

Core Java interview Questions

#Explain public static void main(String args[]) in Java.

main() in Java is the entry point for any Java program. It is always written as **public static void main(String[] args)**.

- **public**: Public is an access modifier, which is used to specify who can access this method. Public means that this Method will be accessible by any Class.
- **static**: It is a keyword in java which identifies it is class-based. main() is made static in Java so that it can be accessed without creating the instance of a Class. In case, main is not made static then the compiler will throw an error as **main()** is called by the JVM before any objects are made and only static methods can be directly invoked via the class.
- **void**: It is the return type of the method. Void defines the method which will not return any value.
- **main**: It is the name of the method which is searched by JVM as a starting point for an application with a particular signature only. It is the method where the main execution occurs.
- **String args[]**: It is the parameter passed to the main method.

#Why Java is platform independent?

Java is called platform independent because of its byte codes which can run on any system irrespective of its underlying operating system.

#Why Java is not 100% Object-oriented?

Java is not 100% Object-oriented because it makes use of eight primitive data types such as boolean, byte, char, int, float, double, long, short which are not objects.

#What are wrapper classes in Java?

Wrapper classes convert the Java primitives into the reference types (objects). Every primitive data type has a class dedicated to it. These are known as wrapper classes because they “wrap” the primitive data type into an object of that class. Refer to the below image which displays different primitive type, wrapper class and constructor argument.

#What are constructors in Java?

In Java, constructor refers to a block of code which is used to initialize an object. It must have the same name as that of the class. Also, it has no return type and it is automatically called when an object is created.

There are two types of constructors:

1. **Default Constructor:** In Java, a default constructor is the one which does not take any inputs. In other words, default constructors are the no argument constructors which will be created by default in case you no other constructor is defined by the user. Its main purpose is to initialize the instance variables with the default values. Also, it is majorly used for object creation.
2. **Parameterized Constructor:** The parameterized constructor in Java, is the constructor which is capable of initializing the instance variables with the provided values. In other words, the constructors which take the arguments are called parameterized constructors.

What is the difference between Array list and vector in Java?

ArrayList	Vector
Array List is not synchronized.	Vector is synchronized.
Array List is fast as it's non-synchronized.	Vector is slow as it is thread safe.
If an element is inserted into the Array List, it increases its Array size by 50%.	Vector defaults to doubling size of its array.
Array List does not define the increment size.	Vector defines the increment size.
Array List can only use Iterator for traversing an Array List.	Vector can use both Enumeration and Iterator for traversing.

#What is the difference between equals() and == in Java?

In general, both equals() and “==” operator in Java are used to compare objects to check equality but here are some of the differences between the two:

1. The main difference between the .equals() method and == operator is that one is a method and the other is the operator.
2. We can use == operators for reference comparison (**address comparison**) and .equals() method for **content comparison**. In simple words, == checks if both objects point to the same memory location whereas .equals() evaluates to the comparison of values in the objects.

```
// Java program to understand
// the concept of == operator
public class Test {
    public static void main(String[] args)
    {
        String s1 = "HELLO";
        String s2 = "HELLO";
        String s3 = new String("HELLO");
        System.out.println(s1 == s2); // true
        System.out.println(s1 == s3); // false
        System.out.println(s1.equals(s2)); // true
        System.out.println(s1.equals(s3)); // true
    }
}
```

Explanation: Here we are creating two objects namely s1 and s2.

- Both s1 and s2 refer to different objects.
- When we use the == operator for s1 and s2 comparison then the result is true as both have the same addresses in the string constant pool.
- Using equals, the result is true because it's only comparing the values given in s1 and s2.

#What are the differences between Heap and Stack Memory in Java?

Stack memory only contains local primitive and reference variables to objects in heap space.

Whenever an object is created, it's always stored in the Heap space.

#Why pointers are not used in Java?

Java doesn't use pointers because they are unsafe and increases the complexity of the program. Since, Java is known for its simplicity of code, adding the concept of pointers will be contradicting. Moreover, since JVM is responsible for implicit memory allocation,

thus in order to avoid direct access to memory by the user, pointers are discouraged in Java.

#What are access modifiers in Java?

In Java, access modifiers are special keywords which are used to restrict the access of a class, constructor, data member and method in another class. Java supports four types of access modifiers:

1. *Default*
2. *Private*
3. *Protected*
4. *Public*

Modifier	Default	Private	Protected	Public
<i>Same class</i>	YES	YES	YES	YES
<i>Same Package subclass</i>	YES	NO	YES	YES
<i>Same Package non-subclass</i>	YES	NO	YES	YES
<i>Different package subclass</i>	NO	NO	YES	YES
<i>Different package non-subclass</i>	NO	NO	NO	YES

#Define a Java Class.

A class in Java is a blueprint which includes all your data. A class contains fields (variables) and methods to describe the behavior of an object. Let's have a look at the syntax of a class.

#What is an object in Java and how is it created?

An object is a real-world entity that has a state and behavior. An object has three characteristics:

1. State
2. Behavior
3. Identity

An object is created using the 'new' keyword. For example:

```
ClassName obj = new ClassName();
```

#What is Object Oriented Programming?

Object-oriented programming or popularly known as OOPs is a programming model or approach where the programs are organized around objects rather than logic and functions.

#What are the main concepts of OOPs in Java?

Object-Oriented Programming or OOPs is a programming style that is associated with concepts like:

1. **Inheritance:** Inheritance is a process where one class acquires the properties of another.

Example:

```
class Doctor {  
    void Doctor_Details() {  
        System.out.println("Doctor Details...");  
    }  
}  
  
class Surgeon extends Doctor {  
    void Surgeon_Details() {  
        System.out.println("Surgen Detail...");  
    }  
}  
  
public class Hospital {  
    public static void main(String args[]) {  
        Surgeon s = new Surgeon();  
        s.Doctor_Details();  
        s.Surgeon_Details();  
    }  
}
```

2. **Encapsulation:** Encapsulation in Java is a mechanism of wrapping up the data and code together as a single unit.

What is encapsulation in Java?

Encapsulation is a mechanism where you bind your data(variables) and code(methods) together as a single unit. Here, the data is hidden from the outer world and can be accessed only via current class methods. This helps in protecting the data from any unnecessary modification. We can achieve encapsulation in Java by:

- Declaring the variables of a class as private.
- Providing public setter and getter methods to modify and view the values of the variables.

```
// Java program to demonstrate encapsulation
class Encapsulate {
    // private variables declared
    // these can only be accessed by
    // public methods of class
    private String geekName;
    private int geekRoll;
    private int geekAge;

    // get method for age to access
    // private variable geekAge
    public int getAge() { return geekAge; }

    // get method for name to access
    // private variable geekName
    public String getName() { return geekName; }

    // get method for roll to access
    // private variable geekRoll
    public int getRoll() { return geekRoll; }

    // set method for age to access
    // private variable geekAge
    public void setAge(int newAge) { geekAge = newAge; }

    // set method for name to access
    // private variable geekName
    public void setName(String newName)
    {
        geekName = newName;
    }

    // set method for roll to access
    // private variable geekRoll
    public void setRoll(int newRoll) { geekRoll = newRoll; }
}

public class TestEncapsulation {
```

```

    public static void main(String[] args)
    {
        Encapsulate obj = new Encapsulate();

        // setting values of the variables
        obj.setName("Harsh");
        obj.setAge(19);
        obj.setRoll(51);

        // Displaying values of the variables
        System.out.println("Geek's name: " + obj.getName());
        System.out.println("Geek's age: " + obj.getAge());
        System.out.println("Geek's roll: " + obj.getRoll());

        // Direct access of geekRoll is not possible
        // due to encapsulation
        // System.out.println("Geek's roll: " +
        // obj.geekName);
    }
}

```

Another Example:

```

//A Account class which is a fully encapsulated class.
//It has a private data member and getter and setter methods.
class Account {
    //private data members
    private long acc_no;
    private String name,email;
    private float amount;
    //public getter and setter methods
    public long getAcc_no() {
        return acc_no;
    }
    public void setAcc_no(long acc_no) {
        this.acc_no = acc_no;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getEmail() {
        return email;
    }
}

```

```

public void setEmail(String email) {
    this.email = email;
}
public float getAmount() {
    return amount;
}
public void setAmount(float amount) {
    this.amount = amount;
}
}

//A Java class to test the encapsulated class Account.
public class TestEncapsulation {
    public static void main(String[] args) {
        //creating instance of Account class
        Account acc=new Account();
        //setting values through setter methods
        acc.setAcc_no(7560504000L);
        acc.setName("Sonoo Jaiswal");
        acc.setEmail("sonoojaiswal@javatpoint.com");
        acc.setAmount(500000f);
        //getting values through getter methods
        System.out.println(acc.getAcc_no()+" "+acc.getName()+" "+acc.getEmail()+" "+acc.getAmount());
    }
}

```

3. **Abstraction:** Abstraction is the methodology of hiding the implementation details from the user and only providing the functionality to the users.

Abstraction can be achieved in two ways:

- **Abstract Classes** (0-100% of abstraction can be achieved)
- **Interfaces** (100% of abstraction can be achieved)

What do you mean by an interface in Java?

An interface in Java is a blueprint of a class or you can say it is a collection of abstract methods and static constants. In an interface, each method is public and abstract but it does not contain any constructor. Thus, interface basically is a group of related methods with empty bodies.

Example:

```

public interface Animal {

```



```

public void eat();
public void sleep();
public void run();
}

```

What is the difference between abstract classes and interfaces?

Abstract Class	Interfaces
An abstract class can provide complete, default code and/or just the details that have to be overridden	An interface cannot provide any code at all, just the signature
In the case of an abstract class, a class may extend only one abstract class	A Class may implement several interfaces
An abstract class can have non-abstract methods	All methods of an Interface are abstract
An abstract class can have instance variables	An Interface cannot have instance variables
An abstract class can have any visibility: public, private, protected	An Interface visibility must be public (or) none
If we add a new method to an abstract class then we have the option of providing default implementation and therefore all the existing code might work properly	If we add a new method to an Interface then we have to track down all the implementations of the interface and define implementation for the new method
An abstract class can contain constructors	An Interface cannot contain constructors
Abstract classes are fast	Interfaces are slow as it requires extra indirection to find the corresponding method in the actual class

What is inheritance in Java?

Inheritance in Java is the concept where the properties of one class can be inherited by the other. It helps to reuse the code and establish a relationship between different classes. Inheritance is performed between two types of classes:

1. Parent class (Super or Base class)

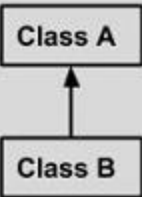
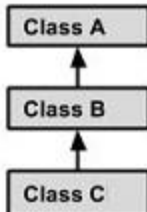
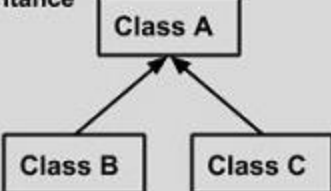
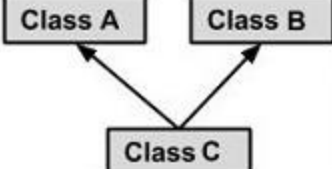
2. Child class (Subclass or Derived class)

A class which inherits the properties is known as Child Class whereas a class whose properties are inherited is known as Parent class.

What are the different types of inheritance in Java?

Java supports four types of inheritance which are:

1. **Single Inheritance:** In single inheritance, one class inherits the properties of another i.e there will be only one parent as well as one child class.
2. **Multilevel Inheritance:** When a class is derived from a class which is also derived from another class, i.e. a class having more than one parent class but at different levels, such type of inheritance is called Multilevel Inheritance.
3. **Hierarchical Inheritance:** When a class has more than one child classes (subclasses) or in other words, more than one child classes have the same parent class, then such kind of inheritance is known as hierarchical.
4. **Hybrid Inheritance:** Hybrid inheritance is a combination of two *or more types* of inheritance.

Single Inheritance  <pre> graph BT B[Class B] --> A[Class A] </pre>	<pre> public class A { } public class B extends A { } </pre>
Multi Level Inheritance  <pre> graph BT C[Class C] --> B[Class B] B --> A[Class A] </pre>	<pre> public class A {} public class B extends A {.....} public class C extends B {.....} </pre>
Hierarchical Inheritance  <pre> graph BT B[Class B] --> A[Class A] C[Class C] --> A </pre>	<pre> public class A {} public class B extends A {.....} public class C extends A {.....} </pre>
Multiple Inheritance  <pre> graph BT C[Class C] --> A[Class A] C --> B[Class B] </pre>	<pre> public class A {} public class B {.....} public class C extends A,B { } // Java does not support multiple Inheritance </pre>

```

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

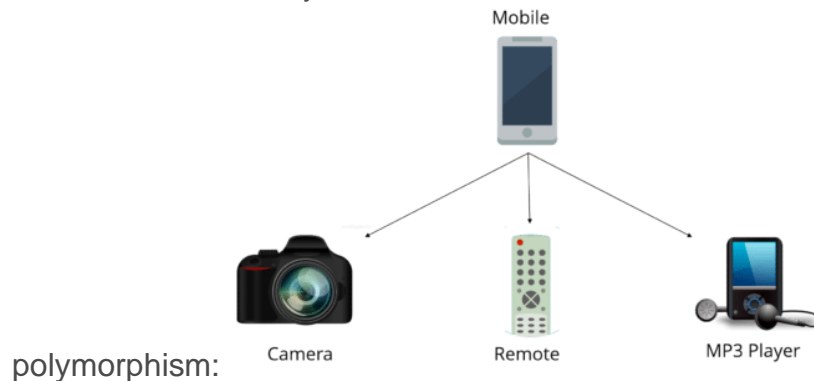
class TestBank{
    public static void main(String args[]){
        Bank b;
        b=new SBI();
        System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
        b=new PNB();
        System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
    }
}

```

4. **Polymorphism:** Polymorphism is the ability of a variable, function or object to take multiple forms.

What is Polymorphism?

Polymorphism is briefly described as “one interface, many implementations”. Polymorphism is a characteristic of being able to assign a different meaning or usage to something in different contexts – specifically, to allow an entity such as a variable, a function, or an object to have more than one form. There are two types of



1. Compile time polymorphism
2. Run time polymorphism

Compile time polymorphism is method overloading whereas Runtime time polymorphism is done using inheritance and interface.

Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters or both. Overloading is related to compile-time (or static) polymorphism.

Method Overloading :

- In Method Overloading, Methods of the same class shares the same name but each method must have a different number of parameters or parameters having different types and order.
- Method Overloading is to “add” or “extend” more to the method’s behavior.
- It is a compile-time polymorphism.
- The methods must have a different signature.
- It may or may not need inheritance in Method Overloading.

// Java program to demonstrate working of method

// overloading in Java.

```
public class Sum {  
  
    // Overloaded sum(). This sum takes two int parameters  
    public int sum(int x, int y)  
    {  
        return (x + y);  
    }  
  
    // Overloaded sum(). This sum takes three int parameters  
    public int sum(int x, int y, int z)  
    {  
        return (x + y + z);  
    }  
  
    // Overloaded sum(). This sum takes two double parameters  
    public double sum(double x, double y)  
    {  
        return (x + y);  
    }  
  
    // Driver code  
    public static void main(String args[])  
    {  
        Sum s = new Sum();  
        System.out.println(s.sum(10, 20));  
        System.out.println(s.sum(10, 20, 30));  
        System.out.println(s.sum(10.5, 20.5));  
    }  
}
```

What is runtime polymorphism or dynamic method dispatch?

In Java, runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this process, an overridden method is called through the reference variable of a superclass. Let's take a look at the example below to understand it better.

Method Overriding:

- In Method Overriding, the subclass has the same method with the same name and exactly the same number and type of parameters and same return type as a superclass.
- Method Overriding is to "Change" existing behavior of the method.
- It is a run time polymorphism.
- The methods must have the same signature.
- It always requires inheritance in Method Overriding.

```

class Car {
void run()
{
System.out.println("car is running");
}
}
class Audi extends Car {
void run()
{
System.out.println("Audi is running safely with 100km");
}
public static void main(String args[])
{
Car b= new Audi(); //upcasting
b.run();
}
}

```

#What is the difference between a local variable and an instance variable?

In Java, a **local variable** is typically used inside a method, constructor, or a **block** and has only local scope. Thus, this variable can be used only within the scope of a block. The best benefit of having a local variable is that other methods in the class won't be even aware of that variable.

Whereas, an **instance variable** in Java, is a variable which is bounded to its object itself. These variables are declared within a **class**, but outside a method. Every object of that class will create it's own copy of the variable while using it. Thus, any changes made to the variable won't reflect in any other instances of that class and will be bound to that particular instance only.

#Differentiate between the constructors and methods in Java?

Methods	Constructors
1. Used to represent the behavior of an object	1. Used to initialize the state of an object

2. Must have a return type

2. Do not have any return type

3. Needs to be invoked explicitly

3. Is invoked implicitly

4. No default method is provided by the compiler

4. A default constructor is provided by the compiler if the class has none

5. Method name may or may not be same as class name

5. Constructor name must always be the same as the class name

#What is final keyword in Java?

final is a special keyword in Java that is used as a non-access modifier. A final variable can be used in different contexts such as:

final variable

When the final keyword is used with a variable then its value can't be changed once assigned. In case the no value has been assigned to the final variable then using only the class constructor a value can be assigned to it.

final method

When a method is declared final then it can't be overridden by the inheriting class.

final class

When a class is declared as final in Java, it can't be extended by any subclass class but it can extend other class.

#What is the difference between break and continue statements?

break

continue

- | | |
|---|--|
| 1. Can be used in switch and loop (for, while, do while) statements | 1. Can be only used with loop statements |
| 2. It causes the switch or loop statements to terminate the moment it is executed | 2. It doesn't terminate the loop but causes the loop to jump to the next iteration |
| 3. It terminates the innermost enclosing loop or switch immediately | 3. A continue within a loop nested with a switch will cause the next loop iteration to execute |

#What is an infinite loop in Java? Explain with an example.

An infinite loop is an instruction sequence in Java that loops endlessly when a functional exit isn't met.

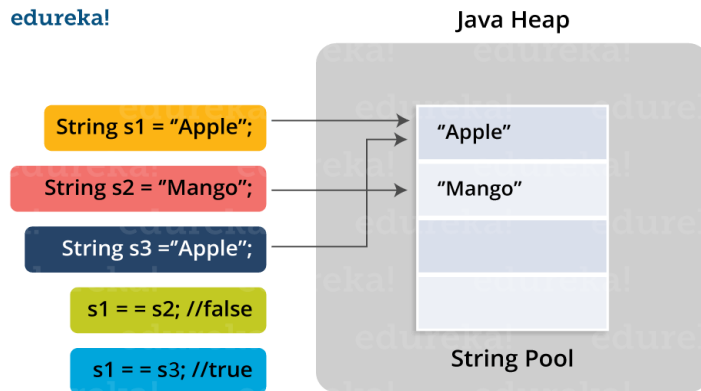
#What is the difference between this() and super() in Java?

In Java, super() and this(), both are special keywords that are used to call the constructor.

this()	super()
1. this() represents the current instance of a class	1. super() represents the current instance of a parent/base class
2. Used to call the default constructor of the same class	2. Used to call the default constructor of the parent/base class
3. Used to access methods of the current class	3. Used to access methods of the base class
4. Used for pointing the current class instance	4. Used for pointing the superclass instance
5. Must be the first line of a block	5. Must be the first line of a block

#What is Java String Pool?

Java String pool refers to a collection of Strings which are stored in heap memory. In this, whenever a new object is created, String pool first checks whether the object is already present in the pool or not. If it is present, then the same reference is returned to the variable else new object will be created in the String pool and the respective reference will be returned.



#Differentiate between static and non-static methods in Java.

Static Method	Non-Static Method
1. The <i>static</i> keyword must be used before the method name	1. No need to use the <i>static</i> keyword before the method name
2. It is called using the class (className.methodName)	2. It can be called like any general method
3. They can't access any non-static instance variables or methods	3. It can access any static method and any static variable without creating an instance of the class

#What is constructor chaining in Java?

In Java, constructor chaining is the process of calling one constructor from another with respect to the current object. Constructor chaining is possible only through legacy where a subclass constructor is responsible for invoking the superclass' constructor first. There could be any number of classes in the constructor chain. Constructor chaining can be achieved in two ways:

1. Within the same class using this()

2. From base class using super()

Difference between String, StringBuilder, and StringBuffer.

Factor	String	StringBuilder	StringBuffer
<i>Storage Area</i>	Constant String Pool	Heap Area	Heap Area
<i>Mutability</i>	Immutable	Mutable	Mutable
<i>Thread Safety</i>	Yes	No	Yes
<i>Performance</i>	Fast	More efficient	Less efficient

#What is a classloader in Java?

The **Java ClassLoader** is a subset of JVM (Java Virtual Machine) that is responsible for loading the class files. Whenever a Java program is executed it is first loaded by the classloader. Java provides three built-in classloaders:

1. Bootstrap ClassLoader
2. Extension ClassLoader
3. System/Application ClassLoader

#What is the difference between an array and an array list?

Array	ArrayList
Cannot contain values of different data types	Can contain values of different data types.
Size must be defined at the time of declaration	Size can be dynamically changed
Need to specify the index in order to add data	No need to specify the index
Arrays are not type parameterized	Arraylists are type
Arrays can contain primitive data types as	Arraylists can contain only objects, no primitive

well as objects

data types are allowed

#What is a Map in Java?

In Java, Map is an interface of Util package which maps unique keys to values. The Map interface is not a subset of the main Collection interface and thus it behaves little different from the other collection types. Below are a few of the characteristics of Map interface:

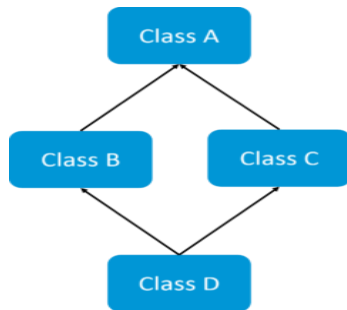
1. Map doesn't contain duplicate keys.
2. Each key can map at max one value.

Can you override a private or static method in Java?

You cannot override a private or static method in Java. If you create a similar method with the same return type and same method arguments in child class then it will hide the superclass method; this is known as method hiding. Similarly, you cannot override a private method in subclass because it's not accessible there. What you can do is create another private method with the same name in the child class. Let's take a look at the example below to understand it better.

```
class Base {  
    private static void display() {  
        System.out.println("Static or class method from Base");  
    }  
    public void print() {  
        System.out.println("Non-static or instance method from Base");  
    }  
    class Derived extends Base {  
        private static void display() {  
            System.out.println("Static or class method from Derived");  
        }  
        public void print() {  
            System.out.println("Non-static or instance method from Derived");  
        }  
    }  
    public class test {  
        public static void main(String args[])  
        {  
            Base obj= new Derived();  
            obj1.display();  
            obj1.print();  
        }  
    }  
}
```

What is multiple inheritance? Is it supported by Java?



If a child class inherits the property from multiple classes is known as multiple inheritance. Java does not allow to extend multiple classes.

The problem with multiple inheritance is that if multiple parent classes have the same method name, then at runtime it becomes difficult for the compiler to decide which method to execute from the child class.

Therefore, Java doesn't support multiple inheritance. The problem is commonly referred to as Diamond Problem.

In case you are facing any challenges with these java interview questions, please comment on your problems in the section below.

In simple words, **Instance refers to the copy of the object at a particular time whereas object refers to the memory address of the class.**

An **object that is created using** a class is said to be an instance of that class. We will sometimes say that the object belongs to the class. The variables that the object contains are called instance variables. The methods (that is, subroutines) that the object contains are called instance methods.