**OOPs Concepts**

### Abstraction and Encapsulation

**https://www.codeproject.com/Articles/1037139/Difference-between-Encapsulation-and-Abstraction-i**

## Abstraction

Abstraction is a process to abstract or hide the functionality and provide users or other programmers to use it only. Like for the method Console.WriteLine(), no one knows what actually is happening behind the function calling. We are just using it by calling and passing the arguments. This is the thing called Abstraction.

## Encapsulation

Encapsulation means to encapsulate or put everything into one thing and provide others to use it. Like in a shaving kit, all the necessary kits are available. And also these kits are available as loose in market. But the shaving kit encapsulates every other kit into a small bag and provides a user to use it.

Hope you now have a basic idea about both of these properties. Let's see a real world example of encapsulation and abstraction.

Let's assume you have to create a method to insert users data and pass it to other developers to use. So first, create a class and add a method to insert the data into database with validation.

There will be three fields:

1. Name
2. Email
3. Phone number

So these inputs have to validate first and then insert into db.

First, create a class with all methods:

Hide Copy Code

```
class User
{
   public bool AddUser(string name, string email, string phone)
   {
      if (ValidateUser(name, email, phone))
      {
         if (AddtoDb(name, email, phone) > 0)
         {
            return true;
         }
```

```
        }
        return false;
    }

    private bool ValidateUser(string name, string email, string phone)
    {
        // do your validation
        return true;
    }

    private int AddtoDb(string name, string email, string phone)
    {
        // Write the Db code to insert the data
        return 1;
    }
}
```

As you can see, there are three methods that are written in this User class.

- AddUser: To call from outside the class. That is why the access modifier is public.
- validateUser: To validate the user's details. Can't access from outside the class. It's private.
- AddtoDb: To insert data into database table and again it is private, can't access from outside the class.

Now another user will just call AddUser method with parameters. And that user has no idea what is actually happening inside the method. I didn't write the code to validate and insert into db, as you can get it from others examples. We will discuss about it later.

To call the AddUser method, do as follows:

Hide Copy Code

```
class Program
{
    static void Main(string[] args)
    {
        User objUser = new User();
        bool f = objUser.AddUser("Arka", "ark@g.com", "1234567890");
    }
}
```

Now come back to the main discussion.

Here, we are hiding the procedure of adding data into database from other users, this is Abstraction. And putting all the three methods into one User class and providing other users to use it, that is called Encapsulation.

So procedure hiding is Abstraction and putting every necessary thing into one is Encapsulation.

"Encapsulation is accomplished by using Class. - Keeping data and methods that accesses that data into a single unit" "Abstraction is accomplished by using Interface. - Just giving the abstract information about what it can do without specifying the back ground details"

**Abstraction**

Abstraction allows us to represent complex real world in simplest manner. It is process of identifying the relevant qualities and behaviors an object should possess, in other word represent the necessary feature without representing the back ground details. Abstraction is a process of hiding work style of an object and showing only those information which are required to understand the object. Abstraction means putting all the variables and methods in a class which are necessary.

good examples are classes provided by the .net framework, for example list or collection. these are very abstract classes that you can use almost everywhere and in a lot of domains. Imagine if .net only implemented a EmployeeList class and a CompanyList that could only hold a list of employees and companies with specific properties. such classes would be useless in a lot of cases. and what a pain would it be if you had to re-implement the whole functionality for a CarList for example. So the "List" is ABSTRACTED away from Employee, Company and Car. The List by itself is an abstract concept that can be implemented by its own class.

Interfaces, abstract classes or inheritance and polymorphism are tools to provide abstraction in c#.

you do abstraction in order to provide reusability.

**Encapsulation**

It is a process of hiding all the internal details of an object from the outside real world. The word Encapsulation, like Enclosing into the capsule. It restrict client from seeing its internal view where behavior of the abstraction is implemented. In Encapsulation, generally to hide data making it private and expose public property to access those data from outer world. Encapsulation is a method for protecting data from unwanted access or alteration. Encapsulation is the mechanism by which Abstraction is implemented.

**Difference between Abstraction and Encapsulation**

Abstraction is a process. It is the act of identifying the relevant qualities and behaviors an object should possess. Encapsulation is the mechanism by which the abstraction is implemented.

| Abstraction | Encapsulation |
|---|---|
| Abstraction solves the problem in the design level. | Encapsulation solves the problem in the implementation level. |
| Abstraction is used for hiding the unwanted data and giving only relevant data. | Encapsulation is hiding the code and data into a single unit to protect the data from outer world. |
| Abstraction is set focus on the object | Encapsulation means hiding the internal details or mechanics of |

| | |
|---|---|
| instead of how it does it. | how an object does something. |
| Abstraction is outer layout in terms of design.<br>For Example: - Outer Look of a iPhone, like it has a display screen. | Encapsulation is inner layout in terms of implementation.<br>For Example: - Inner Implementation detail of a iPhone, how Display Screen are connect with each other using circuits |

**What is Polymorphism?**
In programming languages and type theory, polymorphism is the provision of a single interface to entities of different types.

A polymorphic type is a type whose operations can also be applied to values of some other type, or types.

**What is Inheritance?**
inheritance is when an object or class is based on another object or class, using the same implementation (inheriting from a class) specifying implementation to maintain the same behavior (realizing an interface; inheriting behavior).

It is a mechanism for code reuse and to allow independent extensions of the original software via public classes and interfaces.

**What is Constructor?**
A is special method of the class that will be automatically invoked when an instance of the class is created is called as constructor.

Constructors are mainly used to initialize private fields of the class while creating an instance for the class.

When you are not creating a constructor in the class, then compiler will automatically create a default constructor in the class that initializes all numeric fields in the class to zero and all string and object fields to null.

Syntax.
[Access Modifier] ClassName([Parameters])
{
}

**Types of Constructors**
Basically constructors are 5 types those are
Default Constructor
Parameterized Constructor
Copy Constructor
Static Constructor
Private Constructor

**Define Destructor?**
A destructor is a method which is automatically invoked when the object is destroyed.

Its main purpose is to free the resources (memory allocations, open files or sockets, database connections, resource locks, etc.)

**What is Method Overriding? How to override a function in C#?**

Use the override modifier to modify a method, a property, an indexer, or an event. An override method provides a new implementation of a member inherited from a base class. The method overridden by an override declaration is known as the overridden base method. The overridden base method must have the same signature as the override method.

**Difference between new and override keyword?**

```
using System;

using System.Data;

using System.Text;

using System.Windows.Forms;


namespace BaseDerive

{

   public partial class Form1 : Form

   {

      public Form1()

      {

         InitializeComponent();

      }


      private void Form1_Load(object sender, EventArgs e)

      {


         BaseClass b = new BaseClass();

         b.func1();
```

```
        DeriveClass d = new DeriveClass();
        d.func1();

        //Calls Base class function 1 as new keyword is used.
        BaseClass bd = new DeriveClass();
        bd.func1();

        //Calls Derived class function 2 as override keyword is used.
        BaseClass bd2 = new DeriveClass();
        bd2.func2();

    }
}

public class BaseClass
{
    public virtual void func1()
    {
        MessageBox.Show("Base Class function 1.");
    }

    public virtual void func2()
    {
        MessageBox.Show("Base Class function 2.");
    }
```

```
  public void func3()
      {
         MessageBox.Show("Base Class function 3.");
      }
   }

   public class DeriveClass : BaseClass
   {
      public new void func1()
      {
         MessageBox.Show("Derieve Class fuction 1 used new keyword");
      }

      public override void func2()
      {
         MessageBox.Show("Derieve Class fuction 2 used override keyword");
      }

public void func3()
      {
         MessageBox.Show("Derieve Class fuction 3 used override keyword");
      }

   }
}
```

BaseClass objB = new DeriveClass();

If we create object like above notation and make a call to any function which exists in base class and derive class both, then it will always make a call to function of base class. If we have overidden the method in derive class then it wlll call the derive class function.

**What is a private constructor? Where will you use it?**

When you declare a Constructor with Private access modifier then it is called Private Constructor. We can use the private constructor in singleton pattern.

If you declare a Constructor as private then it doesn’t allow to create object for its derived class, i.e you loose inherent facility for that class.

Example:

<pre class="prettyprint">Class A

{

// some code

Private Void A()

{

//Private Constructor

}

}

Class B:A

{

//code

}

B obj = new B();// will give Compilation Error</pre>

Because Class A constructor declared as private hence its accessibility limit is to that class only, Class B can't access. When we create an object for Class B that constructor will call constructor A but class B have no rights to access the Class A constructor hence we will get compilation error.

**What is Polymorphism?**

Parent classes may define and implement “virtual” methods(Which is done using the “virtual” keyword), and derived classes can override them(using the “override” keyword), which means they provide their own definition and implementation.At run-time, when user’s code calls the method, the CLR looks up the run-time type of the object, and invokes that override of the virtual method. Thus in your source code when a method of the base class is called it executes the overriden method.

**What's the Difference between Interface and Abstract Class**

Abstract Class:

Have constructors.

Not necessarily for the class inheriting it to Implement all the Methods.

Doesn't Support Multiple Inheritance.

Where everything is Opposite in the Interfaces.

**When to Use Abstract Classes and When Interfaces.**

If you anticipate creating multiple versions of your component, create an abstract class. Abstract classes provide a simple and easy way to version your components. By updating the base class, all inheriting classes are automatically updated with the change. Interfaces, on the other hand, cannot be

changed once created. If a new version of an interface is required, you must create a whole new interface.

If the functionality you are creating will be useful across a wide range of disparate objects, use an interface. Abstract classes should be used primarily for objects that are closely related, whereas interfaces are best suited for providing common functionality to unrelated classes.

If you are designing small, concise bits of functionality, use interfaces. If you are designing large functional units, use an abstract class.

If you want to provide common, implemented functionality among all implementations of your component, use an abstract class. Abstract classes allow you to partially implement your class, whereas interfaces contain no implementation for any members.

**Diversities between an abstract method & virtual method ?**

An Abstract method does not provide an implementation and forces overriding to the deriving class (unless the deriving class also an abstract class), where as the virtual method has an implementation and leaves an option to override it in the deriving class. Thus Virtual method has an implementation & provides the derived class with the option of overriding it. Abstract method does not provide an implementation & forces the derived class to override the method.

**What is Early binding and late binding?**

Calling a non-virtual method, decided at a compile time is known as early binding. Calling a virtual method (Pure Polymorphism), decided at a runtime is known as late binding.

**Difference between sealed and static classes**

sealed classes:

1)we can create their instances, but cannot inherit them

ex:

sealed class demo

{

}

class abc:demo

{

--Wrong

}

2)They can contain static as well as nonstatic members.

static classes:

1)we can neither create their instances, nor  inherit them

ex:

static class Program

{

}

2)They can have static members only.

**Can we have an Abstract class without having any abstract method ??**

Yes we can have Abstract class without having any abstract method ..

See the code below

```
using System;

abstract class A
{
    public void Hello()
    {
        Console.WriteLine(" Say Hi");
    }
}

class B:A
{
}

class Demo
{
    public static void Main()
    {
        B b1 = new B();
        b1.Hello();
    }
}
// Output is Say HI
```

the class A is abstract class.. but it does not have any abstract methods..

**We can do it in either of the following two ways:**

*a) Using a built-in function as in the following:*

```
1
2   string strRev,strReal = null;
3   Console.WriteLine("Enter the string..");
4   strReal = Console.ReadLine();
5   char[] tmpChar = strReal.ToCharArray();
6   Array.Reverse(tmpChar);
7   strRev=new string(tmpChar);
8   if(strReal.Equals(strRev, StringComparison.OrdinalIgnoreCase))
9   {
10      Console.WriteLine("The string is pallindrome");
11  }
12  else
13  {
14      Console.WriteLine("The string is not pallindrome");
15  }
    Console.ReadLine();
```

Ref: To check string is palindrome or not in .NET (C#)

*b) Without using a built-in function.*

When I wrote the first program, the interviewer asked me to write the same without using a built-in function.

```
1   private static bool chkPallindrome(string strVal)
2           {
```

```
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
```

```
            try
            {
                int min = 0;
                int max = strVal.Length - 1;
                while (true)
                {
                    if (min > max)
                        return true;
                    char minChar = strVal[min];
                    char maxChar = strVal[max];
                    if (char.ToLower(minChar) != char.ToLower(maxChar))
                    {
                        return false;
                    }
                    min++;
                    max--;
                }
            }
            catch (Exception)
            {

                throw;
            }
        }
```

Ref: You can find more here: C# Palindrome

*7. Write a program to determine the count of a specific character in a string.*

**A.**

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


namespace FindCountCharOccurance
{
    class Program
    {
        static void Main(string[] args)
        {
            string strOccur,strChar = null;
            Console.WriteLine("Enter the string in which you need to find the count o
occurance");
            strOccur = Console.ReadLine();


            Console.WriteLine("Enter the char to be searched..");
            strChar = Console.ReadLine();
            int intCnt =strOccur.Length- strOccur.Replace(strChar, string.Empty).Leng
            Console.WriteLine("Count of occurance is "+intCnt);
            Console.ReadLine();
        }
    }
}
```

Please see this for more suggestions: count the number of characters in a string.

**8. Next he gave me a program like the following and asked me what the output of this will be.**

```
public class A
    {
        public int A()
        {
            Console.WriteLine("Hi you are in class A");
        }
    }
```

A. I said “Here we have a constructor A; a constructor should not have a return type. So the code above will throw a compilation error.”

*9. What may be the output of the following program?*

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


namespace RefClass
{
    class Program
    {
        static void Main(string[] args)
        {
            B bObj= new B();
            Console.ReadLine();


```

```
        }
    }
    public class A
    {
        public A()
        {
            Console.WriteLine("Hi you are in class A");
        }
    }

    public class B:A
    {
        public B()
        {
            Console.WriteLine("Hi you are in class B");
        }
    }
}
```

A. I said the output will be:

Hi you are in class A

Hi you are in class B

Even though you are creating an object of the derived class, it will invoke the base class first.

*10. Write the output of the following program.*

```
class Program
    {
```

```
3           static void Main(string[] args)
4           {
5               B bObj= new B(2);
6               Console.ReadLine();
7
8           }
9       }
10      public class A
11      {
12          public  A()
13          {
14              Console.WriteLine("Hi you are in class A");
15          }
16
17          public A(int x)
18          {
19
20          }
21      }
22
23      public class B:A
24      {
25          public B()
26          {
27              Console.WriteLine("Hi you are in class B");
28          }
```

| | |
|---|---|
| 29 | |