Interface: Declaration of Interface, Implementation of Interface,Multiple interface,Nested Interface,Static methods in interface.Functional interface

Packages and Java Library

Interfaces

An interface is a collection of method prototypes (method name followed by parameters list without any body). The syntax of a method prototype is as follows:

return-type method-name(parameters-list);

An interface can contain constants and method prototypes. The use of an interface is to abstract the class' behaviour from its definition. In this way an interface can specify a set of method prototypes which can be implemented by one or more classes.

It is used to achieve abstraction and multiple inheritance in Java.

Differences between interface and a class

- Objects can be created for classes, where as it is not possible for interfaces.
- Classes can contain methods with body, where as it is not possible in interfaces.
- Some classes can be final, where as interfaces cannot be declared as final.
- Some classes can be abstract, where as interfaces cannot be declared as abstract.
- Various access specifiers like public or private or default can be applied to classes, where as only public or default access specifier is applicable for top-level interface.

Declaration /Defining an Interface

The definition of an interface is very much similar to the definition of a class. The syntax for defining an interface is as follows:

```
interface interface-name
{
        return-type method1(parameters-list);
         return-type method2(parameters-list);
         …
        data-type variable-name1 = value;
        data-type variable-name2 = value;
        …
}
```

Access specifier before interface keyword can be public or default (no specifier). All the methods inside an interface definition do not contain any body. They end with a semi-colon after the parameters list. All variables declared inside an interface are by default final and static. All methods declared inside an interface are by default abstract and both variables as well as methods are implicitly public.

An example for Java interface is as follows:

```
public interface IMovable
{
void run();
void jump();
}
```

In the above example, IMovable is the interface name which contains two methods run and jump.

Implementing of Interfaces

The methods declared in an interface definition must be implemented by the class which inherits that interface. This process of a class implementing the methods in an interface is known as implementing interfaces.

It is mandatory for a class to implement (provide body) all the methods available in an interface. Otherwise, if a class provides implementation for only some methods (partial implementation) then the class should be made abstract. The methods of the interface must be declared as public in the class.

Syntax for implementing an interface is as follows:

```
class ClassName implements InterfaceName
{
 //Implementations of methods in the interface
 }
```

//Write a program to demonstrate interface

```
public interface IMovable
{
        int a=10;
        void run();
        void jump();
}
class person implements IMovable
{
        public void run()
        {
                System.out.println("Person is running.");
        }
        public void jump()
        {
                System.out.println("Person is jumping.");
        }
}
public class interfacedemo
{
public static void main(String[] args)
```

```
{
person p=new person();
p.run();
p.jump();
System.out.println(Imovable.a);
}
```

Output:

```
Person is running
Person is jumping
10
```

Variables in interface:

An interface can contain constants (final variables). A class that implements an interface containing constants can use them as if they were declared in the class. Example demonstrating constants in an interface is given below:

```
interface iarea
{
        double PI=3.142;
}
public class interfacevariable implements iarea
{
        public static void main(String[] args)
        System.out.println("PI value=",+PI);
}
}
```

Output:

```
PI value=3.142
```

```
//Write a program to demonstrate swapping of numbers using interface
interface ISwap
{
        void swap(int x, int y);
}
 class SwapTemp implements ISwap
{
        public void swap(int x, int y)
        {
                int temp;
                temp = x;
                x = y;
                y = temp;
                System.out.println("After interchange, x = " + x + ", y = " + y);
        }
```

```
}

 class SwapNoTemp implements ISwap
{
        public void swap(int x, int y)
        {
                x = x + y;
                y = x - y;
                x = x - y;
                System.out.println("After interchange, x = " + x + ", y = " + y);
        }
}
public class Main{
public static void main(String[] args)
{
        SwapTemp s = new SwapTemp();
        s.swap(10, 20);
        SwapNoTemp = new SwapNoTemp();
        s.swap(10, 20);
}
}
```

Output:

After interchange, x =20 , y = 10

After interchange, x =20 , y = 10

Multiple interfaces:

Multiple inheritances by interface occur if a class implements multiple interfaces or also if an interface itself extends multiple interfaces.

Syntax:

```
interface A
{
        //statements
}
interface B
{
        //statements
}
class c implements A,B
{
        // statements
}
```

//write a program to demonstrate Multiple inheritance by interface

```
interface in1
{
        int a=10;
        public void show();
}
interface in2
{
        int a=10;
        public void show();
}
class c implements in1,in2
{
        public void show()
        {
          System.out.println("A class can implement more than one interface");
        }
}
class multipleinheritance
{
public static void main(String[] args)
{
        c c1=new c();
        c1.show();
        System.out.println(in1.a);
        System.out.println(in2.a);
}
}
```

Output:

A class can implement more than one interface

10

10

Extending Interfaces

Like classes, interfaces can also be extended to provide new functionality. Like classes, we use extends keyword for extending an interface.

Syntax for extending an interface is given below:

```
interface NewInterface extends OldInterface
 {
        //Method prototypes here
 }
```

Example:

```
interface IAreaPeri extends IArea
{
        void peri();
}
```

Some might think instead of creating a new interface and extending the existing interface, why can't we include the new functionality in the existing interface directly? We
can't because the applications of people who are using the existing interface will break if new functionality is included. To avoid that we create a new interface.
All classes which implement the new interface must provide implementation for all the methods in the new interface plus for methods in its parent interface.

Following example demonstrates extending an interface:

```
interface IArea
{
        double PI = 3.142;
        void area();
}
 interface IAreaPeri extends IArea
{
        void peri();
}

class Shape implements IAreaPeri
{
        public void area()
        {
                System.out.println("Area of shape");
        }
        public void peri()
        {
                System.out.println("Perimeter of shape");
        }
}
public class Main
{
public static void main(String[] args)
{
        Shape s = new Shape();
        s.area();
        s.peri();
```

```
}
}
```

Output:

Area of shape
Perimeter of shape

Nested Interfaces

An interface which is declared inside a class or another interface is called a nested interface or a member interface. A nested interface can be declared as public, private or protected. Let's look at an example of how to create and use a nested interface:

Syntax:

```
interface Outerinterfacename
{
        interface innerinterfacename
        {
        //statements
        }
}
```

Class implements the nested interface:

Syntax:

```
class classname implements Outerinterfacename.innerinterfacename
```

Accessing of variables in innerinterface:

```
Outerinterfacename.innerinterfacename.variable
```

//Write a program to demonstrate nested interface

```
interface OuterInterface
{
  int a=10;
  void display();
        interface InnerInterface
        {
                int a=24;
                void myMethod();
         }
}
class C implements OuterInterface.InnerInterface
{
  public void myMethod()
 {
        System.out.println("This is the  method in innerinterface");
  }
  public void display()
```

```
 {
        System.out.println("This is method in outer interface");
 }
}
 class Main{
 public static void main(String args[])
{
  C c1 = new C();
  System.out.println(OuterInterface.a);
  System.out.println(OuterInterface.InnerInterface.a);
  c1.myMethod();
  c1.display();

 }
}
```

Output:

10

24

This is the method in innerinterface

This is method in outer interface

Static Methods in interface

We can define static methods inside the interface. Static methods are used to define utility methods.

```
interface geometry
{
        void draw();
        static int cube(int  x)
        {
         return x*x*x;
        }
}
class rectangle implements geometry
{
        public void draw()
        {
        System.out.println("drawing rectangle");
        }
}
class staticinterface
{
 public static void main(String args[])
```

```
    {
        rectangle r=new rectangle();
        r.draw();
        System.out.println(geometry.cube(3));
    }
}
```

Output:

```
drawing rectangle
27
```

Functional interface:

An Interface that contains exactly one abstract method is known as functional interface. It can have any number of default, static methods but can contain only one abstract method. It can also declare methods of object class.

Functional Interface is also known as Single Abstract Method Interfaces or SAM Interfaces. It is a new feature in Java, which helps to achieve functional programming approach.

From Java 8 onwards, lambda expressions can be used to represent the instance of a functional interface.

Runnable, ActionListener, Comparable are some of the examples of functional interfaces.

//Write a program to demonstrate functional interfaces

```
interface Square
{
   int calculate(int x);
}

class Main
{
   public static void main(String args[])
   {
      int a = 5;
      // lambda expression to define the calculate method
      Square s = (int x)->x*x;
      // parameter passed and return type must be
      // same as defined in the prototype
      System.out.println( s.calculate(a));
   }
}
```

Output:

```
25
```

Differences between Abstract classes and Interface

Abstract Class	Interface
1)abstract keyword is used	1)interface is the keyword used
2)sub classes extends abstract classes	2)Sub classes implements the interfaces
3)Abstract class can have both abstract and non abstract methods	3) Interfaces can have only abstract methods. From Java 8,it can have default and static methods
4)Abstract classes doesn't support Multiple inheritance	4) Interfaces support Multiple inheritance
5) Abstract classes can have final,non final,static and non static variables	5) Interfaces has only static and final variables
6) Abstract classes supports all access modifiers	6)Any method or variable inside an interface is public by default
7) Abstract classes can have constructors and destructors	7)Interfaces are not classes, so no constructors and destructors
8)Objects are created in Abstract class	8)No objects are created