CONCEPT OF ABSTRACTION CLASSES AND INTERFACES IN JAVA

Java abstract class is a class that can not be initiated by itself, it needs to be subclassed by another class to use its properties. An abstract class is declared using the "abstract" keyword in its class definition.

Illustration of Abstract class:

```
abstract class Shape
{
   int color;
   // An abstract function
   abstract void draw();
}
```

In Java, the following some *important observations* about abstract classes are as follows:

1. An instance of an abstract class can not be created.
2. Constructors are allowed.
3. We can have an abstract class without any abstract method.
4. There can be a final method in abstract class but any abstract method in class(abstract class) can not be declared as final or in simpler terms final method can not be abstract itself as it will yield an error: "Illegal combination of modifiers: abstract and final"
5. We can define static methods in an abstract class
6. We can use the abstract keyword for declaring *top-level classes (Outer class) as well as inner classes* as abstract
7. If a class contains at least one abstract method then compulsory should declare a class as abstract
8. If the Child class is unable to provide implementation to all abstract methods of the Parent class then we should declare that Child class as abstract so that the next level Child class should provide implementation to the remaining abstract method

```
// Java Program to Illustrate that an instance of Abstract Class can not be created
 // Class 1 Abstract class
abstract class Base {
   abstract void fun();
}
 // Class 2
class Derived extends Base {
   void fun()
   {
      System.out.println("Derived fun() called");
   }
}
 // Class 3 Main class
class Main {
   public static void main(String args[])
   {
      // Uncommenting the following line will cause
      // compiler error as the line tries to create an
      // instance of abstract class.
       //Base b = new Base();

      // We can have references of Base type.
      Base b = new Derived();
      b.fun();
   }
}
```

OUTPUT

Derived fun() called

```
// Abstract class
abstract class Sunstar {
   abstract void printInfo();
}

// Abstraction performed using extends
class Employee extends Sunstar {
   void printInfo()   {
      String name = "avinash";
      int age = 21;
      float salary = 222.2F;
       System.out.println(name);
      System.out.println(age);
      System.out.println(salary);
   }
}

// Base class
class Base {
   public static void main(String args[])
   {
      Sunstar s = new Employee();
      s.printInfo();
   }
}
```

OUTPUT

avinash

21

222.2

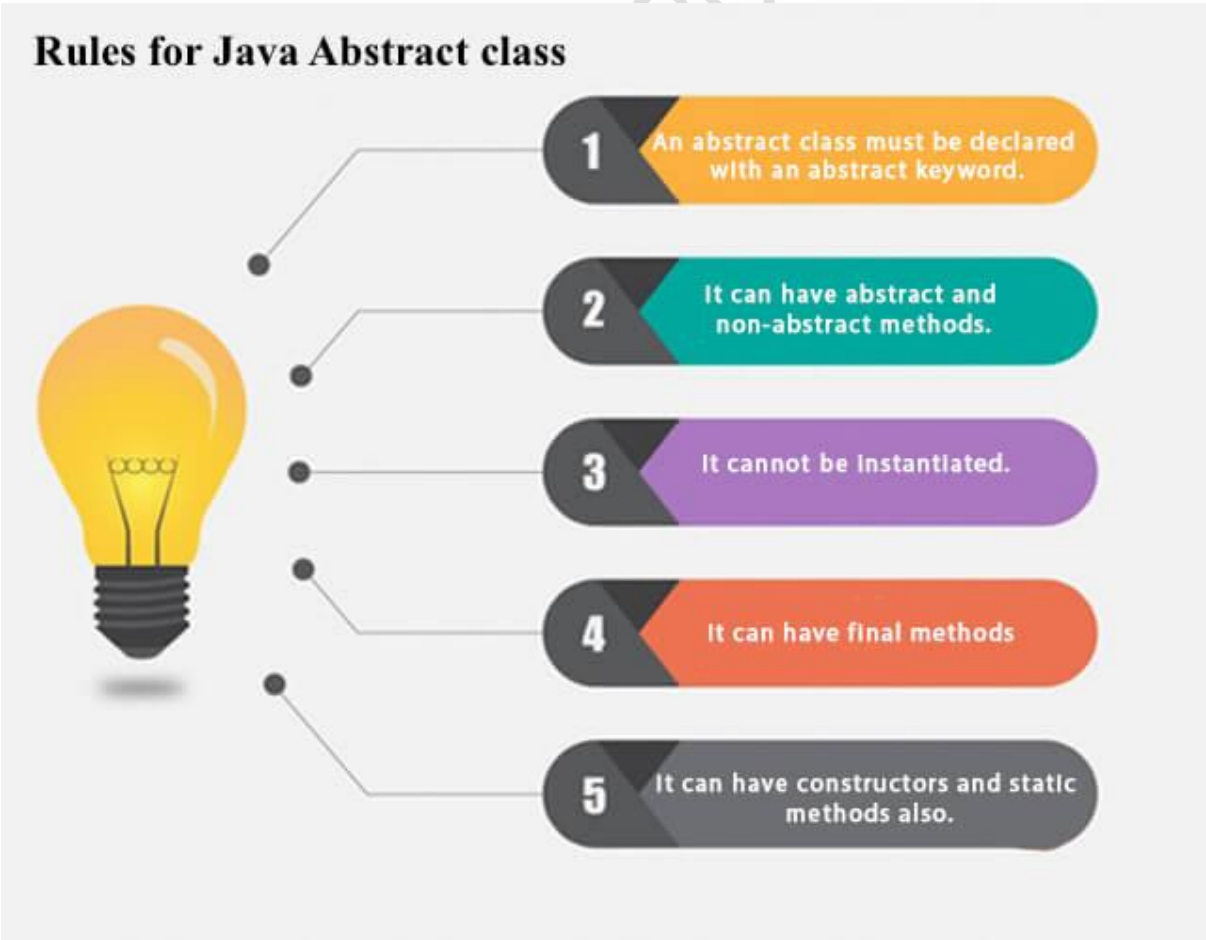
Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

Points to Remember

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.



Q1. Why do we use abstract?

One key reason we use abstract concepts is to simplify complexity. Imagine trying to explain the entire universe with every single atom and star! Abstracts let us zoom out, grab the main ideas like gravity and energy, and make sense of it all without getting lost in the details.

Here are some other reasons why we use abstract in Java:

1. ***Abstraction**: Abstract classes are used to define a generic template for other classes to follow. They define a set of rules and guidelines that their subclasses must follow. By providing an abstract class, we can ensure that the classes that extend it have a consistent structure and behavior. This makes the code more organized and easier to maintain.*

Q2. What is the Difference between Encapsulation and Data Abstraction?

Here are some key difference b/w encapsulation and abstration:

Encapsulation	***Abstraction***
Encapsulation is data hiding(information hiding)	*Abstraction is detailed hiding(implementation hiding).*
Encapsulation groups together data and methods that act upon the data	*Data Abstraction deal with exposing the interface to the user and hiding the details of implementation*
Encapsulated classes are Java classes that follow data hiding and abstraction	*Implementation of abstraction is done using abstract classes and interface*
Encapsulation is a procedure that takes place at the implementation level	*abstraction is a design-level process*

Q3. What is a real-life example of data abstraction?

Television remote control *is an excellent real-life example of abstraction. It simplifies the interaction with a TV by hiding the complexity behind simple buttons and symbols, making it easy without needing to understand the technical details of how the TV functions.*

*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.

*Since Java 8, we can have **default and static methods** in an interface.*

*Since Java 9, we can have **private methods** in an interface.*

Why use Java interface?

There are mainly three reasons to use interface. They are given below.

- *It is used to achieve abstraction.*
- *By interface, we can support the functionality of multiple inheritance.*
- *It can be used to achieve loose coupling.*

How to declare 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.
}
```

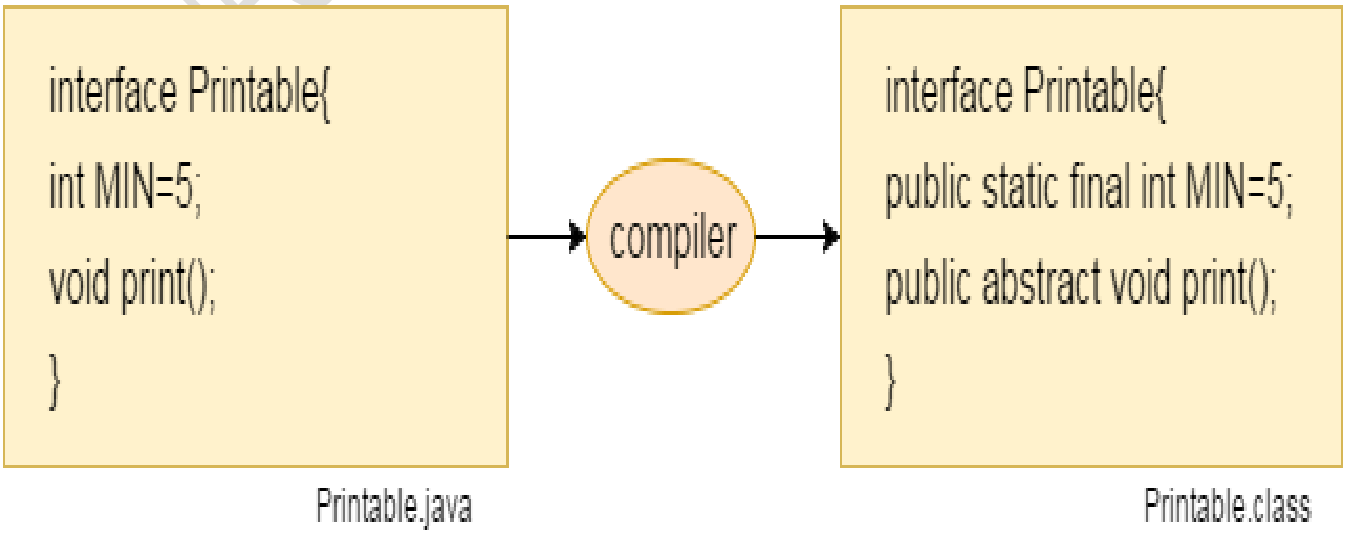
Java 8 Interface Improvement

Since Java 8, interface can have default and static methods which is discussed later.

Internal addition by the compiler

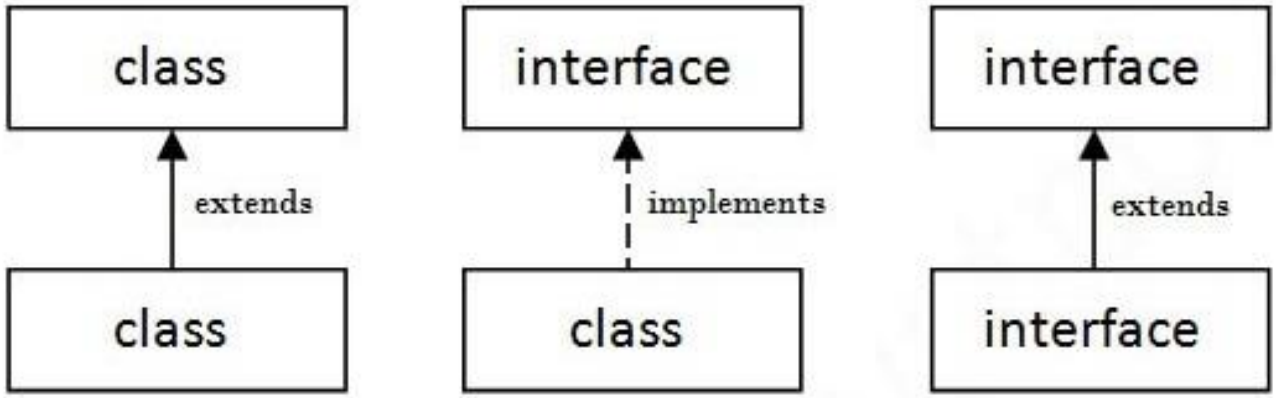
The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.

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 Interface Example

In this example, the Printable interface has only one method, and its implementation is provided in the A6 class.

```
interface printable{
void print();
}
class A6 implements printable{
public void print(){System.out.println("Hello");}


public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }
}
```

OUTPUT

Hello

Java Interface Example: Drawable

In this example, the Drawable interface has only one method. Its implementation is provided by Rectangle and Circle classes. In a real scenario, an interface is defined by someone else, but its implementation is provided by different implementation providers. Moreover, it is used by someone else. The implementation part is hidden by the user who uses the interface.

File: TestInterface1.java

```
//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();
}}
```

OUTPUT

```
drawing circle
```

Java Interface Example: Bank

Let's see another example of java interface which provides the implementation of Bank interface.

File: TestInterface2.java

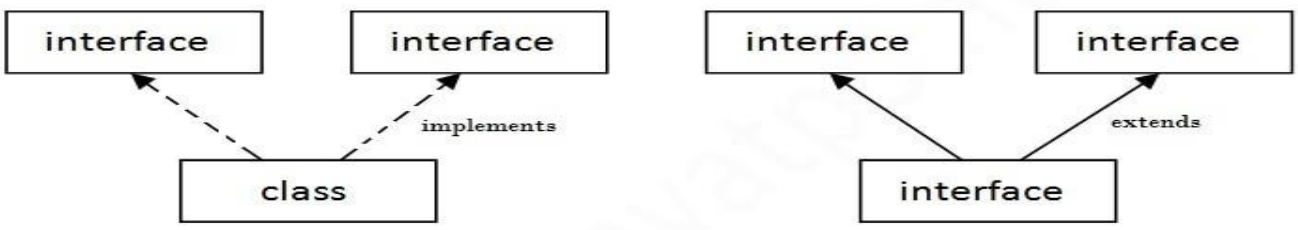
```
interface Bank{
float rateOfInterest();
}
class SBI implements Bank{
public float rateOfInterest(){return 9.15f;}
}
class PNB implements Bank{
public float rateOfInterest(){return 9.7f;}
}
class TestInterface2{
public static void main(String[] args){
Bank b=new SBI();
System.out.println("ROI: "+b.rateOfInterest());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

OUTPUT

ROI: 9.15

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 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();
 }
}
```

OUTPUT

Hello

Welcome

Multiple inheritance is not supported through class in java, but it is possible by an interface, why?

As we have explained in the inheritance chapter, 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:

```
interface Printable{
void print();
}
interface Showable{
void print();
}

class TestInterface3 implements Printable, Showable{
public void print(){System.out.println("Hello");}
public static void main(String args[]){
TestInterface3 obj = new TestInterface3();
obj.print();
 }
}
```

OUTPUT

Hello

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

Interface inheritance

A class implements an interface, but one interface extends another interface.

```
interface Printable{
void print();
}
interface Showable extends Printable{
void show();
}
class TestInterface4 implements Showable{
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
TestInterface4 obj = new TestInterface4();
obj.print();
obj.show();
 }
}
```

OUTPUT

Hello

Welcome

Q4. What is the Difference between Abstract Classes and Interfaces in Java?

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.

Abstract class	***Interface***
*1) Abstract class can **have abstract and non-abstract** methods.*	*Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.*
*2) Abstract class **doesn't support multiple inheritance**.*	*Interface **supports multiple inheritance**.*
*3) Abstract class **can have final, non-final, static and non-static variables**.*	*Interface has **only static and final variables**.*
*4) Abstract class **can provide the implementation of interface**.*	*Interface **can't provide the implementation of abstract class**.*
*5) The **abstract keyword** is used to declare abstract class.*	*The **interface keyword** is used to declare interface.*
*6) An **abstract class** can extend another Java class and implement multiple Java interfaces.*	*An **interface** can extend another Java interface only.*
*7) An **abstract class** can be extended using keyword "extends".*	*An **interface** can be implemented using keyword "implements".*
*8) A Java **abstract class** can have class members like private, protected, etc.*	*Members of a Java interface are public by default.*
*9)**Example:*** *public abstract class Shape{* *public abstract void draw();* *}*	***Example:*** *public interface Drawable{* *void draw();* *}*

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

Example of abstract class and interface in Java

Let's see a simple example where we are using interface and abstract class both.

```
//Creating interface that has 4 methods
interface A{
void a();//bydefault, public and abstract
void b();
void c();
void d();
}

//Creating abstract class that provides the implementation of one method of A interface
abstract class B implements A{
public void c(){System.out.println("I am C");}
}

//Creating subclass of abstract class, now we need to provide the implementation of rest of the methods
class M extends B{
public void a(){System.out.println("I am a");}
public void b(){System.out.println("I am b");}
public void d(){System.out.println("I am d");}
}

//Creating a test class that calls the methods of A interface
class Test5{
public static void main(String args[]){
A a=new M();
a.a();
a.b();
```

a.c();

a.d();

}}

OUTPUT

I am a

I am b

I am C

I am d

PRACTICE TASKS

1. Write a Java program to create an abstract class Animal with an abstract method called sound(). Create subclasses Lion and Tiger that extend the Animal class and implement the sound() method to make a specific sound for each animal.

Click me to see the solution

2. Write a Java program to create an abstract class Shape with abstract methods calculateArea() and calculatePerimeter(). Create subclasses Circle and Triangle that extend the Shape class and implement the respective methods to calculate the area and perimeter of each shape.

Click me to see the solution

3. Write a Java program to create an abstract class BankAccount with abstract methods deposit() and withdraw(). Create subclasses: SavingsAccount and CurrentAccount that extend the BankAccount class and implement the respective methods to handle deposits and withdrawals for each account type.

Click me to see the solution

4. Write a Java program to create an abstract class Animal with abstract methods eat() and sleep(). Create subclasses Lion, Tiger, and Deer that extend the Animal class and implement the eat() and sleep() methods differently based on their specific behavior.

Click me to see the solution

5. Write a Java program to create an abstract class Employee with abstract methods calculateSalary() and displayInfo(). Create subclasses Manager and Programmer that extend the Employee class and implement the respective methods to calculate salary and display information for each role.

Click me to see the solution

6. Write a Java program to create an abstract class Shape3D with abstract methods calculateVolume() and calculateSurfaceArea(). Create subclasses Sphere and Cube that extend the Shape3D class and implement the respective methods to calculate the volume and surface area of each shape.

Click me to see the solution

7. Write a Java program to create an abstract class Vehicle with abstract methods startEngine() and stopEngine(). Create subclasses Car and Motorcycle that extend the Vehicle class and implement the respective methods to start and stop the engines for each vehicle type.

Click me to see the solution

8. Write a Java program to create an abstract class Person with abstract methods eat() and exercise(). Create subclasses Athlete and LazyPerson that extend the Person class and implement the respective methods to describe how each person eats and exercises.

Click me to see the solution

9. Write a Java program to create an abstract class Instrument with abstract methods play() and tune(). Create subclasses for Glockenspiel and Violin that extend the Instrument class and implement the respective methods to play and tune each instrument.

Click me to see the solution

10. Write a Java program to create an abstract class Shape2D with abstract methods draw() and resize(). Create subclasses Rectangle and Circle that extend the Shape2D class and implement the respective methods to draw and resize each shape.

Click me to see the solution

11. Write a Java program to create an abstract class Bird with abstract methods fly() and makeSound(). Create subclasses Eagle and Hawk that extend the Bird class and implement the respective methods to describe how each bird flies and makes a sound.

Click me to see the solution

12. Write a Java program to create an abstract class GeometricShape with abstract methods area() and perimeter(). Create subclasses Triangle and Square that extend the GeometricShape class and implement the respective methods to calculate the area and perimeter of each shape.

Click me to see the solution

Helpful Sites:

https://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/

https://www.codesdope.com/practice/java-abstract-class/