Core Java Interview Questions For Freshers

jHarsh

For the latest Java Interview Questions Refer to the Following Article – Java Interview Questions – Fresher and Experienced (2023)

Java is one of the most popular and widely used programming languages and a platform that was developed by James Gosling in the year 1982. It is based on the concept of object-oriented Programming. A platform is an environment in that develops and runs programs written in any programming language. Java is a high-level, object-oriented, secure, robust, platform-independent, multithreaded, and portable programming language. It is mainly used for programming, window, web-based, enterprise, and mobile applications.

Java is the most used language in top companies such as Uber, Airbnb, Google, Netflix, Instagram, Spotify, Amazon, and many more because of its features and performance. To get into these companies and other software companies, you need to master some important Core Java interview questions to crack their Java Online Assessment round and Java Technical interview.

This Core Java programming interview questions article is written under the guidance of the masters of Java and by getting ideas through the experience of students' recent Java interviews.

Core Java Interview Questions For Freshers

Q1. Explain JVM, JRE, and JDK.

JVM (Java Virtual Machine): JVM(Java Virtual Machine) acts as a run-time engine to run Java applications. JVM is the one that calls the main method present in Java code. JVM is a part of JRE(Java Runtime Environment).

JRE (Java Runtime Environment): JRE refers to a runtime environment in which Java bytecode can be executed. It implements the JVM (Java Virtual Machine) and provides all the class libraries and other support files that JVM uses at runtime. So JRE is a software package that contains what is required to run a Java program. It's an implementation of the JVM which physically exists.

JDK(Java Development Kit): It is the tool necessary to compile, document, and package Java programs. The JDK completely includes JRE which contains tools for Java programmers. The Java Development Kit is provided free of charge. Along with JRE, it includes an interpreter/loader, a compiler (javac), an archiver (jar), a documentation generator (Javadoc), and other tools needed in Java development. In short, it contains JRE + development tools.

Q2. Explain public static void main(String args[]).

- **Public:** Public is an access modifier. Public means that this Method will be accessible by any Class.
- **static:** It is a keyword in Java that identifies it as class-based i.e it can be accessed without creating the instance of a Class. Since we want the main method to be executed without any instance also, we use static.
- **Void:** It is the return type of the method. Void defines the method which will not return any value.
- **main:** This is the first method executed by JVM. The signature of the method must be the same.

Q3. Why Java is platform independent?

Platform independence practically means "write once run anywhere". Java is called so because of its **byte codes** which can run on any system irrespective of its underlying operating system.

Q4. Why is Java not purely Object-oriented?

Java is not considered pure Object-oriented because it supports primitive data types such as boolean, byte, char, int, float, double, long, and short.

Q5. Define class and object. Explain them with an example using Java.

- **Class:** A class is a user-defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order.
- **Superclass(if any):** The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
- **Interfaces:** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
- **Object:** It is a basic unit of Object Oriented Programming and represents real-life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :
- **State:** It is represented by attributes of an object. It also reflects the properties of an object.
- **Behavior:** It is represented by the methods of an object. It also reflects the response of an object to other objects.
- **Identity:** It gives a unique name to an object and enables one object to interact with other objects.

For Example, an Employee is an example of a class. A specific employee with unique identification is an example of an object.

```
class Employee
{
   // instance variables declaration
   // Methods definition
}
```

An object of an employee is a specific employee

```
Employee empObj = new Employee();
```

One of the objects of Employee is referred to by 'empObj'

Q6. What is a method? Provide several signatures of the methods.

- A Java method is a set of statements to perform a task. A method is placed in a class.
- Signatures of methods: The method's name, return type, and the number of parameters comprise the method signature.
- A method can have the following elements in its signature:
  - Access specifier – public, private, protected, etc. (Not mandatory)
  - Access modifier – static, synchronized, etc. (Not mandatory)
  - Return type – void, int, String, etc. (Mandatory)
  - Method name – show() (Mandatory)
  - With or without parameters – (int number, String name); (parenthesis are mandatory)

Example:

Java

```
class Test {
    void fun1() {}
    public double fun2(double x) {}
    public static void fun3() {}
    public static void fun4(String x) {}
}
```

Q7. Explain the difference between an instance variable and a class variable.

An instance variable is a variable that has one copy per object/instance. That means every object will have one copy of it. A class variable is a variable that has one copy per class. The class variables will not have a copy in the object.

Example:

Java

```
class Employee {
    int empNo;
    String empName, department;
    double salary;
    static int officePhone;
}
```

An object referred to by empObj1 is created by using the following:

Employee empObj1 = new Employee();

The objects referred by instance variables empObj1 and empObj2 have separate copies empNo, empName, department, and salary. However, the officePhone belongs to the

class(Class Variable) and can be accessed as Employee.officePhone.

Q8. Which class is the superclass of all classes?

java.lang.Object is the root class for all the Java classes and we don't need to extend it.

Q9. What are constructors in Java?

In Java, a constructor refers to a block of code that is used to initialize an object. It must have the same name as that of the class. Also, it has no return type and is automatically called when an object is created.
If a class does not explicitly declare any, the Java compiler automatically provides a no-argument constructor, also called the default constructor.
This default constructor calls the class parent's no-argument constructor (as it contains only one statement i.e. super();), or the Object class constructor if the class has no other parent (as the Object class is a parent of all classes either directly or indirectly).

There are two types of constructors:

1. Default constructor
2. Parameterized constructor

Q10. What are the different ways to create objects in Java?

There are many different ways to create objects in Java.

1. Using new keyword
2. Using new instance
3. Using clone() method
4. Using deserialization
5. Using newInstance() method of Constructor class

Please refer to this article – 5 Different ways to create objects in Java

Q11. What's the purpose of Static methods and static variables?

When there is a requirement to share a method or a variable between multiple objects of a class instead of creating separate copies for each object, we use static keywords to make a method or variable shared for all objects.

Static variable: Static variables are also known as Class variables.

- These variables are declared similarly to instance variables, the difference is that static variables are declared using the static keyword within a class outside any method constructor or block.

- Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create.
- Static variables are created at the start of program execution and destroyed automatically when execution ends.
- To access static variables, we need not create an object of that class.

Static methods: A static method can be accessed without creating objects. Just by using the Class name, the method can be accessed. The static method can only access static variables, not local or global non-static variables.

For Example:

Java

```
class StaticMethod {
    public static void printMe()
    {
        System.out.println("Static Method access directly by class name!");
    }
}
class MainClass {
    public static void main(String args[])
    {
        StaticMethod.printMe();
    }
}
```

Output

```
Static Method access directly by class name!
```

Q12. Why can static methods not access non-static variables or methods?

A static method cannot access non-static variables or methods because static methods can be accessed without instantiating the class, so if the class is not instantiated the variables are not initialized and thus cannot be accessed from a static method.

Q13. What is a static class?

A class can be said to be a static class if all the variables and methods of the class are static and the constructor is private. Making the constructor private will prevent the class to be instantiated. So the only possibility to access is using the Class name only.

Q14. Explain How many types of variables are in Java?

There are three types of variables in Java:

1. Local Variables
2. Instance Variables
3. Static Variables

Local Variables: A variable defined within a block or method or constructor is called a local variable. These variables are created when the block is entered or the function is called and destroyed after exiting from the block or when the call returns from the function. The scope of these variables exists only within the block in which the variable is declared. i.e. we can access these variables only within that block.

Java

```
// Java program to demonstrate local variables
public class LocalVariable
{
    public void getLocalVarValue()
    {
        // local variable age
        int localVar = 0;
        localVar = localVar + 11;
        System.out.println("value of local variable" +
                                " is: " + localVar);
    }
    public static void main(String args[])
    {
        LocalVariable obj = new LocalVariable();
        obj.getLocalVarValue();
    }
}
```

Output:

```
value of local variable is: 11
```

In the above program, the variable localVar is the local variable to the function getLocalVarValue(). If we use the variable localVar outside getLocalVarValue() function, the compiler will produce an error as “Cannot find the symbol localVar”.

Instance Variables: Instance variables are non-static variables and are declared in a class outside any method, constructor, or block. As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed. Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier then the default access specifier will be used.

Java

```
// Java program to demonstrate instance variables
public class InstanceVariable {
```

```
    int instanceVarId;
    String instanceVarName;
    public static void main(String args[])
    {
        InstanceVariable obj = new InstanceVariable();
        obj.instanceVarId = 0001;
        obj.instanceVarName = "InstanceVariable1";
        System.out.println("Displaying first Object:");
        System.out.println("instanceVarId==" + obj.instanceVarId);
        System.out.println("instanceVarName==" + obj.instanceVarName);

        InstanceVariable obj1 = new InstanceVariable();
        obj1.instanceVarId = 0002;
        obj1.instanceVarName = "InstanceVariable2";
        System.out.println("Displaying Second Object:");
        System.out.println("instanceVarId==" + obj1.instanceVarId);
        System.out.println("instanceVarName==" + obj1.instanceVarName);
    }
}
```

Output:

```
Displaying first Object:
instanceVarId==1
instanceVarName==InstanceVariable1
Displaying Second Object:
instanceVarId==2
instanceVarName==InstanceVariable2
```

In the above program the variables i.e. instanceVarId, instanceVarName are instance variables. In case we have multiple objects as in the above program, each object will have its own copies of instance variables. It is clear from the above output that each object will have its own copy of the instance variable.

Static variable: Static variables are also known as Class variables.

- These variables are declared similarly to instance variables, the difference is that static variables are declared using the static keyword within a class outside any method constructor or block.
- Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create.
- Static variables are created at the start of program execution and destroyed automatically when execution ends.

To access static variables, we need not create an object of that class, we can simply access the variable as:

Java

```
// Java program to demonstrate static variables
public class StaticVar {
    private static int count = 0;
    private int nonStaticCount = 0;

    public void incrementCounter()
    {
        count++;
        nonStaticCount++;
    }
    public static int getStaticCount()
    {
        return count;
    }
    public int getNonStaticCount()
    {
        return nonStaticCount;
    }
    public static void main(String args[])
    {
        StaticVar stVarObj1 = new StaticVar();
        StaticVar stVarObj2 = new StaticVar();
        stVarObj1.incrementCounter();
        stVarObj2.incrementCounter();
        System.out.println("Static count for stVarObj1: " +
                           stVarObj1.getStaticCount());
        System.out.println("NonStatic count for stVarObj1: " +
                             stVarObj1.getNonStaticCount());
        System.out.println("Static count for stVarObj2: " +
                             stVarObj2.getStaticCount());
        System.out.println("NonStatic count for stVarObj2: " +
                             stVarObj2.getNonStaticCount());
    }
}
```

Output:

```
Static count for stVarObj1: 2
NonStatic count for stVarObj1: 1
Static count for stVarObj2: 2
NonStatic count for stVarObj2: 1
```

In the above program, stVarObj1 and stVarObj2 share the same instance of static variable count hence if the value is incremented by one object, the incremented value will be reflected for stVarObj1 and stVarObj2.

Q14. What is the difference between the Set and List interface?

Set and List are both child interfaces of the Collection interface. There are following two main differences between them

1. List can hold duplicate values but Set doesn't allow this.
2. In List interface data is present in the order you inserted but in the case of Set, the insertion order is not preserved.

Q15. What is the difference between StringBuffer and String?

The String class is an Immutable class, i.e. you can not modify its content once created. While StringBuffer is a mutable class, which means you can change its content later. Whenever we alter the content of a String object, it creates a new string and refers to that, it does not modify the existing one. This is the reason that the performance with StringBuffer is better than with String.

Please see String vs StringBuilder vs StringBuffer for more details.

Q16. When is the super keyword used?

The super keyword is used to refer to:

- immediate parent class constructor,
- immediate parent class variable,
- immediate parent class method.

Refer to super for details.

Q17. Differences between HashMap and HashTable in Java.

- HashMap is non-synchronized. It is not thread-safe and can't be shared between many threads without proper synchronization code whereas Hashtable is synchronized. It is thread-safe and can be shared with many threads.
- HashMap allows one null key and multiple null values whereas Hashtable doesn't allow any null key or value.
- HashMap is generally preferred over HashTable if thread synchronization is not needed.

Refer to HashMap and HashTable for more details.

Q18. What happens if you remove the static modifier from the main method?

Program compiles successfully. But at runtime throws an error "NoSuchMethodError".

Q19. Can we overload the main() method?

The main method in Java is no extra-terrestrial method. Apart from the fact that main() is just like any other method & can be overloaded similarly, JVM always looks for the method signature to launch the program.

- The normal main method acts as an entry point for the JVM to start the execution of the program.
- We can overload the main method in Java. But the program doesn't execute the overloaded main method when we run your program, we need to call the overloaded main method from the actual main method only.

Q20. What are wrapper classes in Java?

Wrapper classes convert the Java primitives into 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.

Q21. Why pointers are not used in Java?

Java doesn't use pointers because they are unsafe and increase 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 to avoid direct access to memory by the user, pointers are discouraged in Java.

Q22. What is the meaning of Collections in Java?

The collection is a framework that is designed to store the objects and manipulate the design to store the objects.

Collections are used to perform the following operations:

- Searching
- Sorting
- Manipulation
- Insertion
- Deletion

A group of objects is known as a collection. All the classes and interfaces for collecting are available in the Java util package.

Q23. What is Synchronization?

Synchronization makes only one thread access a block of code at a time. If multiple threads access the block of code, then there is a chance for inaccurate results at the end. To avoid this issue, we can provide synchronization for the sensitive block of codes.

The **synchronized** keyword means that a thread needs a key to access the synchronized code.

Every Java object has a lock. A lock has only one key. A thread can access a synchronized method only if the thread can get the key to the objects to lock. Locks are per object.

Refer to Synchronization for more details.

Q24. What is the disadvantage of Synchronization?

Synchronization is not recommended to implement all the methods. Because if one thread accesses the synchronized code then the next thread should have to wait. So it makes a slow performance on the other end.

Q25. Which class is the superclass for all the classes?

The object class is the superclass of all other classes in Java.

Last Updated : 15 Jun, 2023

77

Similar Reads

1. Difference between Core Java and Advanced Java
2. Difference between Java and Core Java
3. Best Way To Start Learning Core Java – A Complete Roadmap
4. Spring Core Annotations
5. JSTL Core Tags
6. Java Interview Questions on Constructors
7. Commonly Asked Java Programming Interview Questions | Set 1
8. Commonly Asked Java Programming Interview Questions | Set 2
9. Top 20 Java Multithreading Interview Questions & Answers
10. Top 50 Java Collections Interview Questions and Answers

Related Tutorials

1. Spring MVC Tutorial
2. Spring Tutorial