

Beginning C# Object Oriented Programming

*This free book is provided by courtesy of [C# Corner](#) and Mindcracker Network and its authors. Feel free to share this book with your friends and co-workers. **Please do not reproduce, republish, edit or copy this book.***

Syed Shanu
(C# Corner MVP)

Index	Page No
Class	1-4
Object	4
Variable	4-5
Method	6-10
Access Modifiers	10-12
Encapsulation	13
Abstraction	13-14
Inheritance	14-17
Polymorphism	18-19
Abstract Class/Method	19-21
Virtual Method	21-23
Sealed Class/Method	23-25
Static Class/Method	25-29
Interface	30-32

About Author

Syed Shanu is basically from Madurai, Tamil Nadu, India. He was been working in South Korea for past 8 years. He has 9+ years of Experience in Microsoft Technologies. His work experience with Language and Technology start's from ASP and SQL Server, Then VB.NET and C# for PDA Application, Touch Screen Application Development, Desktop Application, ASP.NET Web Application Development, MVC and WPF.



Syed Shanu
(C# Corner MVP)

Introduction

There are many explanations to be found in the internet about C# OOP, but here in my article I will provide a very simple example. In this article, I will use a "House" (like the houses we live in) as a realistic example for a better understanding of OOP in C#.

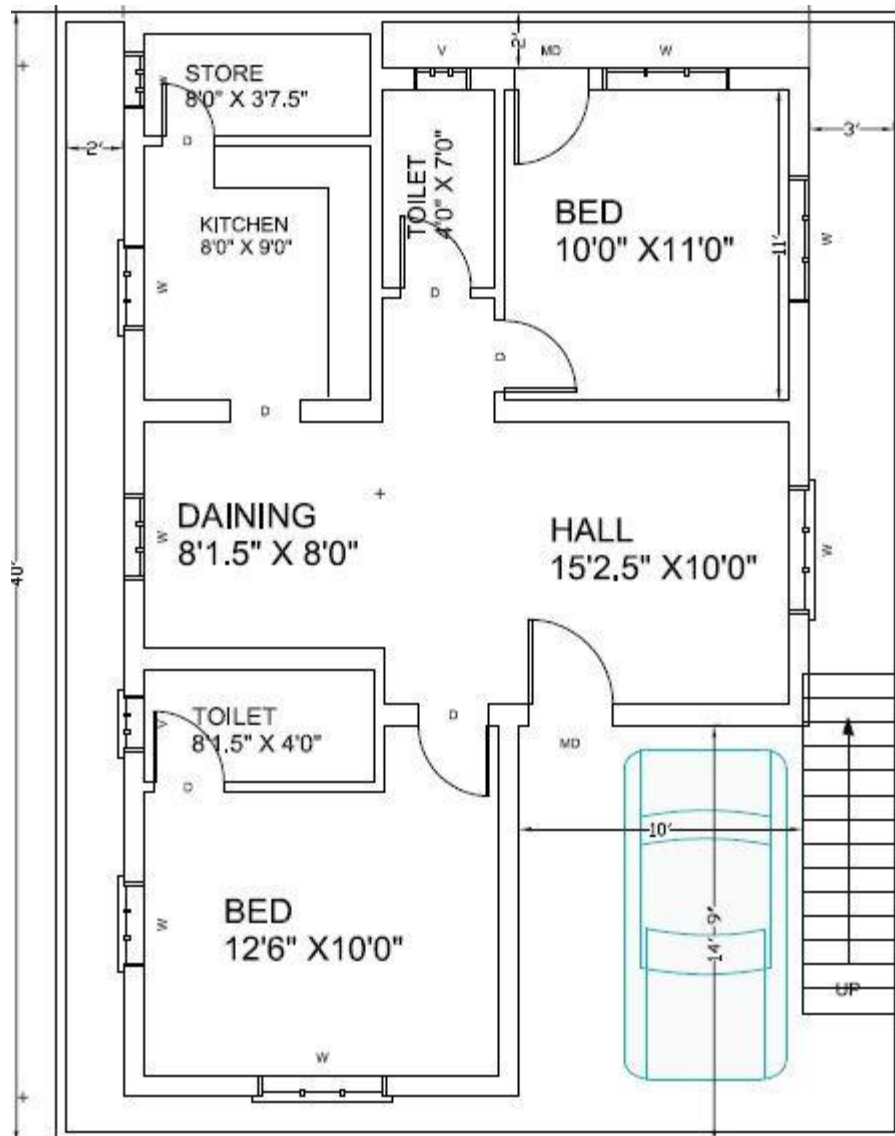
1. Class

A **Class** is a like a blueprint.

What is a blueprint?

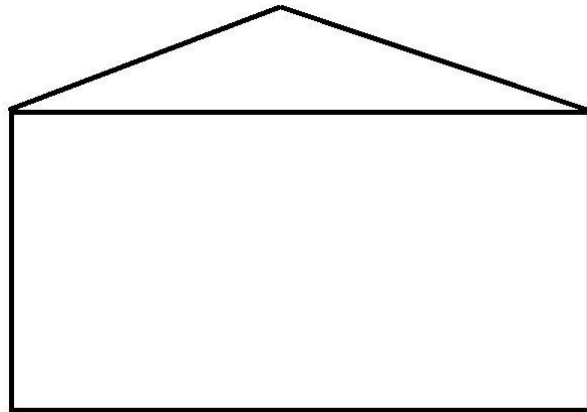
A blueprint is an outline drawing of our actual plan. For example, if we plan to build our new home, the engineer will explain our new house plan with a blueprint as shown in the image below. Once we have finalized the plan the engineer will start building the house with the same plan.

The same as for a blueprint class is an outline of a program. Using the class we can write our own method and declare the variables and using the objects we can access the class method and variables. The class will be complete with variables, methods and objects.



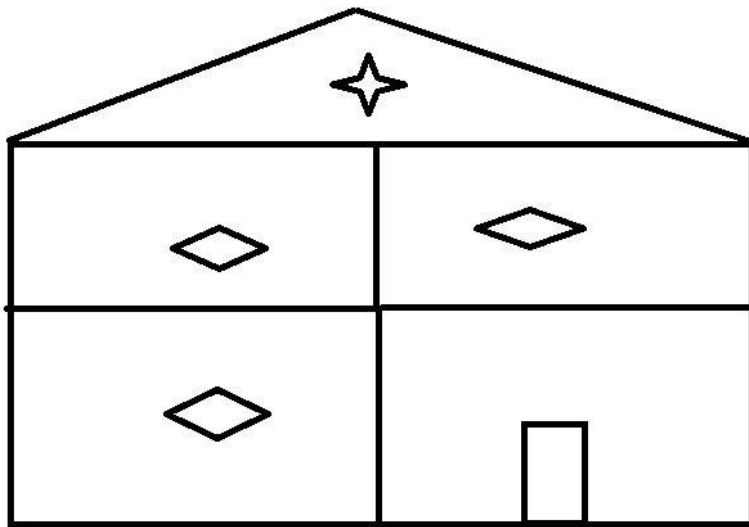
For a better understanding of OOP using a realistic example here I have explained a class for a House. We can imagine a House as an example for a class. In a house, we will have rooms like a living room, bedroom, kitchen and items like TV, refrigerator and so on. The house owner can access and use all the rooms and room's items. The same as for a class with a group of methods and variables. Rooms and each room's items are examples of **methods** and **variables**. So we now have a complete house with all its rooms and room's items. The house owner can access and use all the rooms and room's items. To access and use a class, methods and variables here we use **Objects**. Objects are an instance of a class. We will see the details about objects in the next section.

What will happen if there are no rooms and items in a house? It will be empty and no one can use the house until it has all the rooms and items. See the following image as an example for the empty house.



Now this empty house is a class. So what is the use of a class without methods and variables?

Now let's see an example for a complete house with rooms and items.



So here, we have a complete house. Similarly, the class will be complete with a group of variables, methods and objects. We can see the details of all this in the next sections of this article.

Classes and objects are the fundamental concept of Object Oriented Programming (OOP).

Here is an example of a class. The class should be started with the keyword "class" and next we provide the name for our class. We can provide any meaningful name for the class name. Next we will have the open and close brackets.

```
1. class ShanuHouseClass1
2. {
3. }
```

2. Object

As we have already seen, the house owner can access and use all the rooms of the house and its items. Similarly, to access all the class's method and variables we use objects. We can create one or more objects for the same class. For example we can say for a house there might be one or many owners.

Here is an example of an object. Here "objHouseOwner" is the object of a class that will be used to access all the variables and methods of a class.

```
1. ShanuHouseClass1 objHouseOwner = new ShanuHouseClass1();
```

3. Variable

Variables are used to store our values. Our value can be numeric or text. For example to store a phone number we can use an "int" type variable and to store our name we can use a string type variable with a name for each variable.

Variables can be local or global. For example, we can say if we buy a new TV, TV cable guy will come and set up the TV in our home. He will provide his contact number for future contacts. Usually what we do is take a memo paper and write his contact number and keep it in a common place or in a wallet of ours. If we keep the memo in a common place then everyone visiting our house can see that contact number. Global or public variables are similar to this. If we declare the variable as global then all the methods inside the class can access the

variable. If we store the memo only in our wallet then we can see the contact number. Local private variables are similar to this.

Syntax for variable:

Access-Modifiers Data-Type Variable-Name

By default the access modifiers are private, we can also use public for a variable.

Example of variable:

```
1. int noOfTV = 0;
2. public String yourTVName;
3. private Boolean doYouHaveTV = true;
4.
5. / Example Program
6.
7. class ShanuHouseClass
8. {
9.     int noOfTV = 2;
10.    public String yourTVName = "SAMSUNG";
11.
12.    static void Main(string[] args)
13.    {
14.        ShanuHouseClass objHouseOwner = new ShanuHouseClass();
15.
16.
17.        Console.WriteLine("You Have total " + objHouseOwner.noOfTV + " no of TV :");
18.
19.        Console.WriteLine("Your TV Name is :" + objHouseOwner.yourTVName);
20.    }
21. }
```

In the preceding example program I have declared two variables inside a class. In the main method I created an object of a class. Here we can see the use of the object. We can access the variable of a class and display the output.

The Main method is the default method of C#, where every console and Windows application will start the program execution. In the Main method, we can declare the object for the class and use the object and we can access all the variables and methods of a class. For example, we can say there will be an entrance gate for every house. Using the entrance gate we can enter inside our house. Similarly, to run our program there should be some default program execution starting method. The Main method will be useful in this program execution starting point. Whenever we run our C# Console or windows application, the Main method will first be executed. From the main method we can create an object for our other classes and call their methods.

4. Method or Functions

A method is a group of code statements. Now here we can see the preceding example program with a method.

```
1. class ShanuHouseClass
2. {
3.     int noOfTV = 2;
4.     public String yourTVName = "SAMSUNG";
5.
6.     public void myFirstMethod()
7.     {
8.         Console.WriteLine("You Have total " + noOfTV + "no of TV :");
9.         Console.WriteLine("Your TV Name is :" + yourTVName);
10.        Console.ReadLine();
11.    }
12.
13.    static void Main(string[] args)
14.    {
15.        ShanuHouseClass objHouseOwner = new ShanuHouseClass();
16.        objHouseOwner.myFirstMethod();
17.    }
18. }
```

Note: Most developers were wondering about what the difference is between methods and functions. Both methods and functions are the same. Here in my article, I will use methods instead of functions. However, there is one difference in methods and functions. In OOP languages like C#, Java and so on we use the term method while for non-OOP programming like “C” and so on we use the term function.

The use of methods

Another realistic example. Let's take our mobile phone, we can say that we have a mobile phone and store many songs in it. However, we always like to listen to the selected songs. It will be boring and hard for us to select our favorite song every time and play it. Instead of doing the same work repeatedly, we use the playlist. In the playlist, we can add all our favorite songs. Just click on the playlist of our collections and listen to the music. This will make our work easier and we don't need to do the same thing repetitively. Methods are used like a playlist where we can write all our repeated code in one method and just call the method wherever we needed to.

Here we can see more details about the method.

In a house, there might be one big room or multiple rooms but each room will have different facilities. Similarly in a class we can write one or multiple methods. In a house, there might be two or three bedrooms. Here the room name is Bedroom, but each bedroom can be different by size, color and so on. This means the same rooms with a different type. Similarly, in a class we can create more than one method with the same name but different parameter. In OOP it's called "Polymorphism". We can see the details about Polymorphism later on in this article.

The following is the syntax for the functions:

Access-Modifiers Return-Type Method-Name (Parameter-List)

- **Access-Modifiers:** We will see more details about this topic later on.
- **Return-Type:** If our method returns a value then we use the return type with any data type as string, int and so on. If our method does not return a value then we use the type "Void".
- **Method-Name:** Here we give our name for every method that we create.
- **Parameter-List:** Parameter-List or Arguments that we pass to the function.

Here is an example of a method.

- **Method with Void Type:** Void is a keyword that will not return any data from the method, for example we can see the following method with void type. Here in this method we display all our output using "Console.WriteLine" and have used "Console.ReadLine();" to get the Input. This method has all input and output operations but this method doesn't return a value.

```
1. // Function with void and no parameter -- here void means no return type
2. public void veranda()
3. {
4.     Console.WriteLine("Welcome to Veranda");
5.     Console.WriteLine("How Many Chairs Do you have in your Veranda");
6.     NoofChair = Convert.ToInt32(Console.ReadLine());
7.     Console.WriteLine("I have total " + NoofChair + " Chairs in my Veranda");
8. }
```

- **Method with Return Type:** The methods with return type will return in some result that can be used in our program. For example, here we have a method TVNAME with return type “String”. We can say in our home we might have a TV in our LivingROOM and in the parent's bedroom and also in a child's bedroom. We might have a different TV brand in each room. Suppose we want to know the TV brand name in each room, then we need to enter the same code 3 times. Instead of writing the same code again, we can use a method with return type.

```
1. // Function with Return type as String
2. public string TVNAME()
3. {
4.     Console.WriteLine("Enter Your TV Brand NAME");
5.     YOURTVName = Console.ReadLine();
6.     return YOURTVName;
7. }
```

- **Method with Parameter-List:** So far, we have seen methods without arguments. Arguments are used to pass some data to the method to do our process in a better way. For example, we can say we want to do a painting, to our bedrooms. We need to get the opinions of all the members of the house to understand their choices of color for each bedroom. We can pass the member name and their favorite color as parameter to a method.

```
1. //Function with parameter
2. public void BedRoom(String nameandColor)
3. {
4.     Console.WriteLine(nameandColor);
5. }
```

The same method name with different arguments are called method overloading, here we can see an example of that below. Both methods have the same name but it has different arguments.

```
6. // Same Function Name with Different Parameter
7. public void BedRoom(String MemberName,String Color)
8. {
9.     Console.WriteLine(MemberName + " Like " + Color + "Color");
10. }
```

The complete class with Main method example:

```
11. class ShanuHouseClass1
12. {
```

```
13.     int NoofChair = 0;
14.     public String YOURTVName;
15.     private Boolean DoyouHaveTV = true;
16.
17.
18.
19.     // Function with void and no parameter --
    here void means no return type
20.     public void veranda()
21.     {
22.         Console.WriteLine("Welcome to Veranda");
23.         Console.WriteLine("How Many Chairs Do you have in your Veranda");
24.         NoofChair = Convert.ToInt32(Console.ReadLine());
25.         Console.WriteLine("I have total " + NoofChair + " Chairs in my Veranda");
26.     }
27.
28.
29.     // Function with Return type as String
30.     public string TVNAME()
31.     {
32.         Console.WriteLine("Enter Your TV Brand NAME");
33.         YOURTVName = Console.ReadLine();
34.         return YOURTVName;
35.     }
36.
37.     //Function with parameter
38.     public void BedRoom(String nameandColor)
39.     {
40.         Console.WriteLine(nameandColor);
41.     }
42.
43.     // Same Function Name with Different Parameter
44.     public void BedRoom(String MemberName,String Color)
45.     {
46.         Console.WriteLine(MemberName + " Like " + Color + "Color");
47.     }
48.
49.
50.     static void Main(string[] args)
51.     {
52.         ShanuHouseClass1 objHouseOwner = new ShanuHouseClass1();
53.
54.
55.         objHouseOwner.veranda();
56.         String returnValue = objHouseOwner.TVNAME();
57.         Console.WriteLine("Your TV BRAND NAME IS: "+returnValue);
58.         objHouseOwner.BedRoom("My Name is Shanu I like Lavender color");
59.         objHouseOwner.BedRoom("My Name is Afraz I like Light Blue color");
60.         objHouseOwner.BedRoom("SHANU", "Lavender");
61.         Console.ReadLine();
62.
63.
64.     }
65. }
```

The output of the preceding class is here:

```
file:///D:/Shanu/Shanu Projects/Articles/BasicOOPSConceptofCSharp/ShanuHouseClassConsole/S...
Welcome to Ueranda
How Many Chairs Do you have in your Ueranda
4
I have total 4 Chairs in my Ueranda
Enter Your TV Brand NAME
SAMSUNG LCD
Your TV BRAND NAME IS: SAMSUNG LCD
My Name is Shanu I like Lavender color
My Name is Afraz I like Light Blue color
SHANU Like LavenderColor
```

5. Access Modifiers

Access Modifiers are nothing but the usage and limitation of our type like variables, methods and class. For example we can say it as a security limit.

- private
- public
- internal
- protected
- protected internal

This six are the basic access modifiers we used in our C# class and method and in variables.

Private

Let's take our House example. We will have a security guard in the House, his duty is to guard the entrance to the house. He cannot go inside the house and access all the things. In this case we can create a SecurityGuardClass and declare the variable and method for Security as private.

Public

House owners are public to the class where they can access all the classes related to the House. They have no restrictions to access their house.

Protected

Only the main class and derived classes can have access to protected variables and methods. For example servants and guests are example of protected. For example Servants can go to every room and do cleaning and other activates. However, they have limited access to the house, since they cannot take rest in a bed of the house owner.

Internal

An access limit of a variable or method is within a project. For example let's consider that in our project we have more than one class and we have declared a variable as internal in one class. Let's see an example program for an internal variable.

```
1. public class sampleInternalClass
2. {
3.     internal String myInternal = "Iam Internal Variable";
4. }
5. class ShanuHouseClass
6. {
7.     int noOfTV = 2;
8.     public String yourTVName = "SAMSUNG";
9.
10.    public void myFirstMethod()
11.    {
12.        Console.WriteLine("You Have total " + noOfTV + "no of TV :");
13.        Console.WriteLine("Your TV Name is :" + yourTVName);
14.
15.    }
16.
17.    static void Main(string[] args)
18.    {
19.        ShanuHouseClass objHouseOwner = new ShanuHouseClass();
20.        objHouseOwner.myFirstMethod();
21.        sampleInternalClass intObj = new sampleInternalClass();
22.
23.        Console.WriteLine("Internal Variable Example :" + intObj.myInternal);
24.        Console.ReadLine();
25.
26.    }
27. }
```

Protected Internal

A Protected Internal variable or method has a limitation within a project of a class or a derived class. Here is a sample program for a Protected Internal Variable. In this example I used inheritance. We will see in detail about Inheritance; there are more details in this article.

```
1. public class sampleProtectedInternalClass
2. {
3.     protected internal String myprotectedInternal = "Iam Protected Internal Variable
   ";
4.
5.     public void protectedInternalMethod()
6.     {
7.         Console.WriteLine("Protected Internal Variable Example :" + myprotectedInter
   nal);
8.     }
9. }
10. public class derivedClass : sampleProtectedInternalClass
11. {
12.     public void derivedprotectedInternal()
13.     {
14.         Console.WriteLine("Derived Protected Internal Variable Example :" + myprotec
   tedInternal);
15.     }
16. }
17. class ShanuHouseClass
18. {
19.     int noOfTV = 2;
20.     public String yourTVName = "SAMSUNG";
21.
22.     public void myFirstMethod()
23.     {
24.         Console.WriteLine("You Have total " + noOfTV + "no of TV :");
25.         Console.WriteLine("Your TV Name is :" + yourTVName);
26.
27.     }
28.
29.     static void Main(string[] args)
30.     {
31.         ShanuHouseClass objHouseOwner = new ShanuHouseClass();
32.         objHouseOwner.myFirstMethod();
33.         sampleProtectedInternalClass intObj = new sampleProtectedInternalClass();
34.
35.         intObj.protectedInternalMethod();
36.
37.         derivedClass proIntObj = new derivedClass();
38.         proIntObj.derivedprotectedInternal();
39.         Console.ReadLine();
40.
41.     }
42. }
```

Note: The main and major things we need to understand about OOP are Encapsulation, Abstraction, Polymorphism and Inheritance. We will explain them in detail in this article.

6. Encapsulation

Encapsulation hides the members or variables from outside the class. In our house example, I have already said that a House security guard limitation is at the entrance of the house. The security guard doesn't need to be aware of what is happening inside the house. Therefore, the House Owner will hide everything that happens inside from the security guard for greater safety. The hiding and limitation are called Encapsulation.

For example, we have two classes, the first one is “Houseclass” and the other class “houseSecurityClass”.

Here we can see all the variables are wrapped into a class where “houseSecurityClass” is set to public, so the “Houseclass” can access that, but “houseClass” has both a public and private variable where the private variable of the class cannot be accessed outside of the class.

```
1. public class houseSecurityClass
2. {
3.     public int noofSecurity;
4.     public String SecurityName = String.Empty;
5. }
6.
7. public class Houseclass
8. {
9.     private int noofLockerinHosue = 2;
10.    public string OwnerName = String.Empty;
11. }
```

7. Abstraction

Abstraction shows to and shares some common information with the user. Let's take our House example. In our house we will have a servant. Servants can go to all rooms and do cleaning and other work. The house owner can give full rights or some partial rights to the servant for accessing his house. Here we can see an example program as private declared variables and the methods are not shared with the servant but the public variable and methods are shared with the servant.

```
1. public class HouseServerntClass
2. {
3.
4.     private int SaftyLockerKeyNo = 10001;
5.     public String roomCleanInstructions = "Clean All rooms";
6.
7.     private void saftyNos()
8.     {
```



```
9.         Console.WriteLine("My SaftyLockerKeyNo is" + SaftyLockerKeyNo);
10.     }
11.
12.     public void roomCleanInstruction()
13.     {
14.         Console.WriteLine(roomCleanInstructions);
15.     }
16.
17. }
```

8. Inheritance

Inheritance is used to reuse the code. In the Protected Internal Access modifier section we have already seen an example program for Inheritance. Inheritance is nothing but accessing and using all base class variables and methods in the derived class. Inheritance can be any of the following.

Single level Inheritance: With one base class and one derived class for example.

```
1. public class baseClass
2. {
3.     private void privateMethod()
4.     {
5.         Console.WriteLine("private Method");
6.     }
7.
8.     public void publicMethod()
9.     {
10.        Console.WriteLine("This Method Shared");
11.    }
12. }
13. public class DerivedClass : baseClass
14. {
15.     static void Main(string[] args)
16.     {
17.         DerivedClass obj = new DerivedClass();
18.         obj.publicMethod();
19.         //obj.privateMethod(); //This will be error as private method can not be accessed in Derived Class
20.     }
21.
22. }
```

Note: Sometimes users might be unclear about what a base class is and what a derived class is. A base class is the super-class and a derived class is the class(es) that inherit the base class.

Here we can see a simple inheritance where the base class is the “GuestVist” and the derived class is “**HouseOwnerClass**”. Here the “HouseOwnerClass” class inherits the base class of “**GuestVist**”.

Here the HouseOwnerClass class inherits the base class GuestVist:

```
1. HouseOwnerClass.Here HouseOwnerClass class inherits the base class of GuestVist
2. class GuestVist
3. {
4.     public void Guestwelcomemessage()
5.     {
6.         Console.WriteLine("Welcome to our Home");
7.     }
8.     public void GuestName()
9.     {
10.        Console.WriteLine("Guest name is: Shanu");
11.    }
12.
13. }
14. class HouseOwnerClass : GuestVist
15. {
16.     static void Main(string[] args)
17.     {
18.         HouseOwnerClass obj = new HouseOwnerClass();
19.         obj.Guestwelcomemessage();
20.         obj.GuestName();
21.         Console.ReadLine();
22.     }
23. }
```

Multi level Inheritance: With more than one Derived Class for example. Here we can see an example. The first base class is derived in DerivedClass1 and then the DerivedClass1 is derived in DerivedClass2. Now from DerivedClass2 we can access both **BaseClass's** and DerivedClass1's variables and methods.

```
1. public class baseClass
2. {
3.     private void privateMethod()
4.     {
5.         Console.WriteLine("private Method");
6.     }
7.
8.     public void publicMethod()
9.     {
10.        Console.WriteLine("This Method Shared");
11.    }
12. }
13. public class DerivedClass1 : baseClass
14. {
15.     public void DerivedClass1()
16.     {
17.         Console.WriteLine("Derived Class 1");
18.     }
19. }
20. public class DerivedClass2 : DerivedClass1
21. {
22.     static void Main(string[] args)
```

```
23.     {
24.         DerivedClass2 obj = new DerivedClass2();
25.         obj.PublicMethod();
26.         obj.DerivedClass1();
27.         //obj.PrivateMethod(); //This will be error as private method can not be accessed in Derived Class
28.     }
29.
30. }
```

Multiple Inheritance:

Will .Net Support Multiple Inheritance?

The answer to this question is No. In C#, it's not possible to use Multiple Inheritance in a class.

What is Multiple Inheritance? Multiple Inheritance is nothing but having more than one class and we can inherit both classes in our derived class.

What will happen if I write a Multiple Class Inheritance Using C#?

Let's take our example House. Here we have the derived class "HouseOwnerClass" with the two additional classes "GuestVist" and "FriendsandRelationsClass".

Now suppose for our house both Guest and Friend have visited. For this, we write the preceding three classes and inherit the two classes in our derived class.

When I write the Multiple Inheritance in C #, it will display the warning message during our code and execute our program. The Warning message will be "Expected and interface".

See the following image that shows the warning error message while I write Multiple Inheritance.

```
class GuestVist
{
    public void Guestwelcomemessage()
    {
        Console.WriteLine("Welcome to our Home");
    }
    public void GuestName()
    {
        Console.WriteLine("Guest name is: Shanu");
    }
}
```

```
class FriendsandRelationsClass
{
    public void friendwelcomemessage()
    {
        Console.WriteLine("Welcome to our Home");
    }
    public void FriendName()
    {
        Console.WriteLine("Friend name is: Afraz");
    }
}
```

Here we can see i have write the Multiple Inheritacne ,displaying the warning message as "Expected an interface"

```
class HouseOwnerClass : GuestVist, FriendsandRelationsClass
{
    static void Main(string[] args)
    {
        HouseOwnerClass obj = new HouseOwnerClass();
        obj.Guestwelcomemessage();
        obj.GuestName();
        Console.ReadLine();
    }
}
```

class MultipleinheritanceExample.FriendsandRelationsClass
C#: Expected an interface

Then how can we use Multiple Inheritance?

Interface will be used for Multiple Inheritance.

9. Polymorphism

Poly means more than one form. In the beginning of the article in the Method Section, we have already seen an example of Polymorphism. The same method name with a different parameter is an example of polymorphism.

Method Overloading and **Method Overriding** will be used in polymorphism. Polymorphism has the two types of execution, one is Compile Time Polymorphism and the other one is called Run time Polymorphism.

- **Method Overloading**

Method overloading is nothing but the same method name used for more than one method with different argument(s).

Here is an example program of Method Overloading. As we can see here, the method name “BedRoom” has been used for two methods but the parameter for both methods are different.

```
1. class HouseOwnerClass
2. {
3.     //Function with parameter
4.     public void BedRoom(String nameandColor)
5.     {
6.         Console.WriteLine(nameandColor);
7.     }
8.
9.     // Same Function Name with Different Parameter
10.    public void BedRoom(String MemberName, String Color)
11.    {
12.        Console.WriteLine(MemberName + " Like " + Color + "Color");
13.    }
14.
15.    static void Main(string[] args)
16.    {
17.        HouseOwnerClass objHouseOwner = new HouseOwnerClass();
18.
19.        objHouseOwner.BedRoom("My Name is Shanu I like Lavender color");
20.        objHouseOwner.BedRoom("My Name is Afraz I like Light Blue color");
21.        objHouseOwner.BedRoom("SHANU", "Lavender");
22.        Console.ReadLine();
23.
24.    }
25. }
```

- **Method Overriding**

The difference between Method Overloading and Overriding are that in Method Overloading we will have the same method name with a different argument.

In Method Overriding we will have the same Method Name and same argument and same type but method overriding can only be used in the derived class. Method Overriding cannot be done in the same class.

We will see how Method Overriding can be used in **Abstract Method**, **Virtual Method** and in a **Sealed Method**. Kindly refer to that section in this article.

10. Abstract Class/Method

Abstract Class: An Abstract class will have the keyword “abstract”.

```
1. abstract class GuestVist
2. {
3. }
```

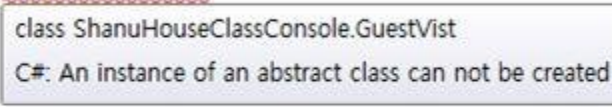
The Abstract class will be a super class for all our classes. An Abstract class cannot be accessed by an object. That means we cannot create an object for an abstract class.

What will happen when we create an object for an Abstract Class

Here we can see the error warning message “An instance of an abstract class cannot be created” when I try to create an object for my abstract class.

```
static void Main(string[] args)
{
    GuestVist objnew = new GuestVist();
}

abstract class GuestVist
{
    public int noofSecurity;
    public String SecurityName = String.Empty;
}
```



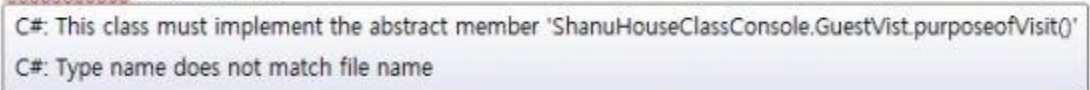
An Abstract class can have both an Abstract Method and a normal method. In an Abstract Class at least one Abstract Method should be declared. In addition, the derived class should override the abstract method. To access the abstract method we should use the “override” keyword in our derived class.

What will happen if we create an abstract method not overridden in the derived class?

Here we can see an abstract class with an abstract method, but the abstract method is not overridden in the derived class. See the following image for the warning message displaying as class must implement the abstract member.

```
abstract class GuestVist
{
    public abstract void purposeofVisit();
}

public class Houseclass : GuestVist
{
    =
```



C#: This class must implement the abstract member 'ShanuHouseClassConsole.GuestVist.purposeofVisit()'
C#: Type name does not match file name

Here we can see an example of an Abstract class and an Abstract method in detail.

In this example, we can see an Abstract class that has a normal method and Abstract method. Abstract methods do not have a body part in the Abstract class that means we can only declare an Abstract method in an Abstract class. There should be a minimum of one Abstract Method in an Abstract Class.

```
1. abstract class GuestVist
2. {
3.     public void Guestwelcomemessage()
4.     {
5.         Console.WriteLine("Welcome to our AbstractHome");
6.     }
7.     public void GuestName()
8.     {
9.         Console.WriteLine("Guest name is: Abstract");
10.    }
11.    public abstract void purposeofVisit();
12.
13. }
14. // derived class to inherit the abstract class
15. public class Houseclass : GuestVist
16. {
```

```
17.     static void Main(string[] args)
18.     {
19.         Houseclass objHouse = new Houseclass();
20.         objHouse.Guestwelcomemessage();
21.     }
22.
23.     public override void purposeofVisit()
24.     {
25.         Console.WriteLine("Abstract just came for a Meetup and spend some time ");
26.     }
27. }
```

11. Virtual Class/Method

A Virtual method is very useful in our day-to-day programming.

What a Virtual method is and what the use of a Virtual method is

Take our House example. A guest confirms, since today a total of five people will visit your home. For this, we write a function to display a message since there are five guests visiting our home. Once a guest visits, we see there are a total of 20 persons that have visited. In some cases it might increase or decrease; we will know when they reach us.

In that case, the guest will be a separate class and House will be a separate class. Without changing the message in the Guest class, how can we change the data in our Derived class?

The difference between an Abstract method and a Virtual method

Both similarities use the override keyword. An Abstract Method can only be declared in an Abstract Class. That means no body part for an abstract method in an Abstract class. However, for virtual it can have a body part.

See the example program below. Here we have both an Abstract Method and a Virtual Method. In the Abstract class, the virtual method says total five guests but in the derived Class program, the message was overridden as 20 guests. See the final output in the following. Guess what will be displayed for the Virtual method? Will the result be 5 Guests or 20 Guests? Check the output below the program.


```
1. abstract class GuestVist
2. {
3.     public abstract void purposeofVisit(); // Abstract Method
4.
5.     public virtual void NoofGuestwillvisit() // Virtual Method
6.     {
7.         Console.WriteLine("Total 5 Guest will Visit your Home");
8.     }
9. }
10. class AbstractHouseClass : GuestVist
11. {
12.     public override void purposeofVisit() // Abstract method Override
13.     {
14.         Console.WriteLine("Abstract just for a Meetup and spend some time ");
15.     }
16.
17.     public override void NoofGuestwillvisit() // Virtual method override
18.     {
19.         Console.WriteLine("Total 20 Guest Visited our Home");
20.     }
21.
22.     static void Main(string[] args)
23.     {
24.         AbstractHouseClass objHouse = new AbstractHouseClass();
25.
26.         objHouse.purposeofVisit();
27.         objHouse.NoofGuestwillvisit();
28.         Console.ReadLine();
29.     }
30.
31. }
```

The Complete Program

```
1. abstract class GuestVist
2. {
3.     public void Guestwelcomemessage()
4.     {
5.         Console.WriteLine("Welcome to our AbstractHome");
6.     }
7.     public void GuestName()
8.     {
9.         Console.WriteLine("Guest name is: Abstract");
10.    }
11.    public abstract void purposeofVisit(); // Abstract Method
12.    public virtual void NoofGuestwillvisit() // Virtual Method
13.
14.    {
15.        Console.WriteLine("Total 5 Guest will Visit your Home");
16.    }
17. }
18. class AbstractHouseClass : GuestVist
19. {
```

```
20.     public override void purposeofVisit() // Abstract method Override
21.     {
22.         Console.WriteLine("Abstract just for a Meetup and spend some time ");
23.     }
24.
25.     public override void NoofGuestwillvisit() // Virtual method override
26.     {
27.         Console.WriteLine("Total 20 Guest Visited our Home");
28.     }
29.
30.     static void Main(string[] args)
31.     {
32.         AbstractHouseClass objHouse = new AbstractHouseClass();
33.         objHouse.Guestwelcomemessage();
34.         objHouse.purposeofVisit();
35.         objHouse.NoofGuestwillvisit();
36.         Console.ReadLine();
37.     }
38. }
```

Output



```
file:///D:/Shanu/Shanu Projects/Articles/BasicOOPSConceptofC...
Welcome to our AbstractHome
Abstract just for a Meetup and spend some time
Total 20 Guest Visited our Home
```

12. Sealed Class/Method

Sealed Class: As the name indicates, this class cannot be inherited by other classes.

Take our House example. In a house, the house owner can have a secret room, perhaps an official or a financial room. The owner doesn't want others to access his official room. The sealed class will be useful in those cases.

A Sealed class can be declared using the keyword Sealed. If one class is declared Sealed, it cannot be inherited in other derived classes.

What will happen when we inherit a sealed class in our derived class

Let's see an example of attempting to inherit my sealed class from my derived class. It shows me the following warning message.

```

public sealed class OwnerofficialRoom
{
    public void AllMyPersonalItems()
    {
        Console.WriteLine("All Items in this rooms are personal to me no one else can access or inherit me");
    }
}
class HouseSealedClass : OwnerofficialRoom
{
    static void Main(str
    {
    }
}

```

```

class SealedClassConsole.OwnerofficialRoom
C#: The sealed class 'SealedClassConsole.OwnerofficialRoom' can not be subclassed

```

Here we can see an example program of the Sealed Class.

```

1. public sealed class OwnerofficialRoom
2. {
3.     public void AllMyPersonalItems()
4.     {
5.         Console.WriteLine("All Items in this rooms are personal to me no one else ca
6.         n access or inherit me");
7.     }
8. }
9. class HouseSealedClass
10. {
11.     static void Main(string[] args)
12.     {
13.         OwnerofficialRoom obj = new OwnerofficialRoom();
14.         obj.AllMyPersonalItems();
15.         Console.ReadLine();
16.     }
17. }

```

Sealed Method: If we declared a method as sealed then that specific method cannot be overridden in the derived class.

Let's see our house class here I have base class with Virtual Method and virtual Sealed methods.

The Virtual method can be overridden in the derived class. But the Virtual Sealed Method cannot be overridden in the sealed class.

```

1. //Base Class with Sealed Method
2. public class OwnerOfficialroomwithrestriction
3. {
4.     public virtual void message()
5.     {
6.         Console.WriteLine("Every one belongs to this house can access my items in my
7.         room except my sealed Item");
8.     }
9. }

```

```
10.     public virtual sealed void myAccountsLocker()
11.     {
12.         Console.WriteLine("This Locker can not be inherited by other classes");
13.     }
14. }
15. //Derived method which inherits Sealed Method class
16. class HouseSealedClass : OwnerOfficialroomwithrestriction
17. {
18.     public override void message()
19.     {
20.         Console.WriteLine("overriden in the derived class");
21.     }
22.
23.     public override void myAccountsLocker()
24.     {
25.         Console.WriteLine("The sealed method Overrides");
26.     }
27. }
```

13. Static Class/Method

We have already learned about what a Sealed Class is in this article. Now let's see what a Static class and Static method are. Both Static and Sealed Class cannot be inherited.

The Difference between a Static Class and a Sealed Class

We can create an object (instance) of the Sealed Class. We can see in my sealed class section I have created a sample Sealed class and in the Main Method I created an object to access the sealed Class. And in a Sealed Class both Static and non-Static methods can be written.

But for a Static Class it's not possible to create an Object. In a Static Class only Static members are allowed. That means in a static class it's not possible to write non-static methods.

We can say our Main method is an example of a Static method. When we create a console application in C# we can see each class has a default Main method. In my article I have explained that when a Console or Windows application begins execution, first the Main method will be executed. There is no need to create an object of the Main method since it was declared as a static method.

```
1. static void Main(string[] args)
2. {
3. }
```

Something interesting about a Static class is that memory will be allocated for all static variables and methods during execution but for non-static variables and methods memory will be allocated only when the object for the class is created.

Let's take our same sealed class example for our static class and method.

What happens when we inherit a Static class in a derived class

Let's see an example when I try to inherit my static class from my derived class. It shows me the following warning message.

```
public static class OwnerofficialRoom
{
    public static void AllMyPersonalItems()
    {
        Console.WriteLine("All Items in this rooms are personal to me no one else can access or inherit me");
    }
}

class HouseStaticClass : OwnerofficialRoom
{
    static void Main(str
    {
    }
}
```

class StaticClass.OwnerofficialRoom
C#: 'StaticClass.OwnerofficialRoom': cannot derive from static class 'StaticClass.HouseStaticClass'

What will happen when we declare a non-Static method in a Static class

Let's see an example of attempting to create a non-Static method at my Static Class.

```
public static class OwnerofficialRoom
{
    public static void AllMyPersonalItems()
    {
        Console.WriteLine("All Items in this rooms are personal to me no one else can access or inherit me");
    }
    public void lam_non_Static_Method()
    {
    }
}
```

C#: Cannot declare instance members in a static class
C#: Name does not match the naming convention. Suggested name 'lamNonStaticMethod'

What will happen when we create an object for the Static class

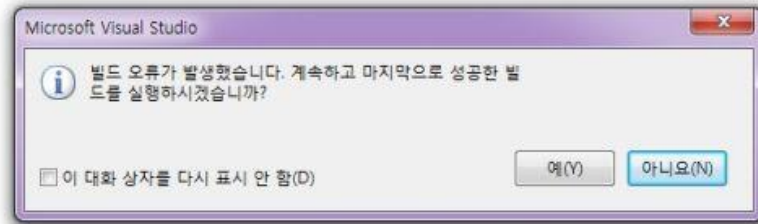
Let's see an example of attempting to create an object for my Static Class.

```

public static class OwnerofficialRoom
{
    public static void AllMyPersonalItems()
    {
        Console.WriteLine("All Items in this rooms are personal to me no one else can access or inherit me");
    }
}

class HouseStaticClass
{
    static void Main(string[] args)
    {
        OwnerofficialRoom objStatic = new OwnerofficialRoom();
        Console.ReadLine();
    }
}

```



When we run the program we get the error message as “Cannot create an instance of a static class”. Sorry, I’m using the Korean Version of Visual Studio so the error message is in the Korean language.

How to call the Static Class Method and variable without creating the Object

It's simple. We can just use the “ClassName.Variable or Method Name” for example “OwnerofficialRoom.AllMyPersonalItems();”
See the following example with a Static class:

```

1. public static class OwnerofficialRoom
2. {
3.     public static void AllMyPersonalItems()
4.     {
5.         Console.WriteLine("All Items in this rooms are personal to me no one else ca
n access or inherit me");
6.     }
7.
8. }
9.
10. class HouseStaticClass
11. {
12.     static void Main(string[] args)
13.     {
14.         OwnerofficialRoom.AllMyPersonalItems();
15.         Console.ReadLine();
16.
17.     }

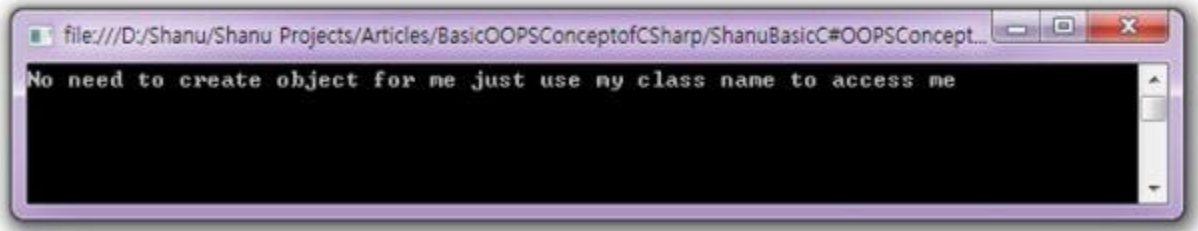
```

```
18. }
```

The output for the preceding program is as in the following:

```
public static class OwnerofficialRoom
{
    public static void AllMyPersonalItems()
    {
        Console.WriteLine("No need to create object for me just use my class name to access me");
    }
}

class HouseStaticClass
{
    static void Main(string[] args)
    {
        OwnerofficialRoom.AllMyPersonalItems();
        Console.ReadLine();
    }
}
```



It is possible to create a Static Method in a non-Static Class

Yes, we can create a Static Method to the non-Static class. No need to create an object to access the static method(s) in a non-static class. We can directly use the class name to access the Static method.

See the following example with Static method in a non-static Class.

```
1. public class OwnerofficialRoom
2. {
3.     public static void AllMyPersonalItems()
4.     {
5.         Console.WriteLine("No need to create object for me just use my class name to
        access me :)");
6.     }
7.
8.     public void non_staticMethod()
9.     {
10.        Console.WriteLine("You need to create an Object to Access Me :)");
11.    }
12.
13.
```

```

14. }
15.
16. class StaticmethodClass
17. {
18.     static void Main(string[] args)
19.     {
20.         // for statci method no need to create object just call directly using the c
           lassname.
21.         OwnerofficialRoom.AllMyPersonalItems();
22.
23.         // for non-static method need to create object to access the method.
24.         OwnerofficialRoom obj = new OwnerofficialRoom();
25.         obj.non_staticMethod();
26.         Console.ReadLine();
27.
28.     }
29. }

```

The output of the preceding program is the following:

```

public class OwnerofficialRoom
{
    public static void AllMyPersonalItems()
    {
        Console.WriteLine("No need to create object for me just use my class name to access me :");
    }

    public void non_staticMethod()
    {
        Console.WriteLine("You need to create an Object to Access Me :(");
    }
}

class StaticmethodClass
{
    static void Main(string[] args)
    {
        // for statci method no need to create object just call directly using the classname.
        OwnerofficialRoom.AllMyPersonalItems();

        // for non-static method need to create object to access the method.
        OwnerofficialRoom obj = new OwnerofficialRoom();
        obj.non_staticMethod();
        Console.ReadLine();
    }
}

```



14. Interface

An interface is same like an abstract class but in an interface we only have a method declaration. In an abstract class however we can have both, a method declaration and a method definition. Methods of an interface must be implemented in an implementing class.

See the following example program for an interface. All the methods of the interface have been implemented in the class. As I have already said, C# doesn't support multiple inheritance. To get multiple inheritance we can use an interface. This following program is an example for multiple inheritance using interface.

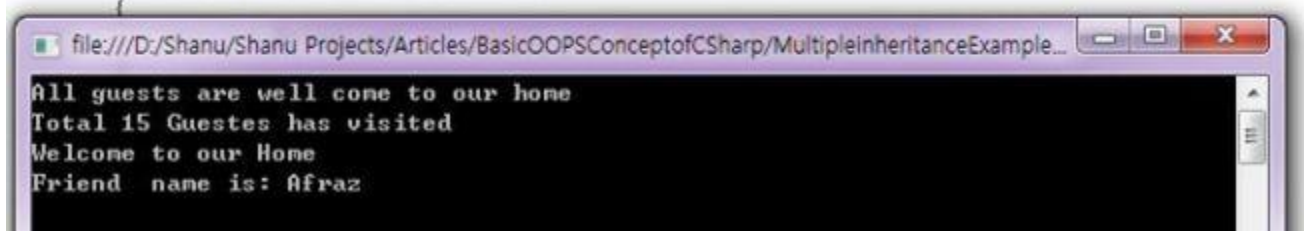
```
1. interface GuestInterface
2. {
3.     void GuestWelcomeMessage();
4.     void NoofGuestes();
5. }
6. interface FriendsandRelationsInterface
7. {
8.     void friendwelcomemessage();
9.     void FriendName();
10. }
11.
12. class HouseOwnerClass : GuestInterface, FriendsandRelationsInterface
13. {
14.     public void GuestWelcomeMessage()
15.     {
16.         Console.WriteLine("All guests are well come to our home");
17.     }
18.
19.     public void NoofGuestes()
20.     {
21.         Console.WriteLine("Total 15 Guestes has visited");
22.     }
23.
24.     public void friendwelcomemessage()
25.     {
26.         Console.WriteLine("Welcome to our Home");
27.     }
28.     public void FriendName()
29.     {
30.         Console.WriteLine("Friend name is: Afraz");
31.     }
32.     static void Main(string[] args)
33.     {
34.         HouseOwnerClass obj = new HouseOwnerClass();
35.         obj.GuestWelcomeMessage();
36.         obj.NoofGuestes();
37.         obj.friendwelcomemessage();
38.         obj.FriendName();
39.         Console.ReadLine();
40.     }
41. }
```

The output will be like this.

```
interface GuestInterface
{
    void GuestWelcomeMessage();
    void NoofGuestes();
}

interface FriendsandRelationsInterface
{
    void friendwelcomemessage();
    void FriendName();
}

class HouseOwnerClass : GuestInterface, FriendsandRelationsInterface
{
    public void GuestWelcomeMessage()
```



```
file:///D:/Shanu/Shanu Projects/Articles/BasicOOPSConceptofCSharp/MultipleinheritanceExample...
All guests are well come to our home
Total 15 Guestes has visited
Welcome to our Home
Friend name is: Afraz
```

In some cases we need to have certain methods that will be used in many derived classes. Each derived can implement different functionality for those Methods. In These cases, we can use the Interface.

We can use our Guest and House example. For the Guest the “Welcome Message” and “No of Guest” Functions are common, but it will be different for different owners in the same house. A Guest might a fathers guest, a Mothers Guest, a Children's Guest or a Family Guest. Each guest can have a different welcome message subject, but the functions are the same as a message. Let's consider now that Father is a class, Mother is a class and Children are one class. Both the guestWelcome Message and Noofguest method are the same for all. In this case, we can create an interface and declare both methods in the interface. All father, mother and children, classes can inherit the interface and write their own method details.

An interface is similar to an Abstract class but the major difference between the Abstract Class and the Interface are that in an Abstract class there can be both an Abstract method and Non-Abstract methods. But in an interface all methods are abstract by default. That means there is not a non-Abstract type method in the interface. All the Methods declared in the interface should be overridden in the derived class.

What will happen when non-abstract methods with body parts are declared in an Interface

It will display the warning message as “**unexpected modifier**” in the Access modifier part and the “**Unexpected Method body**” error warning in the message body. See the following image for the details.

```

interface GuestInterface
{
    public void nonabstractmethodinInterface()
    {
    }
}
class
{
    static void Main(string[] args)
    {
    }
}
  
```

C#: Unexpected method body

Example program for interface:

```

1. interface GuestInterface
2. {
3.     void GuestWelcomeMessage();
4.     void NoofGuestes();
5. }
6. class HouseOwnerClass: GuestInterface
7. {
8.     public void GuestWelcomeMessage()
9.     {
10.         Console.WriteLine("All guests are well come to our home");
11.     }
12.
13.     public void NoofGuestes()
14.     {
15.         Console.WriteLine("Total 15 Guestes has visited");
16.     }
17.
18.     static void Main(string[] args)
19.     {
20.         HouseOwnerClass obj = new HouseOwnerClass();
21.
22.         obj.GuestWelcomeMessage();
23.         obj.NoofGuestes();
24.         Console.ReadLine();
25.     }
26. }
  
```