INTERFACE

An interface is a blueprint of a class, it is a collection of abstract methods and it is used to achieve multiple inheritance. The interface is a mechanism to achieve abstraction, and it represents the IS-A relationship.

An Interface is about capabilities like a Player may be an interface and any class implementing Player must be able to (or must implement) move(). So it specifies a set of methods that the class has to implement and If a class implements an interface and does not provide method bodies for all functions specified in the interface, then the class must be declared abstract.

The interface cannot be instantiated just like the abstract class. Since Java 8 we can have default and static methods in the interface and since Java 9 we can have private methods in an interface.

WHY USE INTERFACE

There are mainly three reasons to use interface:

1. It is used to achieve abstraction
2. By interface we can support the functionality of multiple inheritance
3. It can be used to achieve loose coupling
4. Interfaces are used to implement abstraction. So the question arises why use interfaces when we have abstract classes?

The reason is, abstract classes may contain non-final variables, whereas variables in the interface are final, public and static.

DECLARING AN INTERFACE

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the

fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

Syntax

Interface <interface_name>{

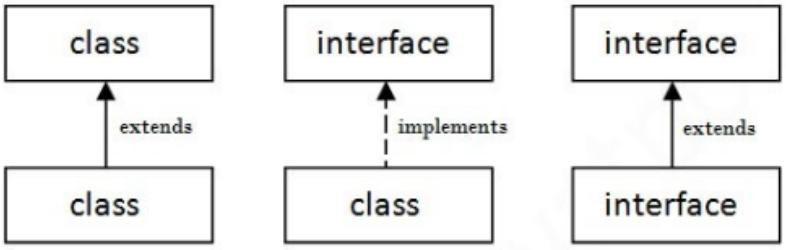
//declare constant fields

//declare methods that abstract by default

}

Relationship Between Classes and Interfaces

As shown below, a class extends another class, and an interface extends another interface but a class implements an interface.



Example

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

```
//Interface declaration: by first user
interface Drawable{
void draw();
}
//Implementation: by second user
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
class Circle implements Drawable{
public void draw(){System.out.println("drawing circle");}
}
//Using interface: by third user
class TestInterface1{
public static void main(String args[]){
Drawable d=new Circle();//In real scenario, object is provided by method e.g. getDrawable()
d.draw();
}}
```

Example 2

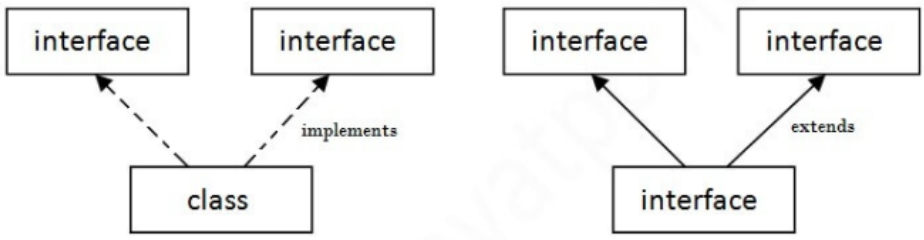
In this Example, a java interface which provides the implementation of the Bank interface.

```
interface Bank{
float rateOfInterest();
}
class SBI implements Bank{
public float rateOfInterest(){return 9.15f;}
}
class PNB implements Bank{
public float rateOfInterest(){return 9.7f;}
}
class TestInterface2{
public static void main(String[] args){
Bank b=new SBI();
System.out.println("ROI: "+b.rateOfInterest());
}}
```

MULTIPLE INHERITANCE IN JAVA BY INTERFACE

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance. Multiple inheritance is not supported in the case of class

because of ambiguity. However, it is supported in case of an interface because there is no ambiguity. It is because its implementation is provided by the implementation class. For example:



Multiple Inheritance in Java

```
interface Printable{
void print();
}
interface Showable{
void show();
}
class A7 implements Printable,Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
 }
}
```

As you can see in the above example, Printable and Showable interfaces have the same methods but its implementation is provided by class TestTnterface1, so there is no ambiguity.

ABSTRACTION

Abstraction is the process of hiding the implementation details and showing only functionality to the user. That is, it shows only essential things to the user and hides the internal details. Abstraction lets you focus on what the object does instead of hot it does it.

There are two ways to achieve abstraction in Java that is the abstract class, and the interface which we have discussed above.

A class which is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated. It has constructors and static methods also. And it can have final method which will force the subclass not to change the body of the method.

ABSTRACT METHOD

A method which is declared as abstract and does not have implementation is known as an abstract method.

Example

In this example, Bike is an abstract class that contains only one abstract method run. Its implementation is provided by the Honda class.

```
abstract class Bike{
  abstract void run();
}
class Honda4 extends Bike{
void run(){System.out.println("running safely");}
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
}
```

Example 2

In this example, if you create the instance of Rectangle class, draw() method of Rectangle class will be invoked.

```
abstract class Shape{
abstract void draw();
}
//In real scenario, implementation is provided by others i.e. unknown by end user
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle");}
}
class Circle1 extends Shape{
void draw(){System.out.println("drawing circle");}
}
//In real scenario, method is called by programmer or user
class TestAbstraction1{
public static void main(String args[]){
Shape s=new Circle1();//In a real scenario, object is provided through method, e.g., getShape() method
s.draw();
}
}
```

An abstract class can have a data member, abstract method, method body (non-abstract method), constructor, and even main() method.

Example

```
//Example of an abstract class that has abstract and non-abstract methods
 abstract class Bike{
   Bike(){System.out.println("bike is created");}
   abstract void run();
   void changeGear(){System.out.println("gear changed");}
 }
//Creating a Child class which inherits Abstract class
 class Honda extends Bike{
 void run(){System.out.println("running safely..");}
 }
//Creating a Test class which calls abstract and non-abstract methods
 class TestAbstraction2{
 public static void main(String args[]){
  Bike obj = new Honda();
  obj.run();
  obj.changeGear();
 }
}
```

DIFFERENCE BETWEEN ABSTRACT CLASS AND INTERFACE

Abstract Class	Interface
Abstract Class can have abstract and non-abstract methods	Interfaces can have only abstract methods. Since java 8, it can have default and static methods also
Abstract class doesn't support multiple inheritance	Interface supports multiple inheritance
Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
Abstract class can provide the implementation of interface	Interface can't provide the implementation of abstract class
Abstract keyword is used to declare abstract class	Interface keyword is used to declare interface
Abstract class can extend another Abstract class and implement multiple java interfaces	Interface can extend another Java interface only
Abstract class can be extended using keyword "extends"	Interface can be implemented using keyword "implements"
Abstract class can have class members like private, protected, etc	Members of java interface are public by default