

***Question No.1***

Java final keyword is a non-access specifier that is **used to restrict a class, variable**, and method. ... And, if we declare a class as final, we restrict the other classes to inherit or extend it. In other words, the final classes can not be inherited by other classes.

***Question No.2***

The wrapper class in Java provides the mechanism to convert primitive into object and object into primitive.

It has two type

* **Boxing**
* **Unboxing**

***Example:***

1. **public** **class** WrapperExample1{
2. **public** **static** **void** main(String args[]){
3. //Converting int into Integer
4. **int** a=20;
5. Integer i=Integer.valueOf(a);//converting int into Integer explicitly
6. Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally
8. System.out.println(a+" "+i+" "+j);
9. }}

***Question No.3***

A Java virtual machine is a virtual machine that enables a computer to run Java programs as well as programs written in other languages that are also compiled to Java bytecode. The JVM is detailed by a specification that formally describes what is required in a JVM implementation.

***Question No.4***

The interface in Java is a mechanism to achieve [*abstraction*](https://www.javatpoint.com/abstract-class-in-java). There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple [inheritance in Java](https://www.javatpoint.com/inheritance-in-java).

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

***Question No.5***

The difference between Multiple and Multilevel inheritances is that Multiple Inheritance **is when a class inherits from many base classes** while Multilevel Inheritance is when a class inherits from a derived class, making that derived class a base class for a new class.

***Question No.6***

**Java inner class** or nested class is a class that is declared inside the class or interface.

We use inner classes to logically group classes and interfaces in one place to be more readable and maintainable.

Additionally, it can access all the members of the outer class, including private data members and methods.

#### **Syntax of Inner class**

1. **class** Java\_Outer\_class{
2. //code
3. **class** Java\_Inner\_class{
4. //code
5. }
6. }

***OR***

The class within in class is called a inner class.

***Question No.7***

Why the main () method in Java is always static? Java main() method is always static, **so that compiler can call it without the creation of an object or before the creation of an object of the class**. In any Java program, the main() method is the starting point from where compiler starts program execution.

***Question No.8***

Data Hiding and Encapsulation are two concepts of OOP. Data hiding is the process of protecting the members of the class from unauthorized access while Encapsulation is the process of wrapping the data members and **methods into a single unit**. This is the key difference between data hiding and encapsulation.

## Data hiding

* It is associated with data security.
* It also helps conceal the complexities of the application.
* It focuses on hiding/restricting the data usage.
* It is considered as a process and a technique.
* This data is always private and inaccessible.

## Encapsulation

* It can be defined as the wrapping up of data into a single module.
* This will hide the complicated and confidential information about the application.
* This encapsulated data can be private or public, depending on the requirement.
* It is considered as a sub-process in the bigger process of data hiding.

***Question No.9***

What happens if no catch handler matches the type of the thrown object? This **causes the search for a match to continue in the next enclosing try statement**. If there's a finally block, it will be executed before the exception goes to the next enclosing try statement.

***Question No.10***

Answer: **The String class is considered as immutable**, so that once it is created a String object cannot be changed. If there is a necessity to make alot of modifications to Strings of characters then StringBuffer should be used.

***Question No.11***

Runtime polymorphism or Dynamic Method Dispatch is **a process in which a call to an overridden method is resolved at runtime rather than compile-time**. In this process, an overridden method is called through the reference variable of a superclass.

***Question No.12***

this vs super keyword. The this **keyword points to a reference of the current class**, while the super keyword points to a reference of the parent class. this can be used to access variables and methods of the current class, and super can be used to access variables and methods of the parent class from the subclass.

# **Inheritance in Java**

**Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of [OOPs](https://www.javatpoint.com/java-oops-concepts) (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new [classes](https://www.javatpoint.com/object-and-class-in-java) that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a parent-child relationship.

### **Why use inheritance in java**

* For [Method Overriding](https://www.javatpoint.com/method-overriding-in-java) (so [runtime polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java) can be achieved).
* For Code Reusability.

### **Terms used in Inheritance**

* **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
* **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
* **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
* **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

### **The syntax of Java Inheritance**

1. **class** Subclass-name **extends** Superclass-name
2. {
3. //methods and fields
4. }

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

### **Java Inheritance Example**



As displayed in the above figure, Programmer is the subclass and Employee is the superclass. The relationship between the two classes is **Programmer IS-A Employee**. It means that Programmer is a type of Employee.

1. **class** Employee{
2. **float** salary=40000;
3. }
4. **class** Programmer **extends** Employee{
5. **int** bonus=10000;
6. **public** **static** **void** main(String args[]){
7. Programmer p=**new** Programmer();
8. System.out.println("Programmer salary is:"+p.salary);
9. System.out.println("Bonus of Programmer is:"+p.bonus);
10. }
11. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=Programmer)

Programmer salary is:40000.0

Bonus of programmer is:10000

In the above example, Programmer object can access the field of own class as well as of Employee class i.e. code reusability.

## **Types of inheritance in java**

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



#### **Note: Multiple inheritance is not supported in Java through class.**

When one class inherits multiple classes, it is known as multiple inheritance. For Example:



## **Single Inheritance Example**

When a class inherits another class, it is known as a single inheritance. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

*File: TestInheritance.java*

1. **class** Animal{
2. **void** eat(){System.out.println("eating...");}
3. }
4. **class** Dog **extends** Animal{
5. **void** bark(){System.out.println("barking...");}
6. }
7. **class** TestInheritance{
8. **public** **static** **void** main(String args[]){
9. Dog d=**new** Dog();
10. d.bark();
11. d.eat();
12. }}

Output:

barking...

eating...

## **Multilevel Inheritance Example**

When there is a chain of inheritance, it is known as multilevel inheritance. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

*File: TestInheritance2.java*

1. **class** Animal{
2. **void** eat(){System.out.println("eating...");}
3. }
4. **class** Dog **extends** Animal{
5. **void** bark(){System.out.println("barking...");}
6. }
7. **class** BabyDog **extends** Dog{
8. **void** weep(){System.out.println("weeping...");}
9. }
10. **class** TestInheritance2{
11. **public** **static** **void** main(String args[]){
12. BabyDog d=**new** BabyDog();
13. d.weep();
14. d.bark();
15. d.eat();
16. }}

Output:

weeping...

barking...

eating...

## **Hierarchical Inheritance Example**

When two or more classes inherits a single class, it is known as hierarchical inheritance. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

*File: TestInheritance3.java*

1. **class** Animal{
2. **void** eat(){System.out.println("eating...");}
3. }
4. **class** Dog **extends** Animal{
5. **void** bark(){System.out.println("barking...");}
6. }
7. **class** Cat **extends** Animal{
8. **void** meow(){System.out.println("meowing...");}
9. }
10. **class** TestInheritance3{
11. **public** **static** **void** main(String args[]){
12. Cat c=**new** Cat();
13. c.meow();
14. c.eat();
15. //c.bark();//C.T.Error
16. }}

Output:

meowing...

eating...

## **Q) Why multiple inheritance is not supported in java?**

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

1. **class** A{
2. **void** msg(){System.out.println("Hello");}
3. }
4. **class** B{
5. **void** msg(){System.out.println("Welcome");}
6. }
7. **class** C **extends** A,B{//suppose if it were
9. **public** **static** **void** main(String args[]){
10. C obj=**new** C();
11. obj.msg();//Now which msg() method would be invoked?
12. }
13. }

# **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*](https://www.javatpoint.com/abstract-class-in-java). There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple [inheritance in Java](https://www.javatpoint.com/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:**

1. **interface** <interface\_name>{
3. // declare constant fields
4. // declare methods that abstract
5. // by default.
6. }

## **Java 8 Interface Improvement**

## **Java 8 Interface Improvement**

Since [Java 8](https://www.javatpoint.com/java-8-features), 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.

1. **interface** printable{
2. **void** print();
3. }
4. **class** A6 **implements** printable{
5. **public** **void** print(){System.out.println("Hello");}
7. **public** **static** **void** main(String args[]){
8. A6 obj = **new** A6();
9. obj.print();
10. }
11. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=A6)

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*

1. //Interface declaration: by first user
2. **interface** Drawable{
3. **void** draw();
4. }
5. //Implementation: by second user
6. **class** Rectangle **implements** Drawable{
7. **public** **void** draw(){System.out.println("drawing rectangle");}
8. }
9. **class** Circle **implements** Drawable{
10. **public** **void** draw(){System.out.println("drawing circle");}
11. }
12. //Using interface: by third user
13. **class** TestInterface1{
14. **public** **static** **void** main(String args[]){
15. Drawable d=**new** Circle();//In real scenario, object is provided by method e.g. getDrawable()
16. d.draw();
17. }}

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestInterface1)

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*

1. **interface** Bank{
2. **float** rateOfInterest();
3. }
4. **class** SBI **implements** Bank{
5. **public** **float** rateOfInterest(){**return** 9.15f;}
6. }
7. **class** PNB **implements** Bank{
8. **public** **float** rateOfInterest(){**return** 9.7f;}
9. }
10. **class** TestInterface2{
11. **public** **static** **void** main(String[] args){
12. Bank b=**new** SBI();
13. System.out.println("ROI: "+b.rateOfInterest());
14. }}

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestInterface2)

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.



1. **interface** Printable{
2. **void** print();
3. }
4. **interface** Showable{
5. **void** show();
6. }
7. **class** A7 **implements** Printable,Showable{
8. **public** **void** print(){System.out.println("Hello");}
9. **public** **void** show(){System.out.println("Welcome");}
11. **public** **static** **void** main(String args[]){
12. A7 obj = **new** A7();
13. obj.print();
14. obj.show();
15. }
16. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=A7)

Output:Hello

Welcome

## **Q) 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](https://www.javatpoint.com/object-and-class-in-java) 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:

1. **interface** Printable{
2. **void** print();
3. }
4. **interface** Showable{
5. **void** print();
6. }
8. **class** TestInterface3 **implements** Printable, Showable{
9. **public** **void** print(){System.out.println("Hello");}
10. **public** **static** **void** main(String args[]){
11. TestInterface3 obj = **new** TestInterface3();
12. obj.print();
13. }
14. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestInterface3)

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.

1. **interface** Printable{
2. **void** print();
3. }
4. **interface** Showable **extends** Printable{
5. **void** show();
6. }
7. **class** TestInterface4 **implements** Showable{
8. **public** **void** print(){System.out.println("Hello");}
9. **public** **void** show(){System.out.println("Welcome");}
11. **public** **static** **void** main(String args[]){
12. TestInterface4 obj = **new** TestInterface4();
13. obj.print();
14. obj.show();
15. }
16. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestInterface4)

Output:

Hello

Welcome

## **Java 8 Default Method in Interface**

Since Java 8, we can have method body in interface. But we need to make it default method. Let's see an example:

*File: TestInterfaceDefault.java*

1. **interface** Drawable{
2. **void** draw();
3. **default** **void** msg(){System.out.println("default method");}
4. }
5. **class** Rectangle **implements** Drawable{
6. **public** **void** draw(){System.out.println("drawing rectangle");}
7. }
8. **class** TestInterfaceDefault{
9. **public** **static** **void** main(String args[]){
10. Drawable d=**new** Rectangle();
11. d.draw();
12. d.msg();
13. }}

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestInterfaceDefault)

Output:

drawing rectangle

default method

## **Java 8 Static Method in Interface**

Since Java 8, we can have static method in interface. Let's see an example:

*File: TestInterfaceStatic.java*

1. **interface** Drawable{
2. **void** draw();
3. **static** **int** cube(**int** x){**return** x\*x\*x;}
4. }
5. **class** Rectangle **implements** Drawable{
6. **public** **void** draw(){System.out.println("drawing rectangle");}
7. }
9. **class** TestInterfaceStatic{
10. **public** **static** **void** main(String args[]){
11. Drawable d=**new** Rectangle();
12. d.draw();
13. System.out.println(Drawable.cube(3));
14. }}

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestInterfaceStatic)

Output:

drawing rectangle

27

## **Q) What is marker or tagged interface?**

An interface which has no member is known as a marker or tagged interface, for example, [Serializable](https://www.javatpoint.com/serialization-in-java), Cloneable, Remote, etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

1. //How Serializable interface is written?
2. **public** **interface** Serializable{
3. }

#### **Nested Interface in Java**

Note: An interface can have another interface which is known as a nested interface. We will learn it in detail in the [nested classes](https://www.javatpoint.com/java-inner-class) chapter. For example:

1. **interface** printable{
2. **void** print();
3. **interface** MessagePrintable{
4. **void** msg();
5. }
6. }

The long question code

import java.lang.Math;

import javax.swing.\*;

import java.awt.event.\*;

public class TextFieldExample implements ActionListener{

JTextField tf1,tf2,tf3;

JButton b1;

TextFieldExample(){

JFrame f= new JFrame();

tf1=new JTextField();

tf1.setBounds(50,50,150,20);

tf2=new JTextField();

tf2.setBounds(50,100,150,20);

tf3=new JTextField();

tf3.setBounds(50,150,150,20);

tf3.setEditable(false);

b1=new JButton("Click Here For calculate");

b1.setBounds(80,200,100,40);

b1.addActionListener(this);

f.add(tf1);f.add(tf2);f.add(tf3);f.add(b1);

f.setSize(300,300);

f.setLayout(null);

f.setVisible(true);

}

public void actionPerformed(ActionEvent e) {

String s1=tf1.getText();

String s2=tf2.getText();

int a=Integer.parseInt(s1);

int b=Integer.parseInt(s2);

int c=0;

if(e.getSource()==b1){

c=(int)Math.pow(a,b);

}

String result=String.valueOf(c);

tf3.setText(result);

}

public static void main(String[] args) {

new TextFieldExample();

} }

***Question.5:***

## **What is Exception Handling?**

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

***Program:***

public class Multi {

public static void main(String[] args) {

try{

int b[]=new int[5];

b[5]=30/0;

String s=null;

System.out.println(s.length());

}

catch(ArithmeticException e)

{

System.out.println("Arithmetic Exception occurs");

}

catch(Exception e)

{

System.out.println("Parent Exception occurs");

}

System.out.println("rest of the code");

}

}

***Question.3:***

public class Banks {

private int bankid;

private String bankname;

private float amount,time,rate,interest;

public void Banks(float amount,float time,float rate)

{

this.amount=amount;

this.time=time;

this.rate=rate;

}

public void Dis()

{

interest=amount\*time\*rate/100;

System.out.println("The interes of the customer is:"+interest);

}

public void Banks(int bankid)

{

this.bankid=bankid;

}

public void Banks(String bankname)

{

this.bankname= bankname;

}

public void setBankid(int i)

{

this.bankid=i;

}

public void setBankname(String n)

{

this.bankname=n;

}

public int getBankid()

{

return bankid;

}

public String getBankname()

{

return bankname;

}

public static void main(String[] args) {

// TODO Auto-generated method stub

Banks b=new Banks();

b.setBankid(47490);

b.setBankname("Amjad");

System.out.println("The id of the customer is:"+b.getBankid());

System.out.println("The name of the customer is:"+b.getBankname());

b.Banks(1,1,1);

b.Dis();

}

}