Chapter - 7

Interfaces

Interfaces

Interfaces are the extension of **abstract classes**. An **interface** is a pure abstract class which contains only abstract methods and final variables. Interface is framework of an object using which new classes can be implemented. To the interfaces objects cannot be instantiated because they do not perform operations.

Syntax:

```
access interface interfacename
{
        [public] type final_variable1=value1, . . .;
                :
        [public] returntype methodname1([type para1, . . .]);
                :
}
```

If the access specifier of interface is not mentioned then the default access specifier becomes default and the interface is available to any class within the same package in which interface exist. To make the interface available outside the package any where then the access specifier of the interface should be public.

The interface and its methods are by default abstract therefore we should not declare them with the keyword **abstact**. The members (variables and methods) of interface are by default public and if we want to specify the access specifier explicitly it should be public otherwise it is error. All variables defined in the interface are by default static and final variables therefore they can be accessed using the interface name.

To the interfaces, objects cannot be instantiated because interfaces are half developed. Interfaces are used for implementing into subclasses and it is the responsibility of subclass that it should override all abstract methods of interfaces otherwise the subclass also becomes an abstract class. Objects can be instantiated to these subclasses to make us of them.

Syntax:

```
class classname [ extends superclass ]  implements interface1[,interface2, . .]
{
        // code to implement the abstract methods of interfaces...
}
```

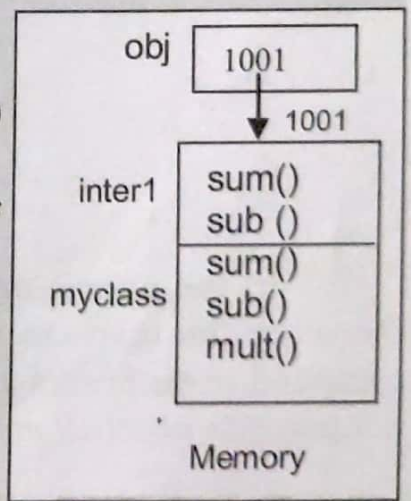
If the subclass is inheriting from superclass and implements from interfaces then the subclass should inherit from superclass first and then it should implement from interfaces. A subclass can inherit from only one supercalss where as it can implement from any number of interfaces therefore multiple inheritance can be achieved in java using interfaces.

A program to demonstrate the interface and its implementation.

Bin>edit in1.java

```
interface inter1
{
      void sum(int x,int y);  // by default public
      void sub(int x,int y);  // by default public
}

class myclass implements inter1
{
      public void sum(int x,int y)
      {
            System.out.print("\n Sum="+(x+y));
      }
      public void sub(int x,int y)
      {
            System.out.print("\n Sub="+(x-y));
      }
      public void mult(int x,int y)
      {
            System.out.print("\n Mult="+(x*y));
      }
}
class in1
{
   public static void main(String s[])
   {
            myclass obj=new myclass();
            obj.sum(10,20);
            obj.sub(10,2);
            obj.mult(6,3);
   }
}
```



```
Bin> javac in1.java
Bin> java in1
        Sum=30
        Sub=12
        Mult=18
```

Explanation:

In the above program, **myclass** is implementing from interface **inter1** therefore **myclass** is overriding all abstract methods(**sum()** and **sub()**) of interface and it also contains its own method **mult()**.

In the main(), **obj** is instantiated to **myclass** as shown in memory and invokes the methods **sum()**, **sub()** and **mult()**. The methods of subclass are executed because subclass is given first preference.

Partial implementation of Interface

If the subclass is not overriding all the abstract methods of interface then the class also becomes an abstract class therefore the class should be declared with the keyword **abstract**. To this abstract class we can't instantiate objects.

For example re-execute the above program by modifying the myclass as shown below.

```
class myclass implements inter1
{
        public void sum(int x,int y)
        {
                System.out.print("\n Sum="+(x+y));
        }
        public void mult(int x,int y)
        {
                System.out.print("\n Mult="+(x*y));
        }
}
```

In the above, myclass is not overriding the **sub()** method of interface therefore the myclass becomes an abstract class and when the program is complied it generates an error saying that ***myclass*** *is not abstract and does not override abstract method* ***sub()*** *in* ***inter1***.

The myclass should declare with the keyword abstract.

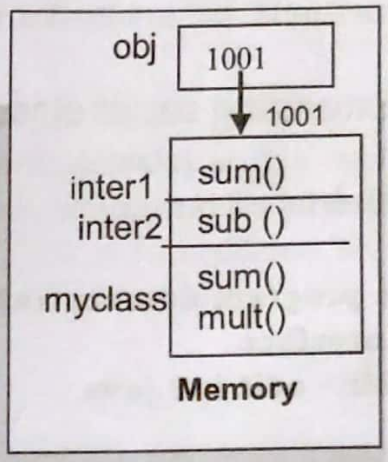
```
abstract myclass implements inter1
{
        :
        :
}
```

If the **myclass** is declared with the keyword **abstract** then the compiler does not generate an error, but to the myclass objects cannot be instantiated. Any class that inherits myclass should implement the abstract method **sub()** otherwise it should be declared with the keyword **abstract**.

Multiple inheritance through interfaces.

In java multiple inheritance cannot be implemented using classes where as it can be achieved using interfaces. A class can implement from any number of interfaces which is said to be multiple inheritance.

The following program demonstrates the **multiple inheritance** using interfaces.



Bin> edit in2.java

```
interface inter1
{
      public void sum(int x,int y);
}
interface inter2
{
      public void mult(int x,int y);
}
class myclass implements inter1,inter2
{
      public void sum(int x, int y)
      {
            System.out.print("\n sum="+(x+y));
      }
      public void mult(int x,int y)
      {
            System.out.print("\n mult="+(x*y));
      }
}
```

```
class in2
{
        public static void main(String argv[])
        {
                myclass obj=new myclass();
                obj.sum(10,20);
                obj.mult(6,2);
        }
}
```

Bin>javac in2.java
Bin>java in2

sum=30
mult=12

Explanation:

In the above program the **myclass** is implementing two interfaces **inter1** and **inter2** therefore the **myclass** is overriding the abstract methods of interfaces. In the main() **obj** is instantiated to **myclass** and using the **obj** methods are invoked and we get the output as shown above.

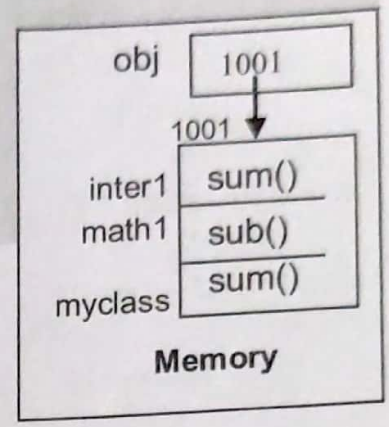
Extending super class and implementing interface

It is possible that a subclass can extend super class and implement interfaces but super class should extend first and then interfaces.

A program demonstrating extending super class and implementing interface.
Bin> edit in3.java

```
interface inter1
{
        public void sum(int x,int y);
}
class math1
{
        public void sub(int x,int y)
        {
                System.out.print("\n sub="+(x-y));
        }
}
```

```
class myclass extends math1 implements inter1
{
        public void sum(int x, int y)
        {
                System.out.print("\n sum="+(x+y));
        }
}
class in3
{
        public static void main(String argv[])
        {
                myclass obj=new myclass();
                obj.sum(10,20);
                obj.sub(20,10);
        }
}
```



Bin>javac in3.java
Bin> java in3

```
sum=30
sub=10
```

Explanation:

In the above program **myclass** is extending **math1** class and implementing **inter1** interface therefore **myclass** is overriding the abstract method **sum()** of **inter1**. In the main(), **obj** is instantiated to **myclass** and using the **obj** the methods are invoked.

Extending interfaces

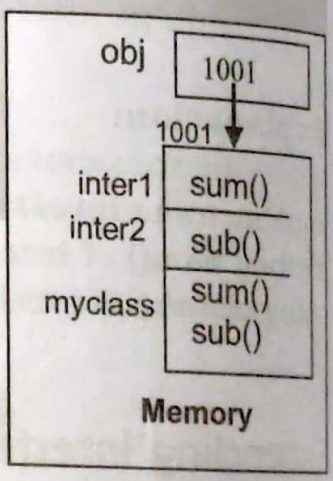
Similar to classes an interface can extend another interface but only single inheritance is allowed. The inherited interface contains all abstract methods of its super. Any class that implements inherited interface should implement all abstract methods of sub and super interfaces.

A program demonstrating extending interfaces.

Bin>edit in4.java

```
interface inter1
{
        public void sum(int x,int y);
}
```

```
interface inter2 extends inter1
{
        public void sub(int x,int y);
}
class myclass implements inter2
{
        public void sum(int x,int y)
        {
                System.out.print("\n sum="+(x+y));
        }
        public void sub(int x,int y)
        {
                System.out.print("\n sub="+(x-y));
        }
}
class in4
{
        public static void main(String argv[])
        {
                myclass obj=new myclass();
                obj.sum(10,20);
                obj.sub(5,2);
        }
}
```



Bin> javac in4.java
Bin> java in4

sum=30
sub=3

Explanation:

In the above program, **myclass** is implementing the **inter2** interface which is extending **inter1** therefore **myclass** is overriding the methods of both **inter1** and **inter2**. In the main(), **obj** is instantiated to **myclass** and invokes the methods.

Variables in interfaces

Variables created in interfaces are by default **public static** and **final** therefore these variables should be initialized, and cannot be modified any where. These variables can be accessed without using the object but can be accessed using the interface name or implementing classname.

Example:

Bin>edit in5.java

```
interface inter1
{
      int A=100;

}
class myclass implements inter1
{
      public void show()
      {
            System.out.print("\n A="+A);
            // A=200; // error: can't modify final
                      // variables values
      }

}
class in5
{
      public static void main(String argv[])
      {
            myclass obj=new myclass();
            obj.show();
            System.out.print("\n inter1.A="+inter1.A);
            System.out.print("\nmyclass.A="+myclass.A);
      }
}
```

Bin>javac in5.java

Bin>java in5

```
A=100
inter1.A=100
myclass.A=100
```

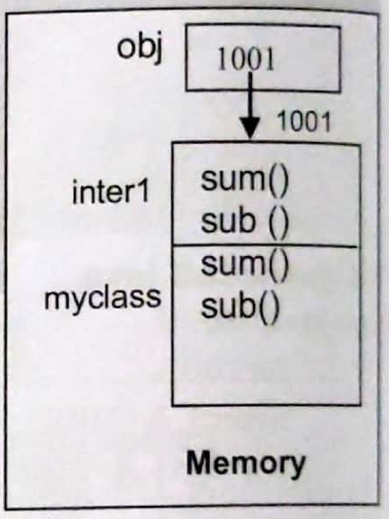
interface reference variable referencing its subclass object

The interface reference variable can store the address of its implemented(sub) class object, this is valid assignment because a super class reference variable can store the reference of subclass. Using the interface reference variable, only the interface members can be accessed but not its subclass.

Example:

Bin>edit in6.java

```
interface inter1
{
        void sum(int x,int y);
        void sub(int x,int y);
}
class myclass implements inter1
{
        public void sum(int x,int y)
        {
                System.out.print("\n Sum="+(x+y));
        }
        public void sub(int x,int y)
        {
                System.out.print("\n Sub="+(x-y));
        }
}
class in6
{
        public static void main(String s[])
        {
                inter1 obj=new myclass();
                obj.sum(10,20);
                obj.sub(10,2);
        }
}
```



Bin> javac in6.java

Bin> java in6

Sum=30

Sub=8

Nested Interfaces or Member Interfaces

Interfaces can be nested in another interface or class. An interface can be declared as a member of a class or other interface. Such an interface is known as nested interface. The nested interface can be declared as private or public or protected. Where as top-level interface is one which should be declared as public or default. The nested interface should be accessed out side the outer class or interface using the outer class name or interface name.

Syntax:

```
class classname
{
        access interface nestedinterfacename
        {
                :
        }
}
```

```
interface outerinterfacename
{
        :
        access interface nestedinterfacename
        {
                :
        }
}
```

A program to demonstrate nested interface.
Bin>edit nestedinter1.java

```
class A
{
    public interface inter1
    {
          public void sum(int x,int y);
    }
}
class B implements A.inter1
{
    public void sum(int x,int y)
    {
          System.out.print("\nsum="+(x+y));
    }
}
class nestedinter1
{
    public static void main(String argv[])
    {
          B obj=new B();
          obj.sum(10,20);
```

```
        A.inter1 i=obj;  // valid statement because
        // super class reference variable can reference
        // sub class object
        obj.sum(5,8);
    }
}
```

Bin>javac nestedinter1.java
Bin>java nestedinter1

sum=30
sum=13

Note: Re-execute the above program replacing **class A** with **interface A** which becomes an example of interface in another interface (nested interface).

Differences between abstract classes and interfaces

Abstract class	Interface
1. Abstract classes are defined using class keyword.	1. Interfaces are defined using interface keyword.
2. Abstract classes should be declared with abstract keyword.	2. Interfaces are by default abstract class and need not to declare with the keyword abstract.
3. Abstract class should inherit into subclass using extends keyword.	3. Interface should inherit into subclass using implements keyword.
4. Abstract methods in the abstract class should be declared with the keyword abstract.	4. Methods in the interface are by default abstract methods therefore need not to declare with abstract keyword.
5. Abstract class can have methods with definition.	5. Interface should not have methods with definition.
6. Variables in abstract class have default access.	6. Variables in the interface are by default public static and final.
7. A subclass should extend from only one abstract class. (i.e. single inheritance)	7. A subclass can implements from any number of interfaces (i.e. multiple inheritance).
8. Abstract classes are not pure abstract classes.	8. Interfaces are pure abstract classes.