OOP-5

Abstract class in Java

A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

Abstraction in Java

- **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.
- Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.
- Abstraction lets you focus on what the object does instead of how it does it.

Cont…

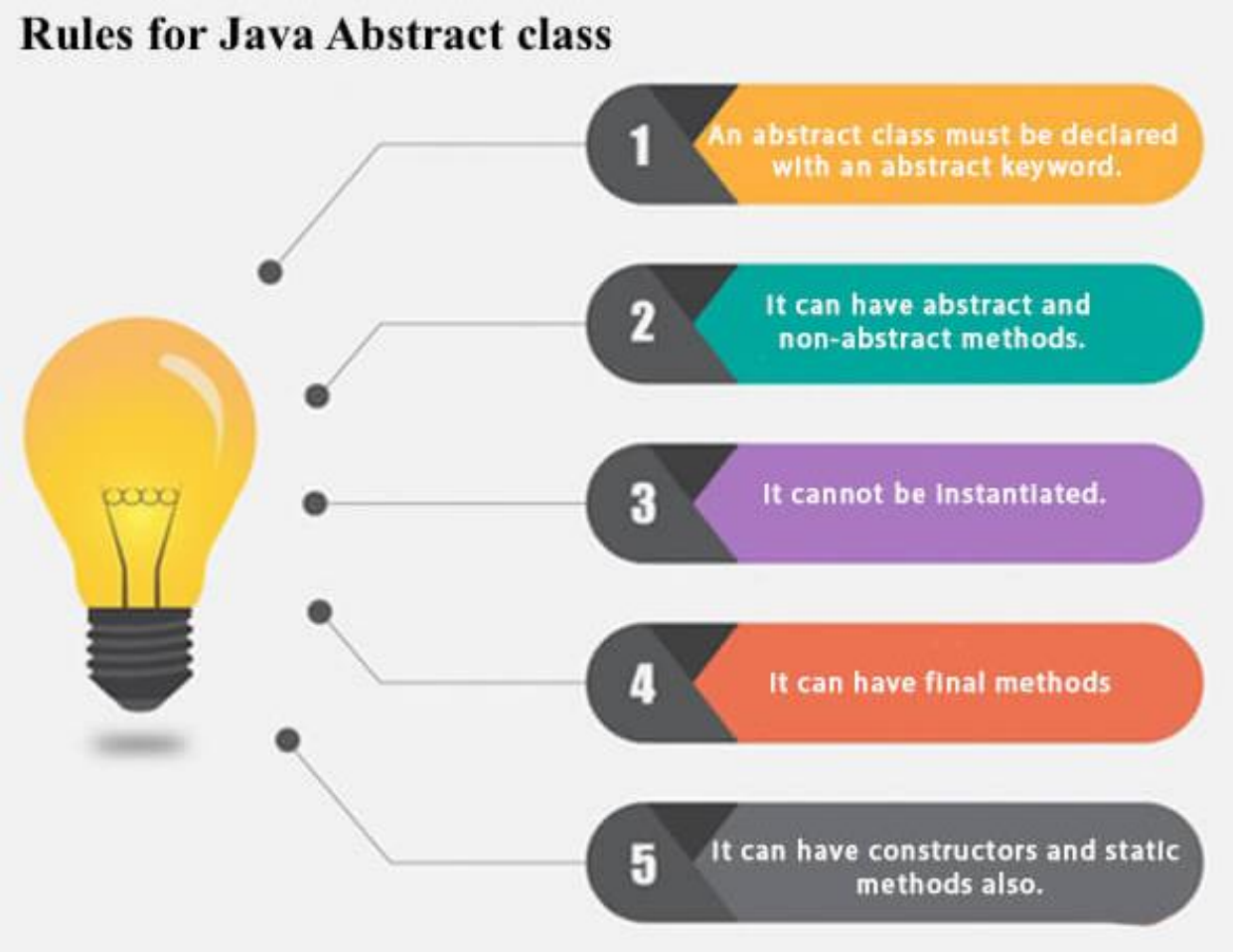
Ways to achieve Abstraction

There are two ways to achieve abstraction in java

- Abstract class (0 to 100%)
- Interface (100%)

An abstract class needs to be extended and its method implemented. It cannot be instantiated.

Points to Remember



Abstract method

Example of abstract class:

```
abstract class A{}
```

Abstract Method in Java

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

Example of abstract method:

```
abstract void printStatus();//no method body and abstract
```

Example of Abstract class that has an abstract method

```
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();
}
```

Output:

```
running safely
```

Understanding the real scenario of Abstract class

- In this example, Shape is the abstract class, and its implementation is provided by the Rectangle and Circle classes. Mostly, we don't know about the implementation class (which is hidden to the end user)

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

Cont…

```
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();
}
}
```

Output:

```
drawing circle
```

Another example of Abstract class in java

```
abstract class Bank{
abstract int getRateOfInterest();
}
class SBI extends Bank{
int getRateOfInterest(){return 7;}
}
class PNB extends Bank{
int getRateOfInterest(){return 8;}
}

class TestBank{
public static void main(String args[]){
Bank b;
b=new SBI();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
b=new PNB();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
}}
```

```
Rate of Interest is: 7 %
Rate of Interest is: 8 %
```

Abstract class having constructor, data member and methods

```
//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..");}
}
```

Cont…

- //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();
- }
- }

Rule: If there is an abstract method in a class, that class must be abstract.

- **class** Bike12{
- **abstract void** run();
- }

```
compile time error
```

Interface in Java

An **interface in java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

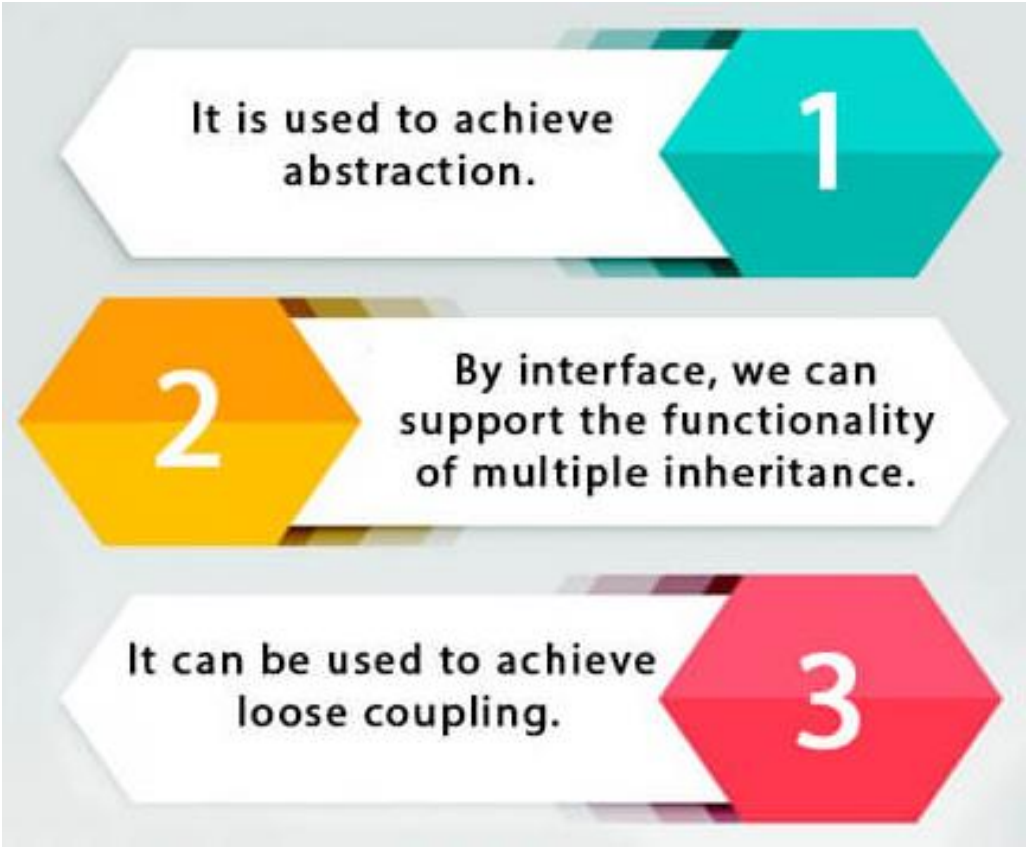
In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

- Java Interface also **represents the IS-A relationship**.
- It cannot be instantiated just like the abstract class.

Syntax :

```
interface <interface_name>
{
// declare constant fields
// declare methods that abstract by default.
}
```

Why use Java interface?

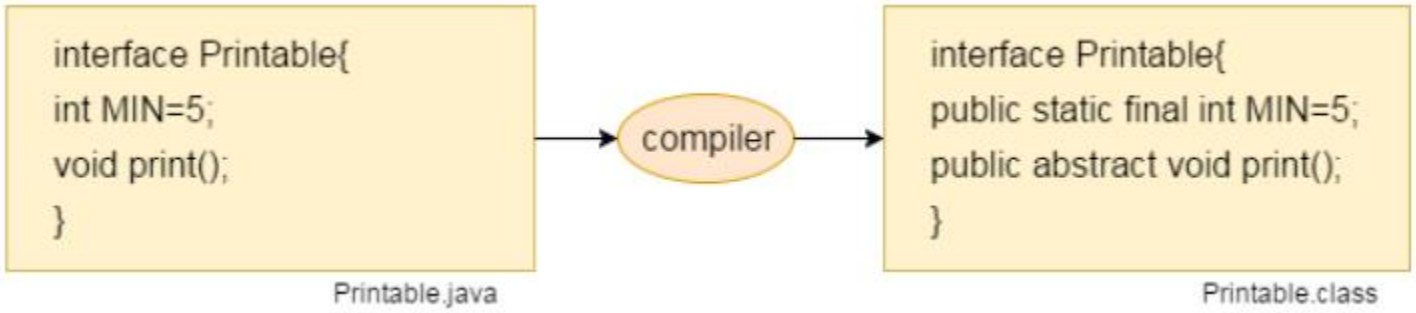


Interfaces are used to implement abstraction. So the question arises why use interfaces when we have abstract classes? The reason is, abstract classes may contain non-final variables, whereas variables in interface are final, public and static.

```
// A simple interface
interface Player
{
    final int id = 10;
    int move();
}
```

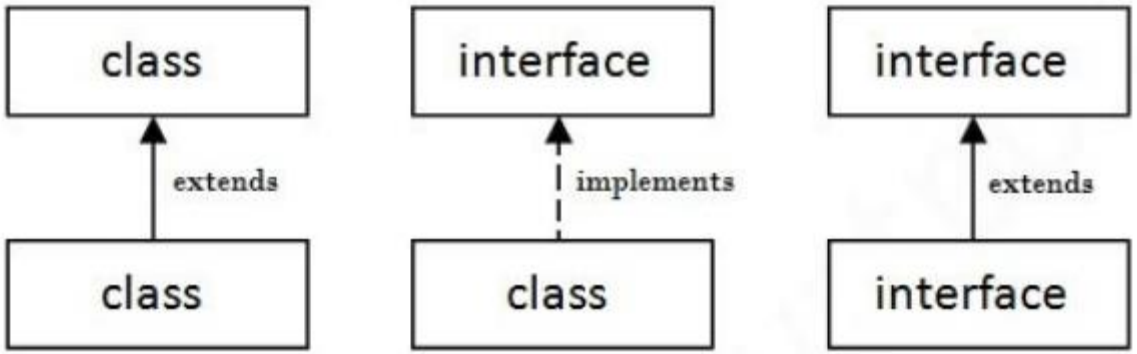
The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.

In other words, Interface fields are public, static and final by default, and the methods are public and abstract.



The relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.



```
// Java program to demonstrate working of
// interface.
import java.io.*;

// A simple interface
interface in1
{
    // public, static and final
    final int a = 10;

    // public and abstract
    void display();
}
```

```
// A class that implements interface.
class testClass implements in1
{
    // Implementing the capabilities of interface.
    public void display()
    {
        System.out.println("Geek");
    }
 }
    public static void main (String[] args)
    {
        testClass t = new testClass();
        t.display();
        System.out.println(t.a);
    }
```

Output:

```
Geek
10
```

A real world example

Let’s consider the example of vehicles like bicycle, car, bike……,they have common functionalities. So we make an interface and put all these common functionalities. And lets Bicylce, Bike, car ….etc implement all these functionalities in their own class in their own way.

```
interface Vehicle {
    // all are the abstract methods.
    void changeGear(int a);
    void speedUp(int a);
    void applyBrakes(int a);
}
```

```
class Bicycle implements Vehicle{

    int speed;
    int gear;
     // to change gear
    @Override
    public void changeGear(int newGear){
       gear = newGear;
    }
    // to increase speed
    @Override
    public void speedUp(int increment){
       speed = speed + increment;
    }
    // to decrease speed
    @Override
    public void applyBrakes(int decrement){
       speed = speed - decrement;
    }
    public void printStates() {
        System.out.println("speed: " + speed
            + " gear: " + gear);
    }
}
```

```
class Bike implements Vehicle {

    int speed;
    int gear;

    // to change gear
    @Override
    public void changeGear(int newGear){
       gear = newGear;
    }
    // to increase speed
    @Override
    public void speedUp(int increment){
       speed = speed + increment;
    }
    // to decrease speed
    @Override
    public void applyBrakes(int decrement){
       speed = speed - decrement;
    }
    public void printStates() {
        System.out.println("speed: " + speed
            + " gear: " + gear);
    }

}
```

```
class GFG {

    public static void main (String[] args) {

        // creating an instance of Bicycle
        // doing some operations
        Bicycle bicycle = new Bicycle();
        bicycle.changeGear(2);
        bicycle.speedUp(3);
        bicycle.applyBrakes(1);

        System.out.println("Bicycle present state :");
        bicycle.printStates();

        // creating instance of bike.
        Bike bike = new Bike();
        bike.changeGear(1);
        bike.speedUp(4);
        bike.applyBrakes(3);

        System.out.println("Bike present state :");
        bike.printStates();
    }
}
```

Output;

```
Bicycle present state :
speed: 2 gear: 2
Bike present state :
speed: 1 gear: 1
```