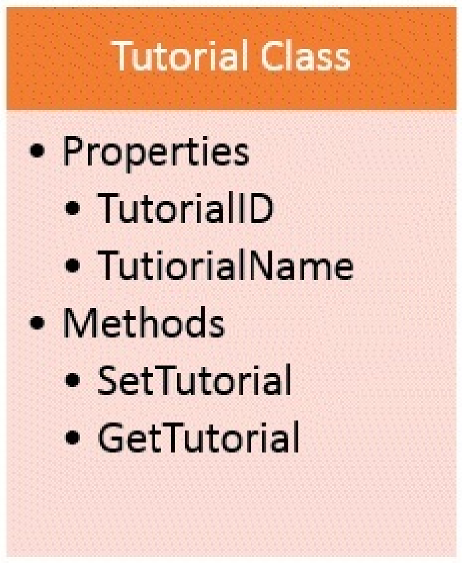
Let's first begin with classes.
As we discussed earlier classes are an encapsulation of **data properties** and **data methods**.
The properties are used to describe the data the class will be holding.
The methods tells what are the operations that can be performed on the data.
To get a better understanding of class and objects, let's look at an example below of how a class would look like.
The name of the class is "Tutorial". The class has the following properties

1. **Tutorial ID** – This will be used to store a unique number which would represent the Tutorial.
2. **Tutorial Name** – This will be used to store the name of the tutorial as a string.

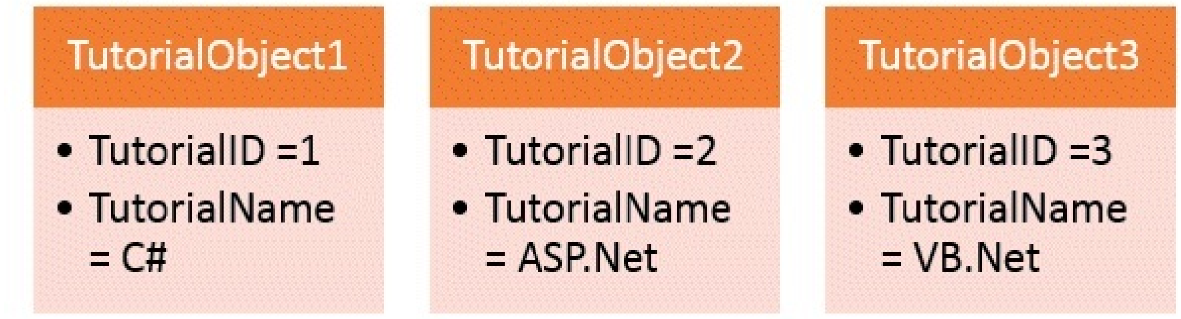
A class also comprises of methods. Our class has the following methods,

1. **SetTutorial** – This method would be used to set the ID and name of the Tutorial. So for example, if we wanted to create a tutorial for .Net, we might create an object for this. The object would have an ID of let’s say 1. Secondly, we would assign a name of “.Net” as the name of the Tutorial. The ID value of 1 and the name of “.Net” would be stored as a property of the object.

2. **GetTutorial** - This method would be used to get the details of a specific tutorial. So if we wanted to get the name of the Tutorial , this method would return the string “.Net”.

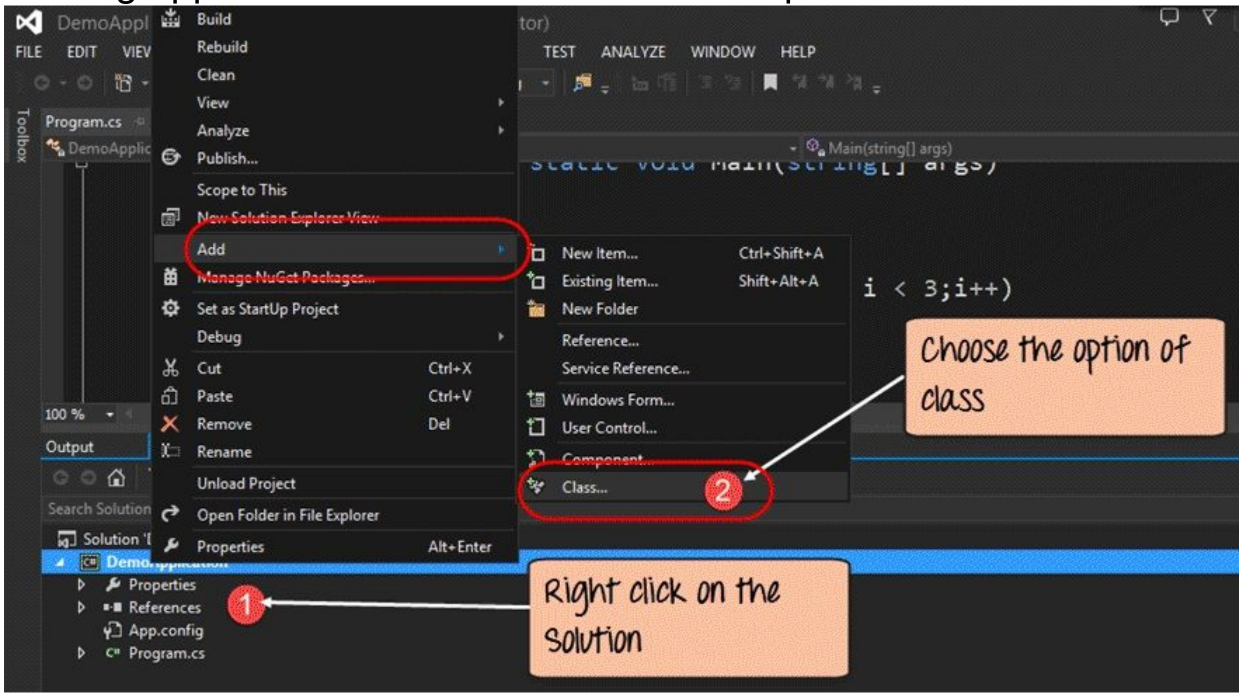


Below is a snapshot of how an object might look like for our Tutorial class. We have 3 objects, each with their own respective TutorialID and TutorialName.



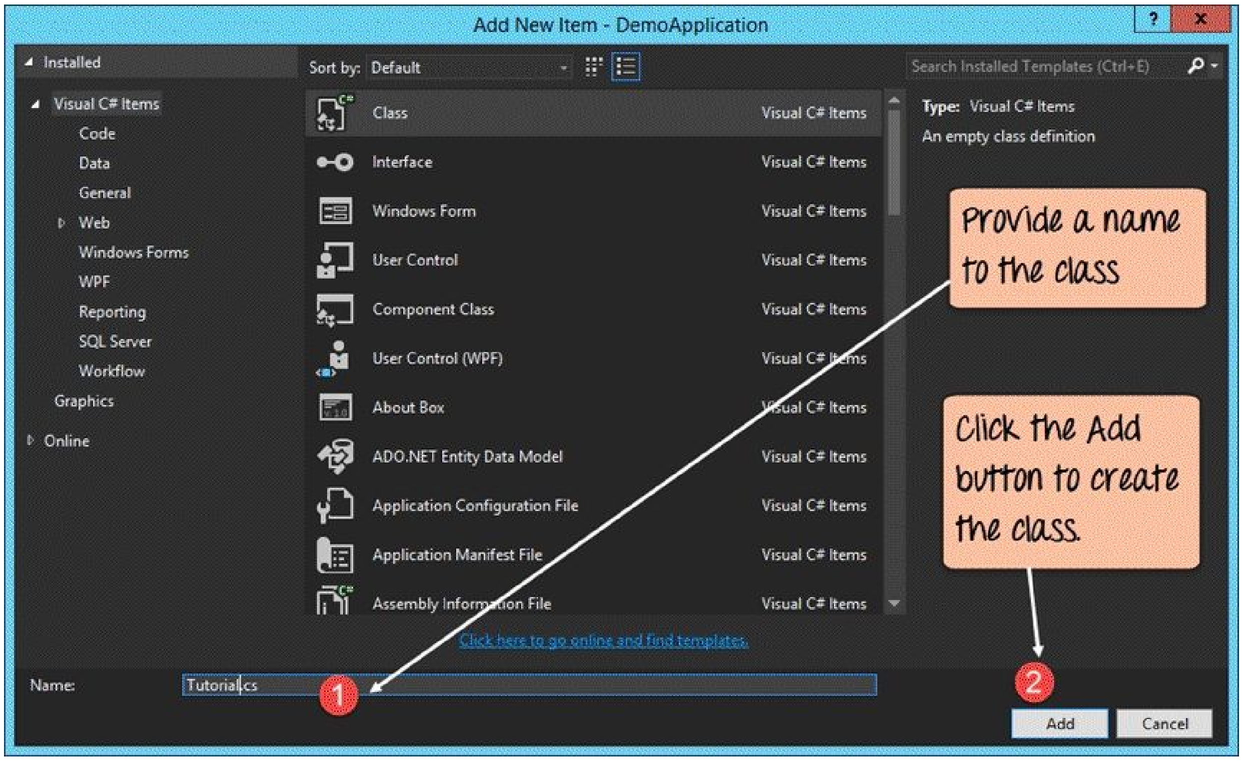
Let’s now dive into Visual Studio to create our class. We are going to build upon our existing console application which was created in our earlier chapter. We will create a class in Visual Studio for our current application.

Let’s follow the below-mentioned steps to get this example in place.
Step 1) The first step involves the creation of a new class within our existing application. This is done with the help of Visual Studio.



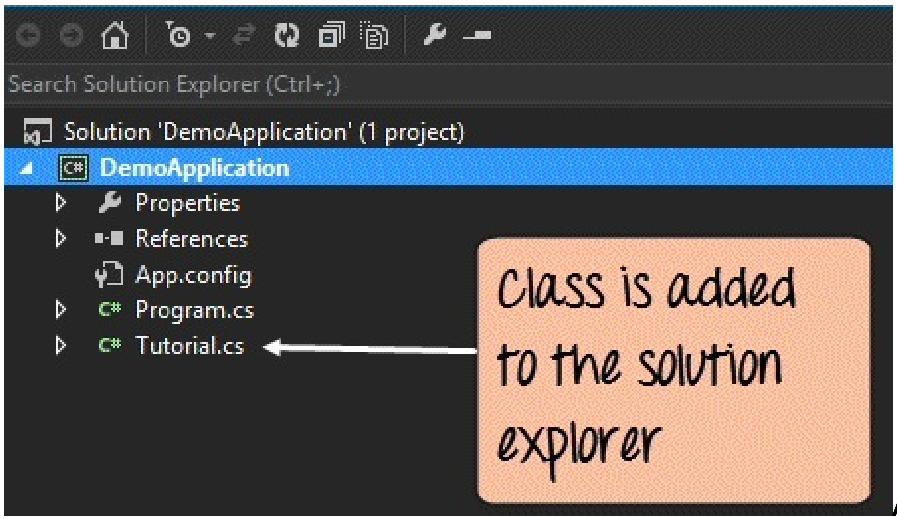
1. The first step is to right click on the solution, which in our case is ‘DemoApplication‘. This will bring up a context menu with a list of options.
2. From the context menu choose the option Add->Class. This will provide the option to add a class to the existing project.

Step 2) The next step is to provide a name for the class and add it to our solution.

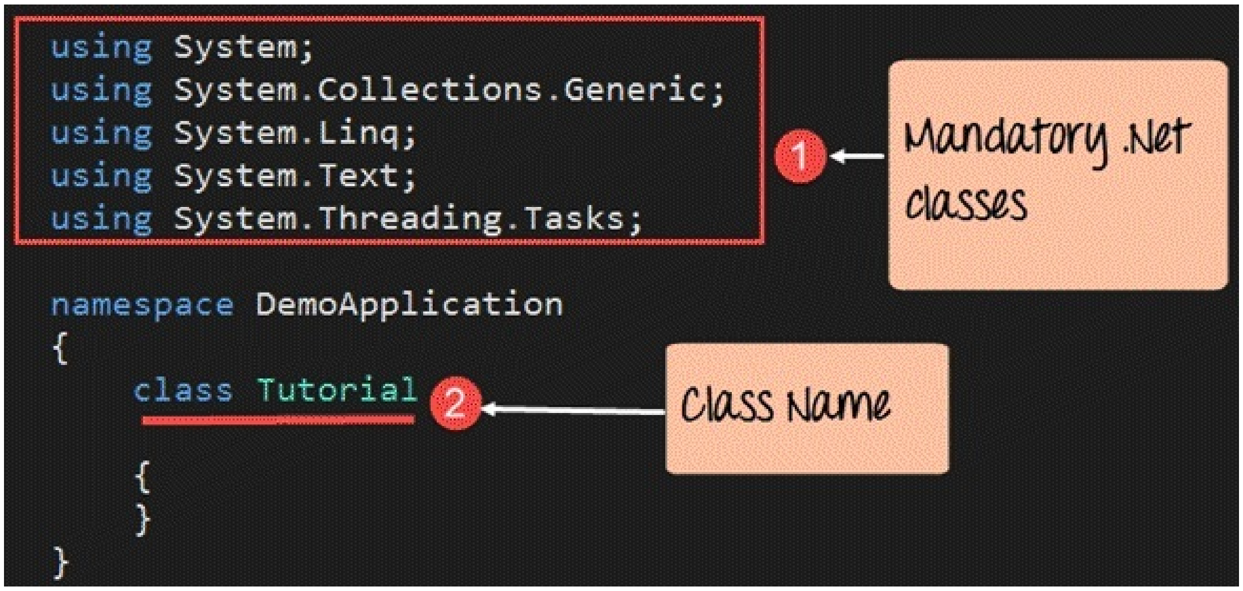

Add New Item - DemoApplication
Installed
Visual C# Items
Code
Data
General
Web
Windows Forms
WPF
Reporting
SQL Server
Workflow
Graphics
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
Class — Visual C# Items
Interface — Visual C# Items
Windows Form — Visual C# Items
User Control — Visual C# Items
Component Class — Visual C# Items
User Control (WPF) — Visual C# Items
About Box — Visual C# Items
ADO.NET Entity Data Model — Visual C# Items
Application Configuration File — Visual C# Items
Application Manifest File — Visual C# Items
Assembly Information File — Visual C# Items
Type: Visual C# Items
An empty class definition
Provide a name to the class
Click the Add button to create the class.
Click here to go online and find templates.
Name: Tutorial.cs
1
2
Add
Cancel

1. In the project dialog box, we first need to provide a name for our class. Let's provide a name of Tutorial.cs for our class. Note that the file name should end with .cs to ensure it is treated as a proper class file.

2. When we click the Add button, the class will be added to our solution.
If the above steps are followed, you will get the below output in Visual Studio. **Output:**


Search Solution Explorer (Ctrl+;)
Solution 'DemoApplication' (1 project)
DemoApplication
Properties
References
App.config
Program.cs
Tutorial.cs
Class is added to the solution explorer

A class named Tutorial.cs will be added to the solution. If you open the file, you will find the below code added to the class file.



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

namespace DemoApplication
{
    class Tutorial
    {
    }
}
```

Code Explanation:

1. The first part contains the mandatory modules which Visual Studio adds to any .Net file. These modules are always required to ensure any .Net program runs on a Windows environment.

2. The second part is the class which is added to the file. The class name is ‘Tutorial’ in our case. This is the name which was specified with the class was added to the solution.

For the moment, our class file does not do anything. In the following topics, we will look into more details on how to work with the class.

Fields and methods

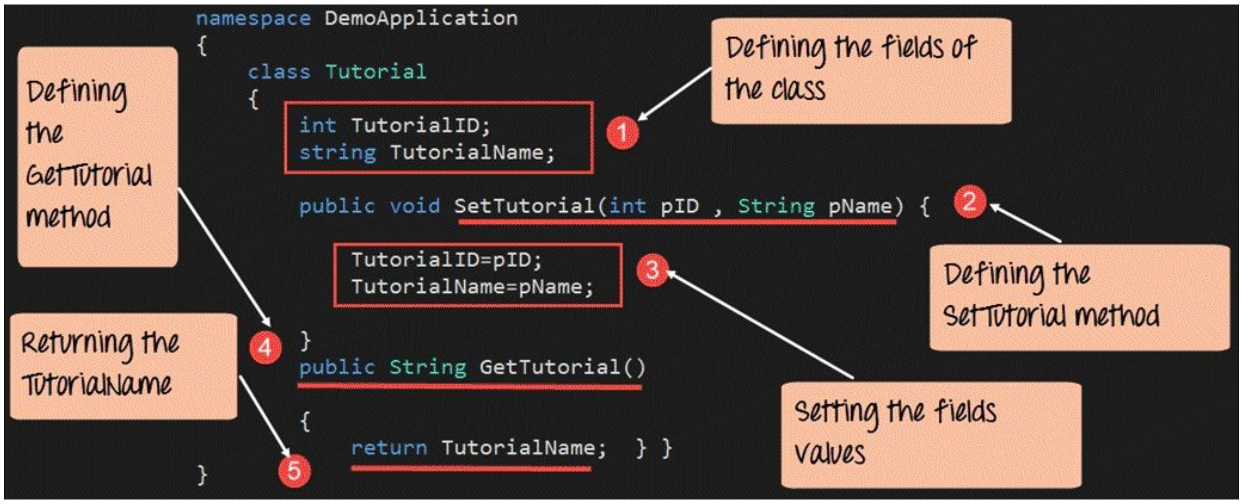
We have already seen how fields and methods are defined in classes in the earlier topic.
For our Tutorial class, we can have the following properties.

1. Tutorial ID – This will be used to store a unique number which would represent the Tutorial.
2. Tutorial Name – This will be used to store the name of the tutorial as a string.

Our Tutorial class can also have the below-mentioned methods. 1. SetTutorial – This method would be used to set the ID and name of the

Tutorial.
2. GetTutorial - This method would be used to get the details of a specific
tutorial.

Let’s now see how we can incorporate fields and methods in our code. **Step 1)** The first step is to ensure the Tutorial class has the right fields and methods defined. In this step, we add the below code to the Tutorial.cs file.



```
namespace DemoApplication
{
    class Tutorial
    {
        int TutorialID;
        string TutorialName;

        public void SetTutorial(int pID , String pName) {

            TutorialID=pID;
            TutorialName=pName;
        }
        public String GetTutorial()
        {
            return TutorialName;  } }
    }
```

Defining the fields of the class
Defining the GetTutorial method
Defining the SetTutorial method
Setting the fields values
Returning the TutorialName
1
2
3
4
5

Code Explanation:

1. The first step is to add the fields of TutorialID and TutorialName to the class file. Since the TutorialID field will be a number, we define it as an Integer, while TutorialName will be defined as string.

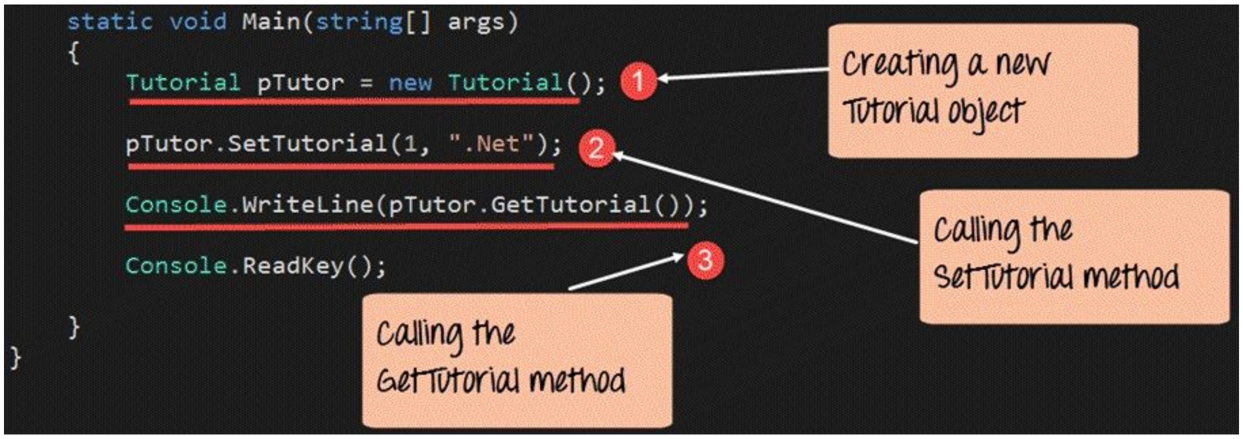
2. Next, we define the SetTutorial method. This method accepts 2 parameters. So if Program.cs calls the SetTutorial method, it would need to provide the values to these parameters. These values will be used to set the fields of the Tutorial object.
Note:-Let's take an example and assume our Program.cs file calls the SetTutorial with the parameters "1" and ".Net". The below steps would be executed as a result of this,
a. The value of pID would become 1
b. The value of pName would be .Net.
c. In the SetTutorial method, these values would then be passed to

TutorialID and TutorialName.
d. So now TutorialID would have a value of 1 and TutorialName would have
a value of ".Net".
3. Here we set the fields of the Tutorial class to the parameters accordingly. So we set TutorialID to pID and TutorialName to Pname.
4. We then define the GetTutorial method to return value of the type "String". This method will be used to return the TutorialName to the calling program. Likewise, you can also get the tutorial id with method

Int GetTutorial 5. Here we return the value of the TutorialName field to the calling program.

Step 2) Now let's add code to our Program.cs, which is our Console application. The Console application will be used to create an object of the "Tutorial class" and call the SetTutorial and GetTutorial methods accordingly.

(**Note**:- An object is an instance of a class at any given time. The difference between a class and an object is that the object actually contains values for the properties.)



```
    static void Main(string[] args)
    {
        Tutorial pTutor = new Tutorial();
        pTutor.SetTutorial(1, ".Net");
        Console.WriteLine(pTutor.GetTutorial());
        Console.ReadKey();
    }
}
```

1
2
3
Creating a new Tutorial object
Calling the SetTutorial method
Calling the GetTutorial method

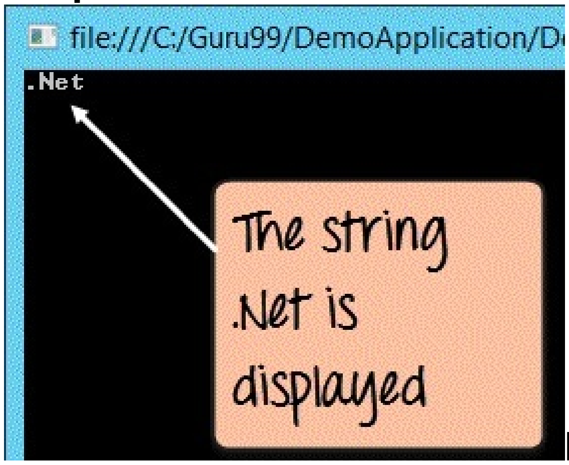
Code Explanation:

1. The first step is to create an object for the Tutorial class. Mark here that this is done by using the keyword 'new'. The 'new' keyword is used to create an object from a class in C#. The object is then assigned to the pTutor variable.

2. The method SetTutorial is then called. The parameters of 1 and ".Net" are passed to the SetTutorial method. These will then be used to set the "TutorialID" and "TutorialName" fields of the class accordingly.

3. We then use the GetTutorial method of the Tutorial class to get the TutorialName. This is then displayed to the console via the Console.WriteLine method.

If the above code is entered properly and the program is run the following output will be displayed.
Output:



```
.Net
```

file:///C:/Guru99/DemoApplication/D
The string .Net is displayed

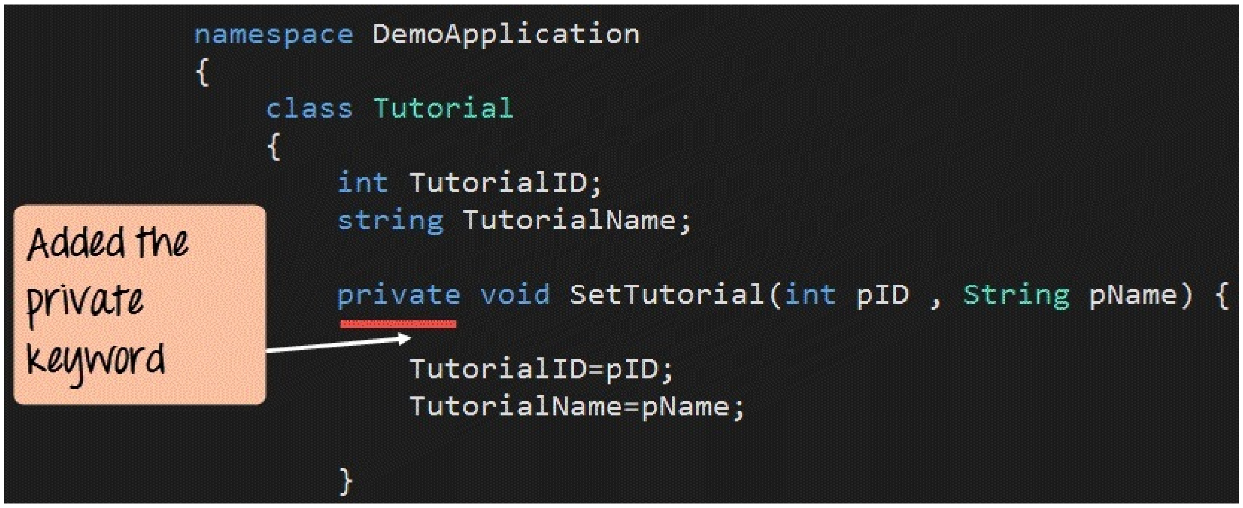
From the output, you can clearly see that the string ".Net" was returned by the GetTutorial method.

Access Modifiers

Access Modifiers are used to define the visibility of a class property or method. There are times when you may not want other programs to see the properties or the methods of class. In such cases, C# gives the ability to put modifiers on class properties and methods. The class modifiers have the ability to restrict access so that other programs cannot see the properties or methods of a class.

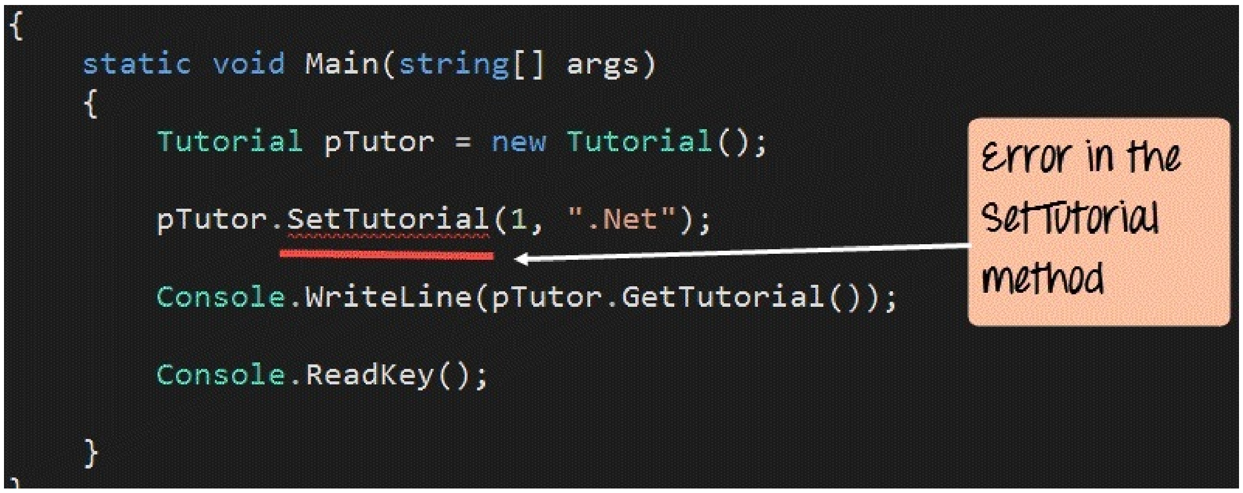
There are generally 3 types of access modifiers. They are explained below.

1. **Private** – When this access modifier is attached to either a property or a method, it means that those members cannot be accessed from any external program.
Let's take an example and see what happens when we use the private access modifier.
Let's modify the current code in our Tutorial.cs file. In the SetTutorial method, let's change the public keyword to private.



```
namespace DemoApplication
{
    class Tutorial
    {
        int TutorialID;
        string TutorialName;

        private void SetTutorial(int pID , String pName) {

            TutorialID=pID;
            TutorialName=pName;

        }
```

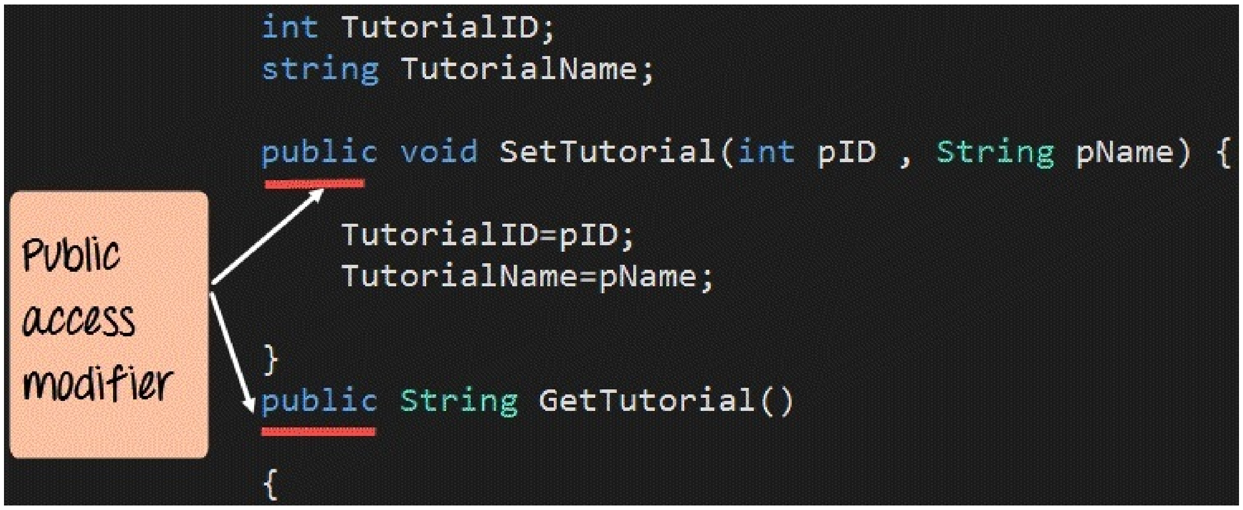
Now let’s switchover to our Program.cs file. You will notice that there is a red squiggly line under the SetTutorial method.
Since we have now declared the SetTutorial method as private in our Tutorial class, Visual Studio has detected this. It has told the user by highlighting it that now this method will not work from the Program.cs file.



```
{
    static void Main(string[] args)
    {
        Tutorial pTutor = new Tutorial();

        pTutor.SetTutorial(1, ".Net");

        Console.WriteLine(pTutor.GetTutorial());

        Console.ReadKey();

    }
```

2. **Public** – When this access modifier is attached to either a property or a

method, it means that those members can be accessed from any external program. We have already seen this in our earlier examples.



```
int TutorialID;
string TutorialName;

public void SetTutorial(int pID , String pName) {

    TutorialID=pID;
    TutorialName=pName;

}
public String GetTutorial()

{
```

Since we have defined our methods as public in the Tutorial class, they have the ability to be accessed from the Program.cs file.

3. **Protected** - When this access modifier is attached to either a property or a method, it means that those members can be accessed only by classes inherited from the current class. This will be explained in more detail in the Inheritance class.

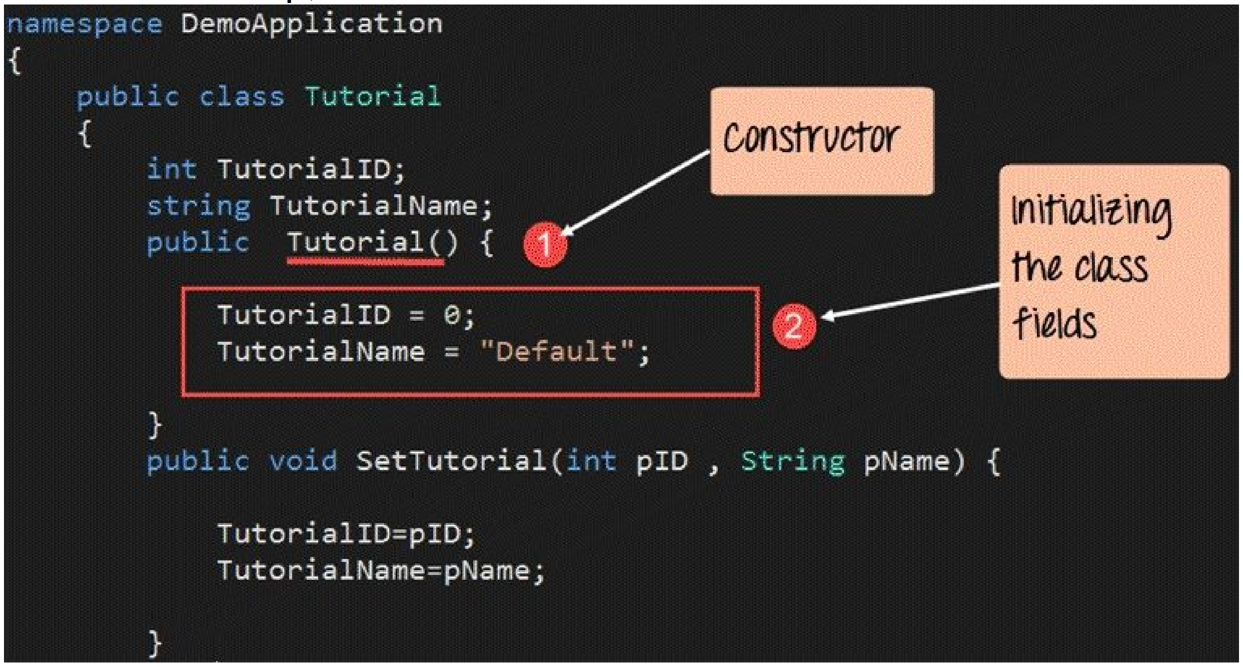
C# Constructor

Constructors are used to initialize the values of class fields when their corresponding objects are created. A constructor is a method which has the same name as that of the class. If a constructor is defined in class, then it will the first method which is called when an object is created. Suppose if we had a class called Employee. The constructor method would also be named as Employee().

The following key things need to be noted about constructor methods 1. The access modifier for the constructor needs to be made as public. 2. There should be no return type for the constructor method.

Let’s now see how we can incorporate the user of constructors in our code. We will use constructors to initialize the TutorialID and TutorialName fields to some default values when the object is created.

Step 1) The first step is to create the constructor for our Tutorial class. In this step, we add the below code to the Tutorial.cs file.



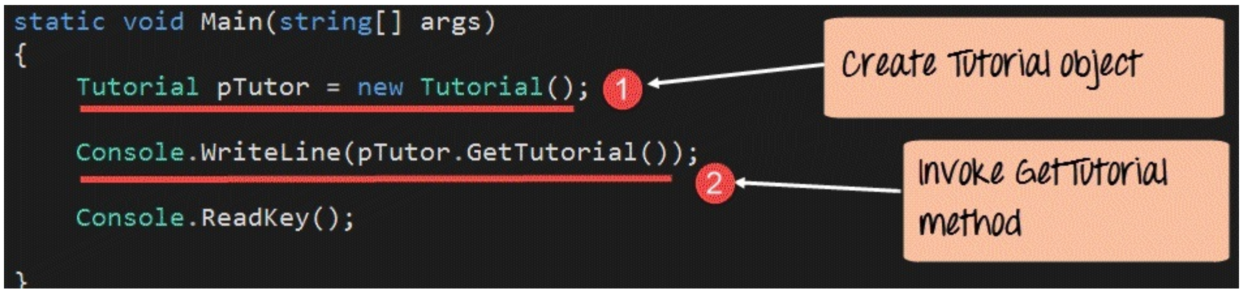
```
namespace DemoApplication
{
    public class Tutorial
    {
        int TutorialID;
        string TutorialName;
        public  Tutorial() {

            TutorialID = 0;
            TutorialName = "Default";

        }
        public void SetTutorial(int pID , String pName) {

            TutorialID=pID;
            TutorialName=pName;

        }
```

Code Explanation:

1. We first add a new method which has the same name as that of the class. Because it is the same name as the class, C# treats this as a constructor method. So now whenever the calling method creates an object of this class, this method will be called by default.

2. In the Tutorial constructor, we are setting the value of TutorialID to 0 and TutorialName to “Default”. So whenever an object is created, these fields will always have these default values.

Now let’s switchover to our Program.cs file and just remove the line, which calls the SetTutorial method. This is because we want to just see how the constructor works.



```
static void Main(string[] args)
{
    Tutorial pTutor = new Tutorial();

    Console.WriteLine(pTutor.GetTutorial());

    Console.ReadKey();
}
```

Code Explanation:

1. The first step is to create an object for the Tutorial class. This is done via the ‘new’ keyword.
2. We use the GetTutorial method of the Tutorial class to get the TutorialName. This is then displayed to the console via the Console.WriteLine method.

If the above code is entered properly and the program is executed, the following output will be displayed.
Output:
From the output, we can see that the constructor was indeed called, and that the value of the TutorialName was set to “Default”.
Note: Here the value “default” is fetched from the constructor.

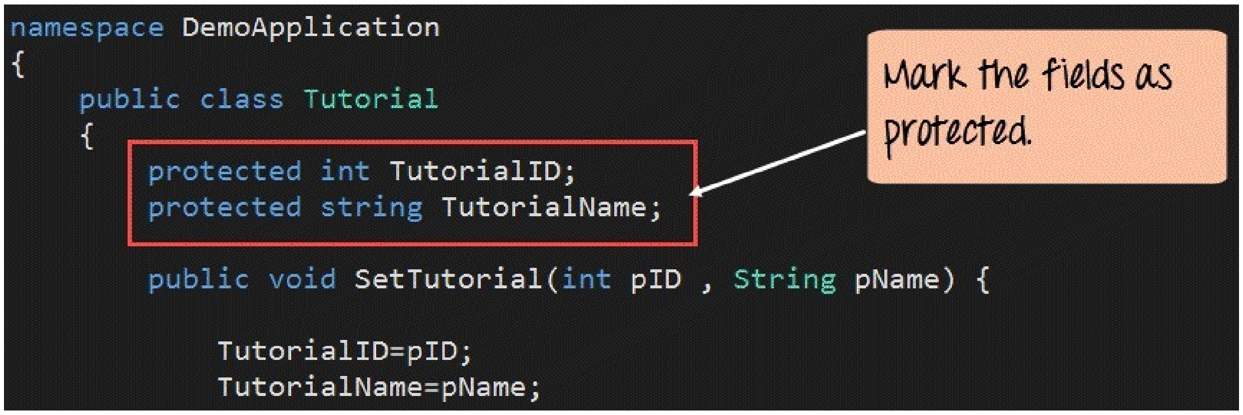
C# Inheritance

Inheritance is an important concept in C#. Inheritance is a concept in which you define parent classes and child classes.

The child classes inherits methods and properties of the parent class, but at the same time, they can also modify the behavior of the methods if required. The child class can also define methods of its own if required.

You will get a better understanding if we see this action.
Let’s now see how we can incorporate the concept of inheritance in our code.

Step 1) The first step is to change the code for our Tutorial class. In this step, we add the below code to the Tutorial.cs file.


```
namespace DemoApplication
{
    public class Tutorial
    {
        protected int TutorialID;
        protected string TutorialName;

        public void SetTutorial(int pID , String pName) {

            TutorialID=pID;
            TutorialName=pName;
```

Note that we need to now add the access modifier of ‘protected’ to both the TutorialID and TutorialName field.

Remember we had mentioned this access modifier in the Access Modifier chapter. Well here you can see the purpose of having this. Only when you have this access modifier (protected), the child class be able to use the fields of the parent class.

Step 2) The second step is to add our new child class. The name of this class will be “Guru99Tutorial”. In this step, we add the below code to the Tutorial.cs file. The code should be placed after the Tutorial class definition.

Code Explanation:

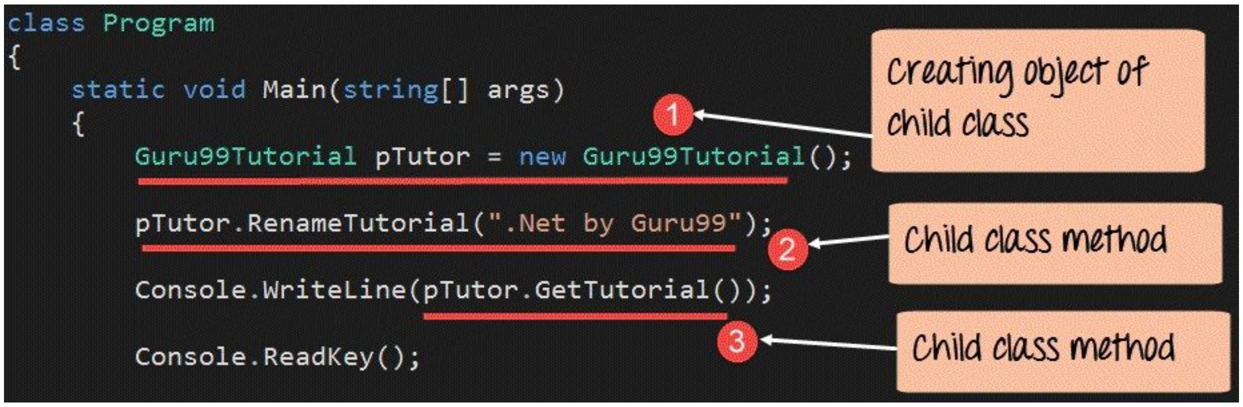
1. The first step is to create the Guru99Tutorial child class. We also need to mention that this class is going to be a child class of the Tutorial class. This is done by the ‘:’ keyword.

2. Next, we are defining a method called RenameTutorial. It will be used to rename the TutorialName field .This method accepts a string variable which contains the new name of the Tutorial.

3. We then assigned the parameter pNewName to the TutorialName field.

Note : - Even though we have not defined the TutorialName field in the “Guru99Tutorial” class, we are still able to access this field. This is because of the fact that “Guru99Tutorial” is a child class of Tutorial class. And because we made the fields of the Tutorial class as protected, they can be accessed by this class.

Step 3) The last step is to modify our main Program.cs file. In our console application, we are going to make an object of the Guru99Tutorial class. With this object, we are going to call the RenameTutorial method. We are then going to display the TutorialName field with the help of the GetTutorial method.



```
class Program
{
    static void Main(string[] args)
    {
        Guru99Tutorial pTutor = new Guru99Tutorial();

        pTutor.RenameTutorial(".Net by Guru99");

        Console.WriteLine(pTutor.GetTutorial());

        Console.ReadKey();
```

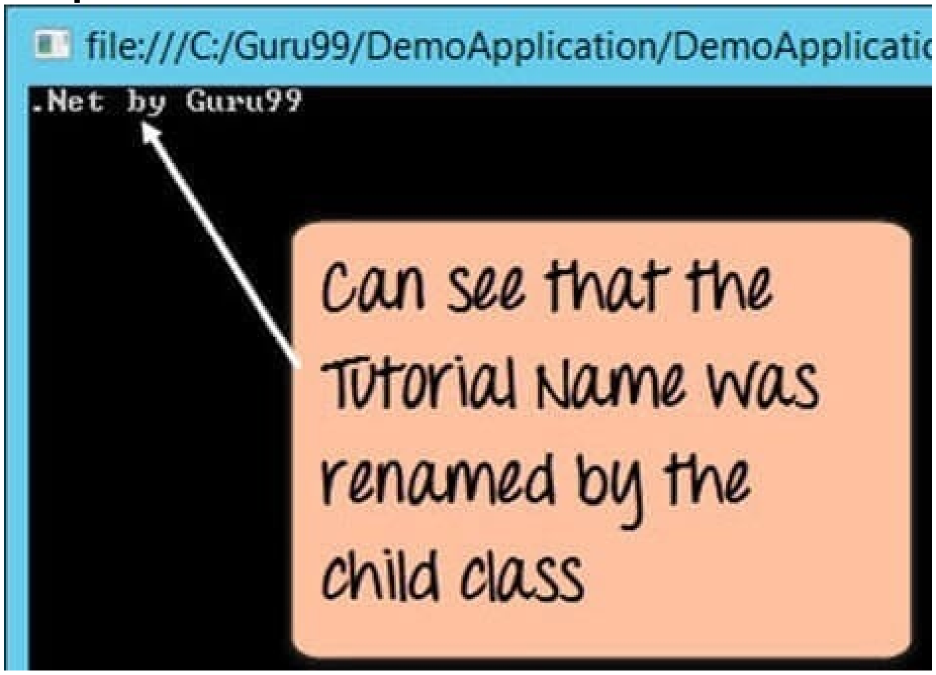
Code Explanation:

1. The first step is to create an object for the Guru99Tutorial class. This is done via the 'new' keyword. Note that this time we are not creating an object of the Tutorial class.

2. We use the RenameTutorial method of the Guru99Tutorial class to change the TutorialName field. We pass the string ".Net by Guru99" to the RenameTutorial method.

3. We then call the GetTutorial method. Note that even though this method is not defined in the Guru99Tutorial class, we are still able to access this method. The output of the GetTutorial method is then displayed to the console via the Console.WriteLine method.
If the above code is entered properly and the program is executed successfully, the following output will be displayed.

Output:


```
.Net by Guru99
```
file:///C:/Guru99/DemoApplication/DemoApplicati
Can see that the Tutorial Name was renamed by the child class

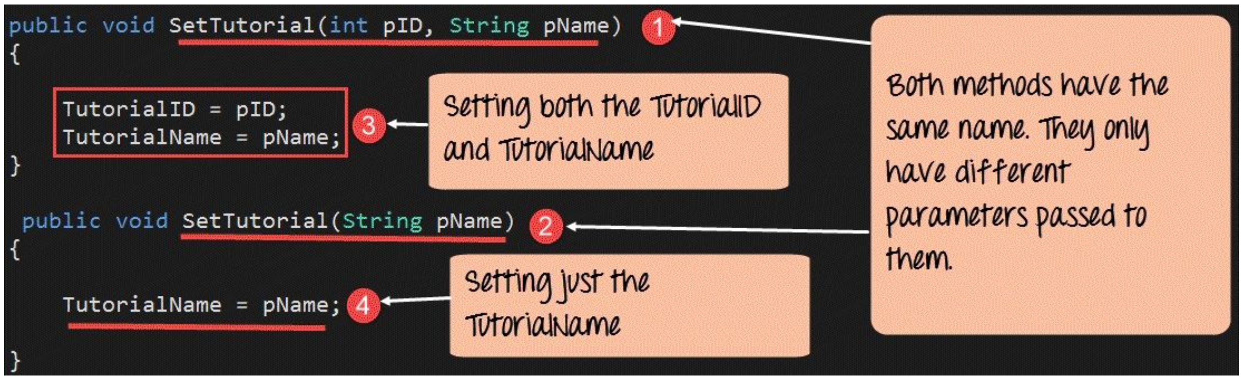
From the output, we can clearly see that the TutorialName field was renamed to “.Net by Guru99”. This was made possible of the RenameTutorial method called by the child class.

C# Polymorphism

Polymorphism is a concept wherein a method can be defined more than one time. But each time, the function would have a different set of parameters passed on to it.

You will get a better understanding if we see this action.
Let’s now see, how we can incorporate the concept of Polymorphism in our code.

Step 1) The first step is to change the code for our Tutorial class. In this step, we add the below code to the Tutorial.cs file.



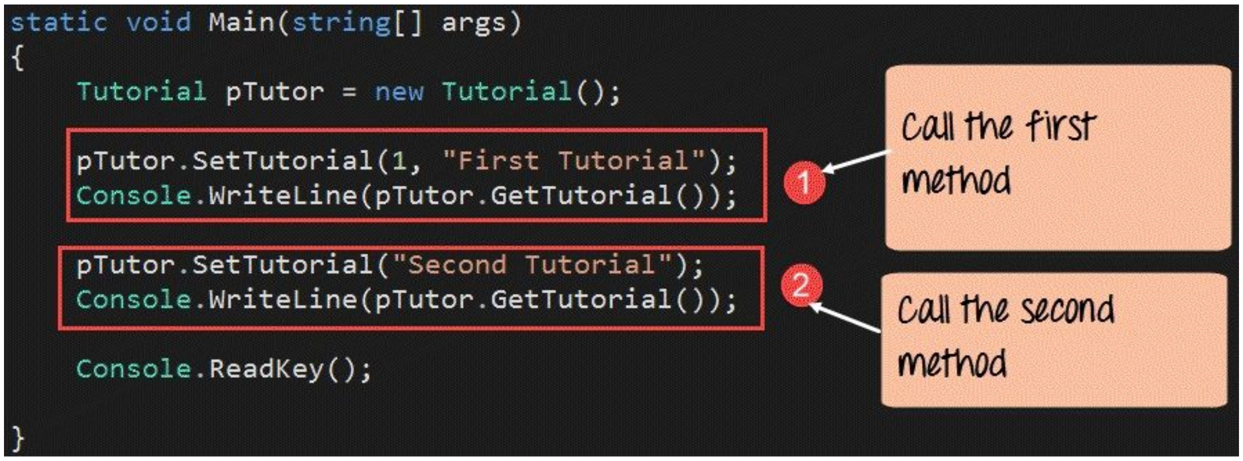
```
public void SetTutorial(int pID, String pName)
{
    TutorialID = pID;
    TutorialName = pName;
}

public void SetTutorial(String pName)
{
    TutorialName = pName;
}
```

1
2
3
4
Setting both the TutorialID and TutorialName
Setting just the TutorialName
Both methods have the same name. They only have different parameters passed to them.

Code Explanation:
1 & 2) The first step is the same as in our earlier examples. We are keeping the definition of the SetTutorial method as it is.
3) This method sets the TutorialID and the TutorialName based on the parameters pID and pName.

4) This is where we make a change to our class wherein we add a new method with the same name of SetTutorial. Only this time we are only passing one parameter which is the pName. In this method, we are just setting the field of TutorialName to pName.

Step 2) The last step is to modify our main Program.cs file. In our console application, we are going to make an object of the Guru99Tutorial class.



```
static void Main(string[] args)
{
    Tutorial pTutor = new Tutorial();

    pTutor.SetTutorial(1, "First Tutorial");
    Console.WriteLine(pTutor.GetTutorial());

    pTutor.SetTutorial("Second Tutorial");
    Console.WriteLine(pTutor.GetTutorial());

    Console.ReadKey();
}
```

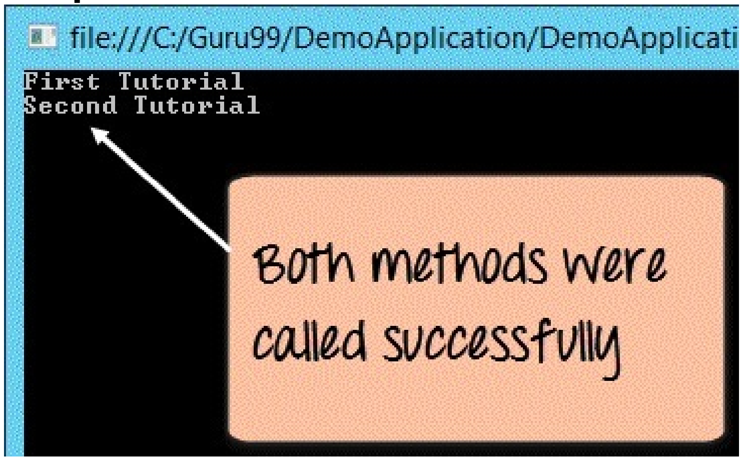
Code Explanation:
1. In the first step, we are using the SetTutorial method with 2 parameters.

Where we are passing both the TutorialID and TutorialName to this method. 2. In the second step, we are now calling the SetTutorial method with just one
parameter. We are just passing the TutorialName to this method.

If the above code is entered properly and the program is run the following output will be displayed. If in case you wanted to also fetch the Tutorial ID along with the Tutorial Name , you should follow the below step

1. Create a separate method called public int GetTutorialID
2. In that method write the code line "return TutorialID." This can be used to return the TutorialID to the calling program.

Output:



```
First Tutorial
Second Tutorial
```

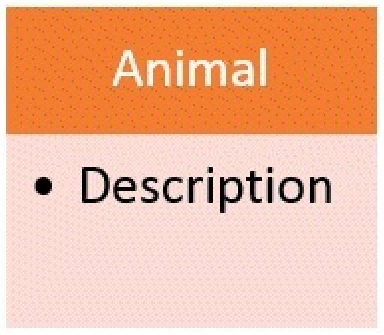
file:///C:/Guru99/DemoApplication/DemoApplicati
Both methods were called successfully

From the output, we can clearly see that both methods were called successfully. Because of this, the strings "First Tutorial" and "Second Tutorial" were sent to the console.

C# Abstract classes

An abstract class is used to define what is known as a base class. A base class is a class which has the most basic definition of a particular requirement.

A typical example of an abstract class is given below. Below is the definition of a class called 'Animal.' When the 'Animal' class is defined, there is nothing known about the animal, whether it is a dog or a cat. The method called description is just a generic method defined for the class.

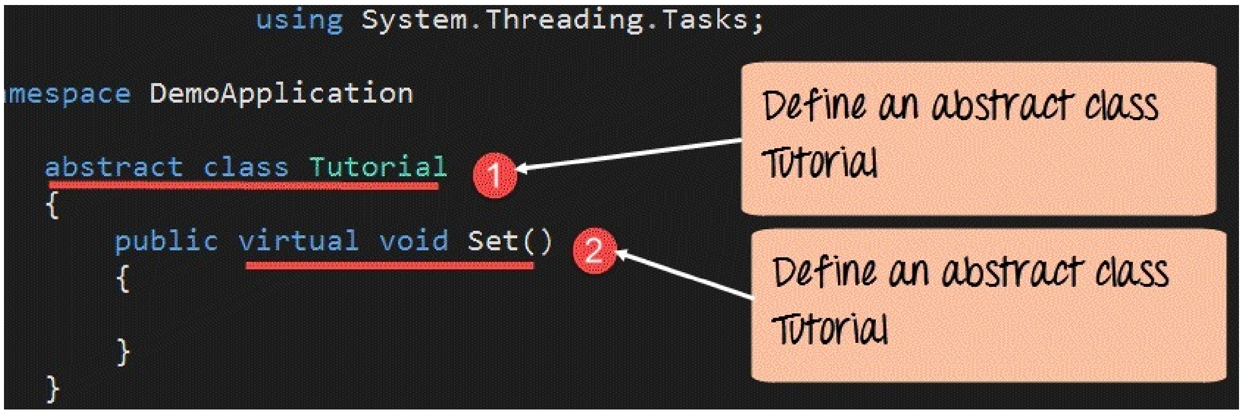


Now when it is known what exactly the Animal is going to be, we create another class which inherits the base class. If we know that the animal is in fact a Dog, we create Dog class which inherits the main base class. The key difference here is that the Dog class cannot change the definition of the Description method of the Animal class. It has to define its own method called Dog-Description. This is the basic concept of abstract classes.

Dog : Animal
• Dog-Description

Let's see how we can change our code to include an abstract class. Note that we will not be running the code, because there is nothing that can be run using an abstract class.

Step 1) As a first step, let's create an abstract class. The class will be called Tutorial and will just have one method. All the code needs to be written in the Program.cs file.



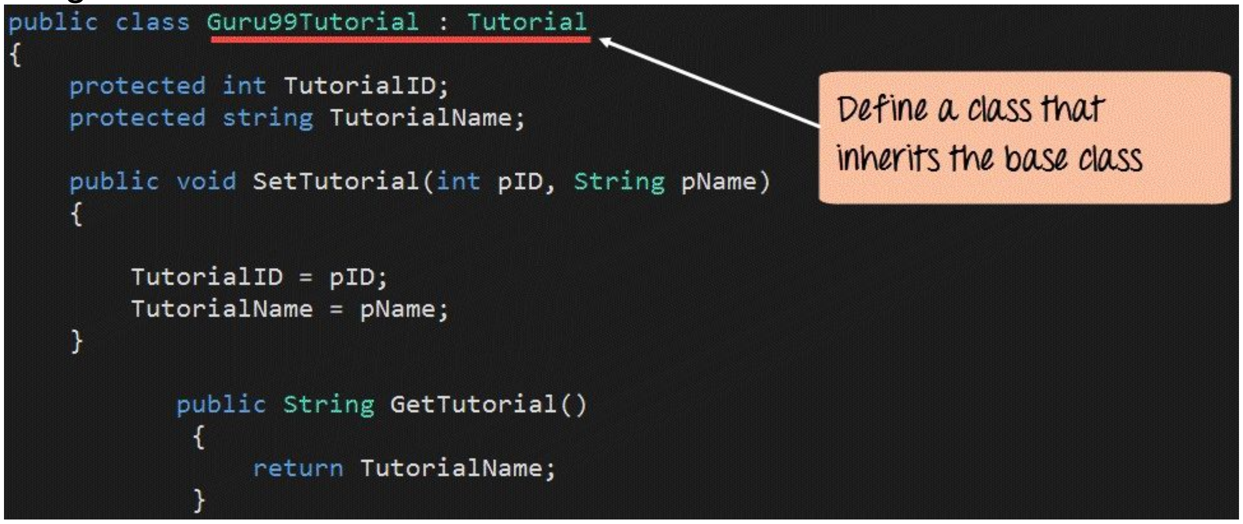
```
              using System.Threading.Tasks;

mespace DemoApplication

  abstract class Tutorial
  {
      public virtual void Set()
      {

      }
  }
```

1
2
Define an abstract class Tutorial
Define an abstract class Tutorial

Code Explanation:

1. We first define the abstract class. Note the use of the abstract keyword. This is used to denote that the class is an abstract class.
2. Next, we are defining our method which does nothing. The method must have the keyword called virtual. This means that the method cannot be changed by the child class. This is a basic requirement for any abstract class.

Step 2) Now let's add our child class. This code is added to the Program.cs file.



```
public class Guru99Tutorial : Tutorial
{
    protected int TutorialID;
    protected string TutorialName;

    public void SetTutorial(int pID, String pName)
    {

        TutorialID = pID;
        TutorialName = pName;
    }

           public String GetTutorial()
            {
                return TutorialName;
            }
```

There is nothing exceptional about this code. We just define a class called

'Guru99Tutorial' which inherits the abstract Tutorial class. We then define the same methods as we have been using from before.

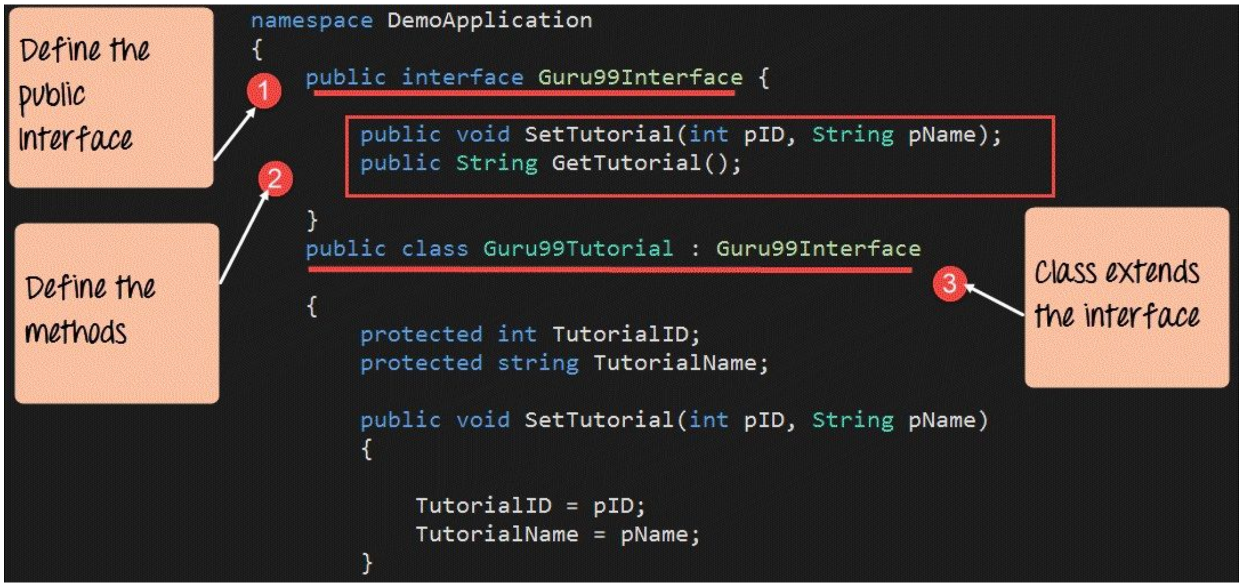
Note: Here we cannot change the definition of the Set method which was defined in the Tutorial class. In the Tutorial class, we had defined a method called 'Set' (public virtual void Set()). Since the method was part of the abstract class, we are not allowed to define the Set method again in the Guru99Tutorial class.

C# Interface

Interfaces are used along with classes to define what is known as a contract. A contract is an agreement on what the class will provide to an application. An interface declares the properties and methods. It is up to the class to define exactly what the method will do.

Let's look at an example of an interface by changing the classes in our Console application. Note that we will not be running the code, because there is nothing that can be run using an interface.

Let's create an interface class. The class will be called "Guru99Interface." Our main class will then extend the defined interface. All the code needs to be written in the Program.cs file.



```
namespace DemoApplication
{
    public interface Guru99Interface {

        public void SetTutorial(int pID, String pName);
        public String GetTutorial();

    }
    public class Guru99Tutorial : Guru99Interface

    {
        protected int TutorialID;
        protected string TutorialName;

        public void SetTutorial(int pID, String pName)
        {

            TutorialID = pID;
            TutorialName = pName;
        }
```

Code Explanation:

1. We first define an interface called “Guru99Interface.” Note that the keyword “interface” is used to define an interface.
2. Next, we are defining the methods that will be used by our interface. In this case, we are defining the same methods which are used in all of earlier examples. Note that an interface just declares the methods. It does not define the code in them.
3. We then make our Guru99Tutorial class extend the interface. Here is where we actually write the code that defines the various methods declared in the interface. This sort of coding achieves the following
It ensures that the class, Guru99Tutorial, only adds the code which is necessary for the methods of “SetTutorial” and “GetTutorial” and nothing else. It also ensures that the interface behaves like a contract. The class has to abide by the contract. So if the contract says that it should have 2 methods called “SetTutorial” and “GetTutorial,” then that is how it should be.

Summary

The class is an encapsulation of data properties and methods. The properties are used to define the type of data in the class. The methods define the operations which can be performed on the data.
A constructor is used to initialize the fields of a class whenever an object is created.
The constructor is a method which has the same names as the class itself. Inheritance is where a child class inherits the fields and methods of the parent class. The child class can then also define its own methods.
Polymorphism is the concept wherein one method can be defined multiple times. The only difference is the number of parameters which are passed to the method.
An abstract class is a base class which has the very basic requirements of what a class should look like. It is not possible for the child class to inherit the methods of the base class.
An interface defines a contract which the class will comply by. The interface defines what are the operations that the class can perform.