INTERFACES

An interface is a reference type in Java. It is similar to class. It is a **collection of abstract methods**. Along with abstract methods, an interface may **also contain constants, default methods, static methods, and nested types**. Method bodies exist only for default methods and staticmethods.

An interface is similar to a class in the following ways:

- An interface **can contain any number of methods**.
- **An interface is written in a file with a .java extension**, with the name of the interfacematching the name of the file.
- **The byte code of an interface appears in a .class file.**
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

Uses of interface:

- Since java does not support **multiple inheritance** in case of class, it can be achievedby using interface.
- It is also used to achieve **loose coupling.**
- Interfaces are used to implement **abstraction**.

Defining an Interface

An interface is defined much like a class.

Syntax:

accessspecifier interface interfacename

{

return-type method-name1(parameter- list);

return-type method-name2(parameter-list);

type final-varname1 = value;

type final-varname2 = value;

// ...

return-type method-nameN(parameter-list);

type final-varnameN = value;

}

When no access specifier is included, then default access results, and the interface is only available to other members of the package in which it is declared. When it is declared as pub-lic, the interface can be used by any other code.

- The java file must have the same name as the interface.

parameter list. They are abstract methods; there can be no default implementation of any method specified within an interface.

- Each class that includes an interface must implement all of the methods.
- Variables can be declared inside of interface declarations. They are implicitly final and static, meaning they cannot be changed by the implementing class. They must also be initialized.
- All methods and variables are implicitly public.

Sample Code:

The following code declares a simple interface Animal that contains two methods called eat() and travel() that take no parameter.

```
/* File name : Animal.java
*/interface Animal {
  public void eat();
  public void travel();
}
```

Implementing an Interface

Once an interface has been defined, one or more classes can implement that interface. To implement an interface, the **'implements'** clause is included in a class definition and then the methods defined by the interface are created.

Syntax:

```
class classname [extends superclass] [implements interface [,interface...]]
{
// class-body
}
```

Properties of java interface

- If a class implements more than one interface, the interfaces are separated with a comma.
- If a class implements two interfaces that declare the same method, then the same method will be used by clients of either interface.
- The methods that implement an interface must be declared public.
- The type signature of the implementing method must match exactly the type signature specified in the interface definition.

Rules

- A class can implement more than one interface at a time.
- A class can extend only one class, but can implement many interfaces.
- An interface can extend another interface, in a similar way as a class can extend

Sample Code 1:

The following code implements an interface Animal shown earlier.

```
/* File name : MammalInt.java */
public class Mammal implements Animal
{
  public void eat()
{
    System.out.println("Mammal eats");
  }
  public void travel()
{
    System.out.println("Mammal travels");
  }
  public int noOfLegs()
{
    return 0;
  }
  public static void main(String args[])
{
    Mammal m = new Mammal();
    m.eat();
    m.travel();
  }
}
```

Output:

```
Mammal eats Mammal
travels
```

It is both permissible and common for classes that implement interfaces to define additional members of their own. In the above code, Mammal class defines additional method called noOfLegs().

Sample Code 2:

The following code initially defines an interface 'Sample' with two members. This interface is implemented by a class named 'testClass'.

```
import java.io.*;
// A simple interface
interface Sample
```

```
{
   final String name = "Shree";
   void display();
}
 // A class that implements interface.
public class testClass implements Sample
{
   public void display()
   {
      System.out.println("Welcome");
   }
   public static void main (String[] args)
   {
      testClass t = new testClass();
      t.display();
      System.out.println(name);
   }
}
```

Output:

```
Welcome
Shree
```

Sample Code 3:

In this example, Drawable interface has only one method. Its implementation is providedby Rectangle and Circle classes.

```
interface Drawable
{
void draw();
}
class Rectangle implements Drawable
{
public void draw()
{
   System.out.println("Drawing rectangle");
}
}
```

```
class Circle implements Drawable
{
public void draw()
{
   System.out.println("Drawing circle");
}
}
public class TestInterface
{
public static void main(String args[])
{
Drawable d=new Circle();
d.draw();
}
  }
```

Output:

```
Drawing circle
```

Nested Interface

An interface can be declared as a member of a class or another interface. Such an interface is called a member interface or a nested interface. A nested interface can be declared as public, private, or protected.

Sample Code:

```
interface MyInterfaceA
{
   void display(); interface
   MyInterfaceB
   {
      void myMethod();
   }
}
public class NestedInterfaceDemo1 implements MyInterfaceA.MyInterfaceB
{
    public void myMethod()
   {
       System.out.println("Nested interface method");
    }
    public static void main(String args[])
   {
      MyInterfaceA.MyInterfaceB obj= new NestedInterfaceDemo1();
    obj.myMethod();
   }
 }
```

Output:

```
Nested interface method
```