Abstract Class

There are situations in which you will want to define a superclass that declares the structure of a given abstraction without providing a complete implementation of every method. That is, sometimes you will want to create a superclass that only defines a generalized form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details. Such a class determines the nature of the methods that the subclasses must implement. One way this situation can occur is when a superclass is unable to create a meaningful implementation for a method.

You can require that certain methods be overridden by subclasses by specifying the abstract type modifier. These methods are sometimes referred to as subclasser responsibility because they have no implementation specified in the superclass. Thus, a subclass must override them—it cannot simply use the version defined in the superclass. To declare an abstract method, use this general form:

Syntax: `abstract return_type MethodName(parameter-list);`

As you can see, no method body is present. **Any class that contains one or more abstract methods must also be declared abstract. To declare a class abstract, you simply use the abstract keyword in front of the class keyword at the beginning of the class declaration. There can be no objects of an abstract class.** That is, an abstract class cannot be directly instantiated with the new operator. Such objects would be useless, because an abstract class is not fully defined. Also, you cannot declare abstract constructors, or abstract static methods. Any subclass of an abstract class must either implement all of the abstract methods in the superclass, or be itself declared abstract.

Note `Abstract` **class cannot be instantiate - cannot create object**

```
abstract class A
{
      abstract public int arith(int a, int b);
}

class B extends A
{
      public int arith (int a, int b)
      {
      return(a-b);
      }
}
class C extends B
{
      public int arith(int a,int b)
      {
      return(a*b);
      }
}
class D extends C
{
      public int arith(int a,int b)
      {
      return(a/b);
      }
}
class E extends D
{
      public int arith(int a,int b)
      {
      return(a+b);
      }
}
class AbstractDemo
{
 public static void main(String args[])
 {
 A p1;  //reference variable for base class
 p1=new B();
 System.out.println("Subtraction of two number is "+p1.arith(20,10));
 p1=new C();
 System.out.println("Product of two nos. is "+p1.arith(20,10));
 p1=new D();
 System.out.println("Division of two nos is "+p1.arith(20,10));
 p1= new E();
 System.out.println("sum of two nos is "+p1.arith(20,10));
 }
}
```

OUTPUT

```
Addition of two number is 30
```

```
Subtraction of two number is 10
Product of two number is 200
Division of two number is 2
```

9. Interfaces

In Java, the multiple inheritance problem is solved with a powerful construct called **interfaces**. Interface can be used to define a generic template and then one or more a classes to define implementations of the interface.

Interfaces just specify the method declaration (implicitly public and abstract) and can only contain fields (which are implicitly public static final). Interface definition begins with a keyword interface. An interface like that of an abstract class cannot be instantiated.

Java does not support multiple inheritance, but it allows you to extend one class and implement many interfaces.

If a class that implements an interface should define all the methods of the interface.

9.1 Defining an interface

```
public interface <interface-name>
{
        final datamembers;
        +
        abstract methods();
}
```

Note :

- methods are public and abstract
- fields are public and final .

9.2 Implementing interfaces

Once an interface has been defined, one or more classes can implements that interface. To implements the interface, include the implements keyword in a class definition and then create a method defined by the interface.

```
class sub-class-name implements interface-name
{

}
```

Declaration an object

Syntax `interface-name reference = new sub-class-name();`

Program 1

```
//class implementing an interface

interface A
{
        abstract  public void arith(int a, int b);
}
class B implements A
{
        public void arith(int a, int b)
```

```
        {
                int c=a+b;
                System.out.println("Sum of two numbers is "+c);
        }
}
class InterfaceDemo1
{
        public static void main(String args[])
        {
        A obj = new B();
        obj.arith(20,10);
        }
}
```

Output

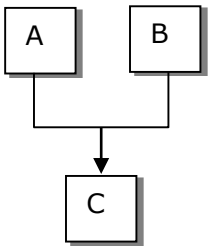
```
Sum of two numbers is 30
```

Program 2

```
                        interface                       Shape
                                                        {
                public                  String      baseclass="shape";

                    abstract        public      void        draw();

 }

        class           Circle          implements          Shape
 {

      public void draw()
      {
                        System.out.println("Drawing   Circle   here");
      }

 }
public class Main
{
      public static void main(String[] args)
      {
                                Shape   circleShape=new   Circle();
                                        circleShape.draw();
      }
}
```

9.3 Multiple Inheritance

It is a process of creating a sub class from more than one super class is called as Multiple Inheritance. In Java a sub-class cannot inherit more than one super-class, so achiving multiple inheritance is not possible through classes.



Multiple inheritance through Interfaces

In java a class implements more than one interfaces, the interfaces are separated by comma. In this multiple inheritance can achieved.

Program

```
      interface A
      {
            abstract public void sum(int  a, int b);
      }
```

```
	interface B
	{
		abstract public void mul(int a,int b);
		abstract public void msg();
	}
	interface C
	{
		abstract public void diff(int a,int b);
	}

	class D implements A,B,C
	{
		public void sum(int a,int b)
		{
			int c=a+b;
			System.out.println("Sum of two numbers is "+c);
		}
		public void mul(int a, int b)
		{
			int c=a*b;
			System.out.println("Product of two numbers is "+c);
		}
		public void diff(int a,int b)
		{
			int c=a-b;
			System.out.println("Subtraction of two numbers is "+c);
		}
	}
class InterfaceDemo2
{
	public static void main (String args[])
	{
	D obj = new D( );

	obj.sum(20,10);
	obj.diff(20,10);
	obj.mul(20,10);

	}
}
```

Output

```
Addition of two number is 30
Subtraction of two number is 10
Product of two number is 200
```

9.4 Extending an Interface

Like class, an interface can inherit another interface by using keyword extends.

Syntax

```
	interface B extends A
```

Program

```
	interface A
	{
		abstract  public void sum(int a,int b);
	}
	interface B extends A
```

```
	{
		abstract  public void mul(int a,int b);
	}
	interface C extends B
	{
		abstract  public void diff (int a , int b);
	}
	class D implements C
	{
		public void sum(int a,int b)
		{
			int c=a+b;
			System.out.println("Sum of two numbers is "+c);
		}
		public void diff(int a,int b)
		{
			int c=a-b;
			System.out.println("Subtraction of two numbers is "+c);
		}

		public void mul(int a,int b)
		{
			int c=a*b;
			System.out.println("Product of two numbers is "+c);
		}
	}
class InterfaceDemo3
{
	public static void main(String args[])
	{
	D obj = new D( );
	obj.sum(20,10));
	obj.diff(20,10));
	obj.mul(20,10));

	}
}
```

Output

```
Addition of two number is 30
Subtraction of two number is 10
Product of two number is 200
```

Class vs Interface

1. A class extends only one class
2. An Interface extends only one Interface
3. But, A class implements more than one Interfaces

Note No way an interface can extends or implements a class

9.5 Constants in Interfaces

We can use interface to import shared constants into multiple constant by simple declaring an interface that contains variables which are initialize to the required value.

```
interface A
{
	public final int a=10;
```

```
	public final int b=20;
	abstract public void sum(int a, int b);
}
class B implements A
{
	public void sum(int a, int b)
	{
			int c=a+b;
			System.out.println("Sum of two numbers is "+c);


	}
}
class InterfaceDemo4
{
	public static void main(String args[])
	{
		A obj=new B();
		obj.sum(obj.a,obj.b);
	}
}
```

Output `Sum of two numbers is 30`

Abstract class vs Interface

What is an Abstract Class?

An abstract class is a special kind of class that cannot be instantiated. So the question is why we need a class that cannot be instantiated? An abstract class is only to be sub-classed (inherited from). In other words, it only allows other classes to inherit from it but cannot be instantiated. The advantage is that it enforces certain hierarchies for all the subclasses. In simple words, it is a kind of contract that forces all the subclasses to carry on the same hierarchies or standards.

What is an Interface?

An interface is not a class. It is an entity that is defined by the word Interface. An interface has no method implementation; it only has the signature or in other words, just the definition of the methods without the body. As one of the similarities to Abstract class, it is a contract that is used to define hierarchies for all subclasses or it defines specific set of methods and their arguments. The main difference between them is that a class can implement more than one interface but can only inherit from one abstract class. Since java doesn't support multiple inheritance, interfaces are used to implement multiple inheritance.

Both Together

When we create an interface, we are basically creating a set of methods without any implementation that must be overridden by the implemented classes. The advantage is that it provides a way for a class to be a part of two classes: one from inheritance hierarchy and one from the interface.

When we create an abstract class, we are creating a base class that might have one or more completed methods but at least one or more methods are left uncompleted and declared abstract. If all the methods of an abstract class are uncompleted then it is same as an interface. The purpose of an abstract class is to provide a base class definition for

how a set of derived classes will work and then allow the programmers to fill the implementation in the derived classes.

There are some similarities and differences between an interface and an abstract class that I have arranged in a table for easier comparison:

Feature	Interface	Abstract class
Abstract	All are implicitly abstract methods and cannot have implementations.	Can have instance methods that implements a default behavior.
Constructor	Does not have	has the constructor
Variables	by default final	non-final variables
Access specifier	public by default	can have the usual flavors of class members like private, protected, etc..
Multiple inheritance	Achieve, because class may inherit several interfaces.	Cannot Achieve, because class may inherit only one abstract class.
Extends	An interface can extend another Java interface only	an abstract class can extend another Java class and implement multiple Java interfaces
Instatitation (object creation)	Interface is absolutely abstract and cannot be instantiated	abstract class also cannot be instantiated, but can be invoked if a main() exists.
Core VS Peripheral	Interfaces are used to define the peripheral abilities of a class. In other words both Human and Vehicle can inherit from a IMovable interface.	An abstract class defines the core identity of a class and there it is used for objects of the same type.
Homogeneity	If various implementations only share method signatures then it is better to use Interfaces.	If various implementations are of the same kind and use common behaviour or status then abstract class is better to use.
Speed	Requires more time to find the actual method in the corresponding classes.	Fast
Adding functionality (Versioning)	If we add a new method to an Interface then we	If we add a new method to an abstract class then

Feature	Interface	Abstract class
	have to track down all the implementations of the interface and define implementation for the new method.	we have the option of providing default implementation and therefore all the existing code might work properly.

9.6 Final Modifier

Final Modifier can be apply **to variables, methods and classes**

9.6.1 Final variable

If we make a variable as final, the variable will become a constant. In this case we can't change the final value.

```
class FinalVariable
{
	public static void main(String args[])
	{
		final int a=10;
		a=a+5;//error
		System.out.println(a);
	}
}
```

Error Message `Cannot assign a value to final variable a`

9.6.2 Final Class

To stop further inheritance, make that class as final, we can't create a sub class from final class.

```
final class A
{
	public int arith(int a,int b)
	{
	return(a+b);
	}

}

class B extends A
{
	public int arith(int  a, int b)
	{
	return(a-b);
	}
}
```

Error Message `Cannot inherit from final class A`

9.6.3 Final Method

Final method is used to put a lock on the method to prevent any inheriting class from changing its meaning. This is done for design reasons when you want to make sure that a method's behavior is retained during inheritance and cannot be overwritten.

```
class A
{
      public int arith(int a,int b)
      {
            return(a+b);
      }
}
class B extends A
{
      final public int arith(int a, int b)
      {
            return(a-b);
      }
}
class C extends B
{
      public int arith(int a, int b)
      {
            return(a*b);
      }
}
class D extends C
      {
      final public int arith(int a, int b)
      {
            return(a/b);
      }
}
class FinalMethod
{
      public static void main(String args[])
      {
            A p;
            p=new B();
            System.out.println(p.arith(10,20));
            p=new C();
            System.out.println(p.arith(10,20));
            p=new D();
            System.out.println(p.arith(10,20));
      }
}
```

Error Message

```
arith(int,int) in class C cannot override arith(int,int)
in class B overridden method is final
```