Top Java Interview Questions and Answers

We also recommend you brush up on your Java skills with this Java Cheat Sheet before starting your Java interview preparation.

We have divided these interview questions into several sections, including one for programming Java interview questions.

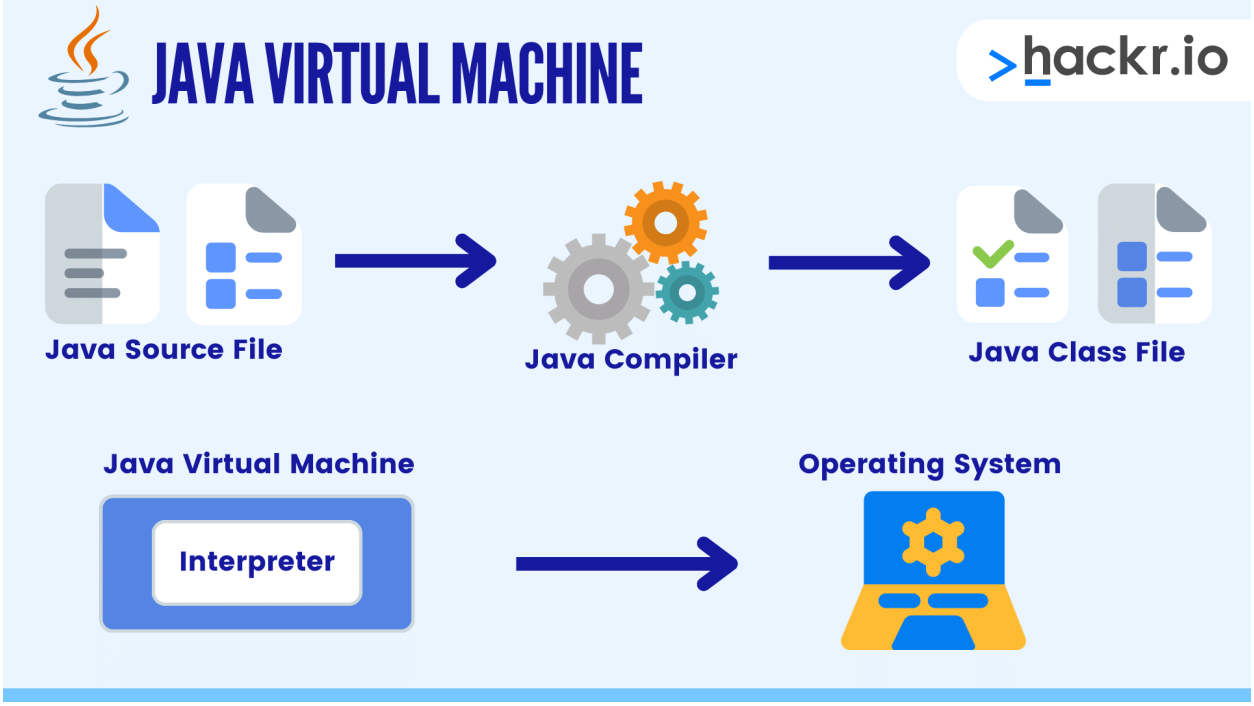
Basic Java Interview Questions

1. What is Java?

Java is an object-oriented, high-level, general-purpose programming language originally designed by James Gosling and further developed by the Oracle Corporation. It is one of the most popular programming languages in the world.

Learn more about Java here.

2. What is the Java Virtual Machine?



JVM is a program that interprets the intermediate Java byte code and generates the desired output. It is because of bytecode and JVM that programs written in Java are highly portable.

3. What are the features of Java?

The following are the various features of the Java programming language:

- **High Performance:** Using a JIT (Just-In-Time) compiler allows high performance in Java. The JIT compiler converts the Java bytecode into machine language code, which then gets executed by the JVM.
- **Multi-threading:** A thread is a flow of execution. The JVM creates a thread which is called the main thread. Java allows the creation of several threads using either extending the thread class or implementing the Runnable interface.
- **OOPs Concepts:** Java follows various OOPs concepts, namely abstraction, encapsulation, inheritance, object-oriented, and polymorphism

- **Platform Independency:** Java makes use of the Java Virtual Machine or JVM, which allows a single Java program to operate on multiple platforms without any modifications.

You may want to check out detailed explanations of Java features here.

4. How does Java enable high performance?

In Just-in-Time compilation, the required code is executed at run time. Typically, it involves translating bytecode into machine code and then executing it directly. It allows for high performance. The JIT compiler is enabled by default in Java and gets activated as soon as a method is called.

It then compiles the bytecode of the Java method into native machine code.

After that, the JVM calls the compiled code directly instead of interpreting it.

5. What are the differences between JVM, JRE, and JDK?

Parameters	JVM	JRE	JDK
Full-Form	Java Virtual Machine	Java Runtime Environment	Java Development Kit
Purpose	It provides a runtime environment to execute Java bytecode.	It is a set of software tools used for developing Java applications.	It is a software development environment used to develop Java applications.
Existence	It is a runtime instance created when we run a Java class.	It exists physically.	It exists physically.

Implementation	Its implementation is known as JRE	It is the implementation of JVM	It is an implementation of any one of the below given Java Platforms released by Oracle Corporation: Standard Edition Java Platform Enterprise Edition Java Platform Micro Edition Java Platform

6. What is the JIT compiler?

JIT compiler runs after the program is executed and compiles the code into a faster form, hosting the CPU's native instructing set. JIT can access dynamic runtime information, whereas a standard compiler doesn't and can make better optimizations like inlining functions that are used frequently.

7. What are Java IDEs?

A Java IDE is a software that allows Java developers to easily write as well as debug Java programs. It is basically a collection of various programming tools, accessible via a single interface, and several helpful features, such as code completion and syntax highlighting.

Codenvy, Eclipse, and NetBeans are some of the most popular Java IDEs.

8. Java is a platform-independent language. Why?

Java does not depend on any particular hardware or software because it is compiled by the compiler and then converted into byte code. Byte code is platform-independent and

can run on multiple systems. The only requirement is that Java needs a runtime environment, i.e., JRE, which is a set of tools used for developing Java applications.

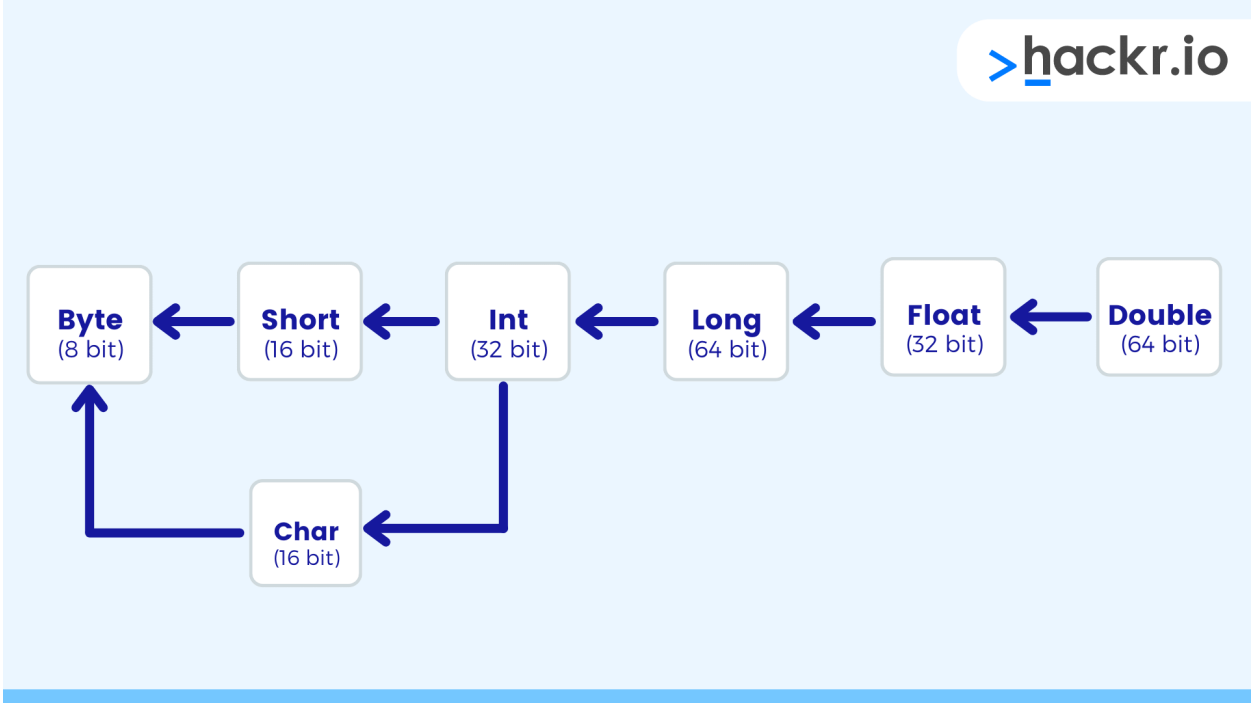
9. Explain Typecasting.

The concept of assigning a variable of one data type to a variable of another data type. This is not possible for the boolean data type. There are two types: implicit and explicit.

10. What are the different types of typecasting?

The different types of typecasting are:

- **Implicit:** Storing values from a smaller data type to the larger data type. It is automatically done by the compiler.
- **Explicit:** Storing the value of a larger data type into a smaller data type. This results in information loss:



  - **Truncation:** While converting a value from a larger data type to a smaller data type, the extra data will be truncated. This code example explains it :

```
float f = 3.14f;
```

```
  int i  = (int) f;
```

After execution, the variable i will contain only 3 and not the decimal portion.

- **Out of Range:** Typecasting does not allow assigning value more than its range; if that happens then the data is lost in such cases. This code example explains it:

```
long l = 123456789;

byte b = (byte) l;       // byte is of not the same range as long so there will be loss of data.
```

11. Explain access modifiers in Java.

Access modifiers are predefined keywords in Java that are used to restrict the access of a class, method, constructor, and data member in another class. Java supports four access modifiers:

- Default
- Private
- Protected
- Public

Modifier	Default	Private	Protected	Public
Same class	yes	yes	yes	yes
Same package subclass	yes	no	yes	yes
Same package non-subclass	yes	no	yes	yes
Different package subclass	no	no	yes	yes

Different package non-subclass	no	no	no	yes

12. What are the default values for local variables?

The local variables are not initialized to any default value, neither primitives nor object references.

OOPs Java Interview Questions

13. What is Object-Oriented Programming?

OOPs is a programming paradigm centered around objects rather than functions. It is not a tool or a programming language, it is a paradigm that was designed to overcome the flaws of procedural programming.

There are many languages that follow OOPs concepts — some popular ones are Java, Python, and Ruby. Some frameworks also follow OOPs concepts, such as Angular.

14. Explain the OOPs concepts.

The following are the various OOPS Concepts:

- **Abstraction:** Representing essential features without the need to give out background details. The technique is used for creating a new suitable data type for some specific application.
- **Aggregation:** All objects have their separate lifecycle, but ownership is present. No child object can belong to some other object except for the parent object.
- **Association:** The relationship between two objects, where each object has its separate lifecycle. There is no ownership.
- **Class:** A group of similar entities.
- **Composition:** Also called the death relationship, it is a specialized form of aggregation. Child objects don't have a lifecycle. As such, they automatically get deleted if the associated parent object is deleted.

- **Encapsulation:** Refers to the wrapping up of data and code into a single entity. This allows the variables of a class to be only accessible by the parent class and no other classes.
- **Inheritance:** When an object acquires the properties of some other object, it is called inheritance. It results in the formation of a parent-child relationship amongst classes involved. This offers a robust and natural mechanism of organizing and structuring software.
- **Object:** Denotes an instance of a class. Any class can have multiple instances. An object contains the data as well as the method that will operate on the data
- **Polymorphism:** Refers to the ability of a method, object, or variable to assume several forms.

Decision Making Java Interview Questions

15. Differentiate between break and continue.

Break	Continue
Used with both loop and switch statement.	Used with only loop statements.
It terminates the loop or switch block.	It does not terminate but skips to the next iteration.

Classes, Objects, and Methods Java Interview Questions

16. What is an Object?

An instance of a Java class is known as an object. Two important properties of a Java object are behavior and state. An object is created as soon as the JVM comes across the new keyword.

17. Define classes in Java.

A class is a collection of objects of similar data types. Classes are user-defined data types and behave like built-in types of a programming language.

Syntax of a class:

```
class Sample{

member variables

methods()

}
```

Example of Class:

```
public class Shape

  {

     String Shape name;

     void area()

        {

        }

     void volume ()

        {

        }

     void num_sides()

        {

        }
```

```
}
```

18. What are static methods and variables?

A class has two sections: one declares variables, and the other declares methods. These are called instance variables and instance methods, respectively. They are termed so because every time a class is instantiated, a new copy of each of them is created.

Variables and methods can be created that are common to all objects and accessed without using a particular object by declaring them static. Static members are also available to be used by other classes and methods.

19. What do you mean by Constructor?

A constructor is a method that has the same name as that of the class to which it belongs. As soon as a new object is created, a constructor corresponding to the class gets invoked. Although the user can explicitly create a constructor, it is created on its own as soon as a class is created. This is known as the default constructor. Constructors can be overloaded.

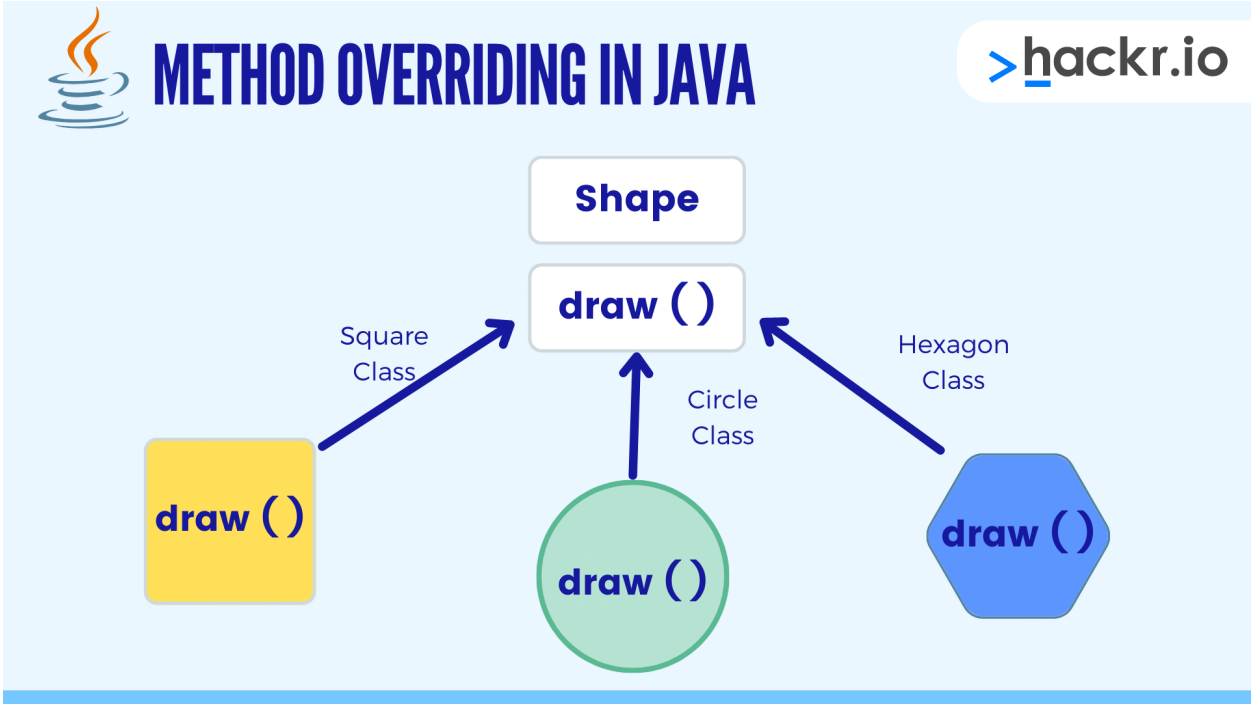
If an explicitly-created constructor has a parameter, then it is necessary to create another constructor without a parameter.

20. What are local variables and instance variables?

Variables that are only accessible to the method or code block in which they are declared are known as local variables. Instance variables, on the other hand, are accessible to all methods in a class.

While local variables are declared inside a method or a code block, instance variables are declared inside a class but outside a method. Even when not assigned, instance variables have a value that can be null, 0, 0.0, or false. This isn't the case with local variables that need to be assigned a value, where failing to assign a value will yield an error. Local variables are automatically created when a method is called and destroyed as soon as the method exits. For creating instance variables, the *new* keyword must be used.

21. What is Method Overriding?



Method overriding in Java allows a subclass to offer a specific implementation of a method that has already been provided by its parent or superclass. Method overriding happens if the subclass method and the Superclass method have:

- The same name
- The same argument
- The same return type

22. What is Overloading?

Overloading is the phenomenon when two or more different methods (method overloading) or operators (operator overloading) have the same representation. For example, the + operator adds two integer values but concatenates two strings. Similarly, an overloaded function called Add can be used for two purposes

1. To add two integers
2. To concatenate two strings

Unlike method overriding, method overloading requires two overloaded methods to have the same name but different arguments. The overloaded functions may or may not have different return types.

23. What role does the final keyword play in Java? What impact does it have on a variable, method, and class?

The final keyword in Java is a non-access modifier that applies only to a class, method, or variable. It serves a different purpose based on the context where it is used.

- **With a class:** When a class is declared as final, then it is disabled from being subclassed i.e., no class can extend the final class.
- **With a method:** Any method accompanying the final keyword is restricted from being overridden by the subclass.
- **With a variable:** A variable followed by the final keyword is not able to change the value that it holds during the program execution. So, it behaves like a constant.

Arrays, Strings and Vectors Java Interview Questions

24. Draw a comparison between Array and ArrayList.

An array necessitates stating the size during the time of declaration, while an array list doesn't necessarily require size as it changes size dynamically. To put an object into an array, there is the need to specify the index. However, no such requirement is in place for an array list. While an array list is parameterized, an array is not parameterized.

25. What are the differences between String, Stringbuilder, and Stringbuffer?

String variables are stored in a constant string pool. With the change in the string reference, it becomes impossible to delete the old value. For example, if a string has stored a value "Old," then adding the new value "New" will not delete the old value. It will still be there, however, in a dormant state. In a Stringbuffer, values are stored in a stack. With the change in the string reference, the new value replaces the older value.

The Stringbuffer is synchronized (and therefore, thread-safe) and offers slower performance than the StringBuilder, which is also a Stringbuffer but is not synchronized. Hence, performance is faster in Stringbuilder than the Stringbuffer.

26. What is the String Pool?

The string pool is a collection of strings stored in the heap memory refers to. Whenever a new object is created, it is checked if it is already present in the string pool. If it is already present, then the same reference is returned to the variable, otherwise a new object is created in the string pool, and the respective reference is returned.

Advanced Java Interview Questions

Interfaces and Abstract Classes Java Interview Questions

27. What do you know about Interfaces?

A Java interface is a template that has only method declarations and not method implementations. It is a workaround for achieving multiple inheritances in Java. Some worth remembering important points regarding Java interfaces are:

- A class that implements the interface must provide an implementation for all methods declared in the interface.
- All methods in an interface are internally public abstract void.
- All variables in an interface are internally public static final.
- Classes do not extend but implement interfaces.

28. How is an Abstract class different from an Interface?

There are several differences between an Abstract class and an Interface in Java, summed up as follows:

- **Constituents:** An abstract class contains instance variables, whereas an interface can contain only constants.

- **Constructor and Instantiation:** While an interface has neither a constructor nor it can be instantiated, an abstract class can have a default constructor that is called whenever the concrete subclass is instantiated.
- **Implementation of Methods –** All classes that implement the interface need to provide an implementation for all the methods contained by it. A class that extends the abstract class, however, doesn't require implementing all the methods contained in it. Only abstract methods need to be implemented in the concrete subclass.
- **Type of Methods:** Any abstract class has both abstract as well as non-abstract methods. Interface, on the other hand, has only a single abstract method.

29. Please explain Abstract class and Abstract method.

An abstract class in Java is a class that can't be instantiated. Such a class is typically used for providing a base for subclasses to extend as well as implementing the abstract methods and overriding or using the implemented methods defined in the abstract class.

To create an abstract class, it needs to be followed by the abstract keyword. Any abstract class can have both abstract as well as non-abstract methods. A method in Java that only has the declaration and not implementation is known as an abstract method. Also, an abstract method name is followed by the abstract keyword. Any concrete subclass that extends the abstract class must provide an implementation for abstract methods.

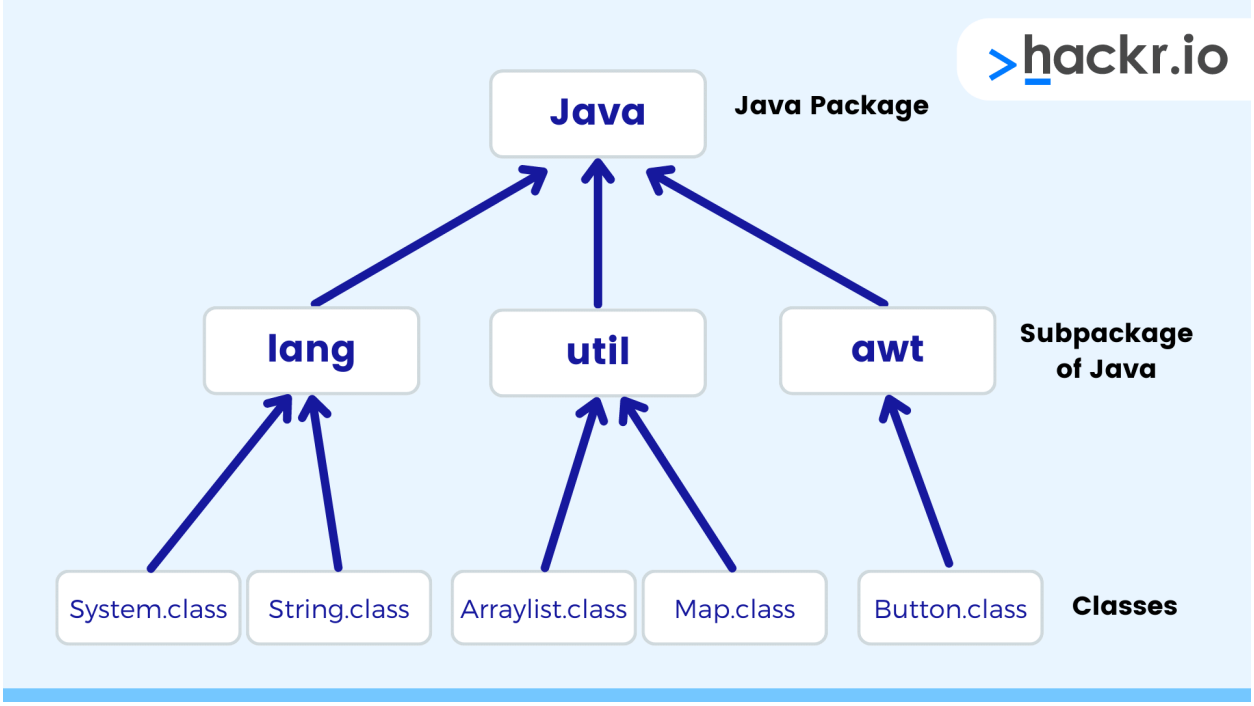
30. What is multiple inheritance? Does Java support multiple inheritance? If not, how can it be achieved?

If a subclass or child class has two parent classes, that means it inherits the properties from two base classes; it has multiple inheritances. Java does not have multiple inheritances as in case the parent classes have the same method names. Then at runtime, it becomes ambiguous, and the compiler is unable to decide which method to execute from the child class.

Packages Java Interview Questions

31. What are packages in Java? State some advantages.

Packages are Java's way of grouping a variety of classes and/or interfaces together. The functionality of the objects decides how they are grouped. Packagers act as "containers" for classes.



Enlisted below are the advantages of Packages:

1. Classes of other programs can be reused.
2. Two classes with the same can exist in two different packages.
3. Packages can hide classes, thus denying access to certain programs and classes meant for internal use only.
4. They also separate design from coding.

Multithreading Java Interview Questions

32. How do you make a thread in Java? Give examples.

To make a thread in Java, there are two options:

- **Extend the Thread Class**: The thread is available in the java.lang.Thread class. To make a thread, you need to extend a thread class and override the run method. For example,

```
public class Addition extends Thread {

public void run() {

}

}
```

A disadvantage of using the thread class is that it becomes impossible to extend any other classes.

Nonetheless, it is possible to overload the run() method in the class

- **Implement Runnable Interface:** Another way of making a thread in Java is by implementing a runnable interface. For doing so, there is the need to provide the implementation for the run() method that is defined as follows:

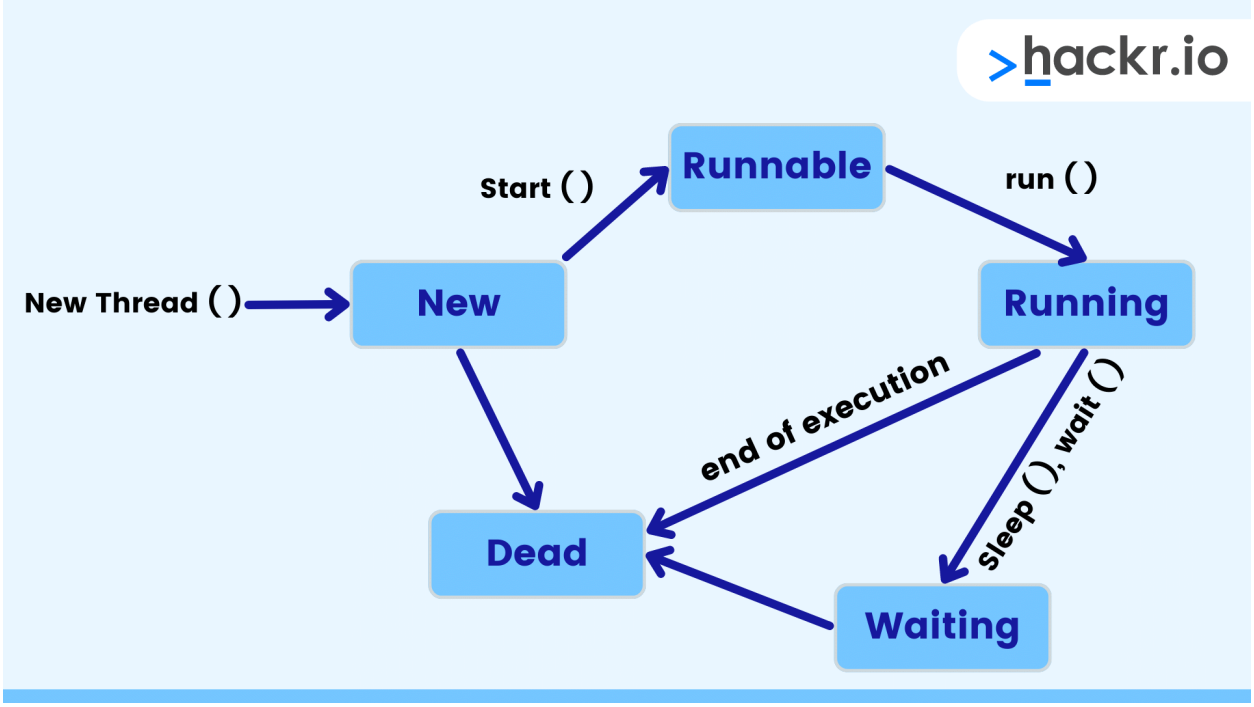
```
interface. For example,

public class Addition implements Runnable {

public void run() {

}

}
```

33. Why do we use the yield() method?

The yield() method belongs to the thread class. It transfers the currently running thread to a runnable state and also allows the other threads to execute. In other words, it gives

equal priority threads a chance to run. Because yield() is a static method, it does not release any lock.

34. Explain the thread lifecycle in Java.



The thread lifecycle has the following states and follows the following order:

- **New:** In the very first state of the thread lifecycle, the thread instance is created, and the start() method is yet to be invoked. The thread is considered alive now.
- **Runnable:** After invoking the start() method, but before invoking the run() method, a thread is in the runnable state. A thread can also return to the runnable state from waiting or sleeping.
- **Running:** The thread enters the running state after the run() method is invoked. This is when the thread begins execution.
- **Non-Runnable:** Although the thread is alive, it is not able to run. Typically, it returns to the runnable state after some time.

- **Terminated:** The thread enters the terminated state once the run() method completes its execution. It is not alive now.

35. When is the Runnable interface preferred over thread class and vice-versa?

In Java, it is possible to extend only one class. Hence, the thread class is only extended when no other class needs to be extended. If it is required for a class to extend some other class than the thread class, then we need to use the Runnable interface.

36. Draw a comparison between notify() and notifyAll() methods.

The *notify()* method is used for sending a signal to wake up a single thread in the waiting pool. Contrarily, the *notifyAll()* method is used for sending a signal to wake up all threads in a waiting pool.

37. How will you distinguish processes from threads?

There are several fundamental differences between a process and a thread, stated as follows:

- **Definition:** A process is an executing instance of a program whereas, a thread is a subset of a process.
- **Changes:** A change made to the parent process doesn't affect child processes. However, a change in the main thread can yield changes in the behavior of other threads of the same process.
- **Communication –** While processes require inter-process communication for communicating with sibling processes, threads can directly communicate with other threads belonging to the same process.
- **Control:** Processes are controlled by the operating system and can control only child processes. On the contrary, threads are controlled by the programmer and are capable of exercising control over threads of the same process to which they belong.

- **Dependence:** Processes are independent entities while threads are dependent entities
- **Memory:** Threads run in shared memory spaces, but processes run in separate memory spaces.

38. What is the join() method? Give an example.

We use the join() method for joining one thread with the end of the currently running thread. It is a non-static method and has an overloaded version. Consider the example below:

```
public static void main (String[] args) {

Thread t = new Thread();

t.start();

t.join();

}
```

The main thread starts execution in the example mentioned above. As soon as the execution reaches the code t.start(), then the thread t starts its stack for execution. The JVM switches between the main thread and the thread there. Once the execution reaches the t.join(), then the thread t alone is executed and allowed to complete its task. Afterwards, the main thread resumes execution.

39. How do you make a thread stop in Java?

There are three methods in Java to stop the execution of a thread:

- **Blocking:** This method is used to put the thread in a blocked state. The execution resumes as soon as the condition of the blocking is met. For instance, the ServerSocket.accept() is a blocking method that listens for incoming socket connections and resumes the blocked thread only when a connection is made.

- **Sleeping:** This method is used for delaying the execution of the thread for some time. A thread upon which the sleep() method is used is said to enter the sleep state. It enters the runnable state as soon as it wakes up i.e., the sleep state is finished. The time for which the thread needs to enter the sleep state is mentioned inside the braces of the sleep() method. It is a static method.
- **Waiting:** Although it can be called on any Java object, the wait() method can only be called from a synchronized block.

Exception Handling Java Interview Questions

40. What are the various types of exceptions? How do you handle them?

Java has provision for two types of exceptions:

- **Checked Exceptions:** Classes that extend the Throwable class, except Runtime exception and Error, are called checked exceptions. Such exceptions are checked by the compiler during the compile time. These types of exceptions must either have appropriate try/catch blocks or be declared using the throws keyword. ClassNotFoundException is a checked exception.
- **Unchecked Exceptions:** Such exceptions aren't checked by the compiler during the compile time. As such, the compiler doesn't necessitate handling unchecked exceptions. Arithmetic Exception and ArrayIndexOutOfBounds Exception are unchecked exceptions.

Exceptions in Java are handled in two ways:

Declaring the throws keyword: We can declare the exception using throws keyword at the end of the method. For example:

```
class ExceptionCheck{

public static void main(String[] args){
```

```
add();

}

public void add() throws Exception{

addition();

}

}
```

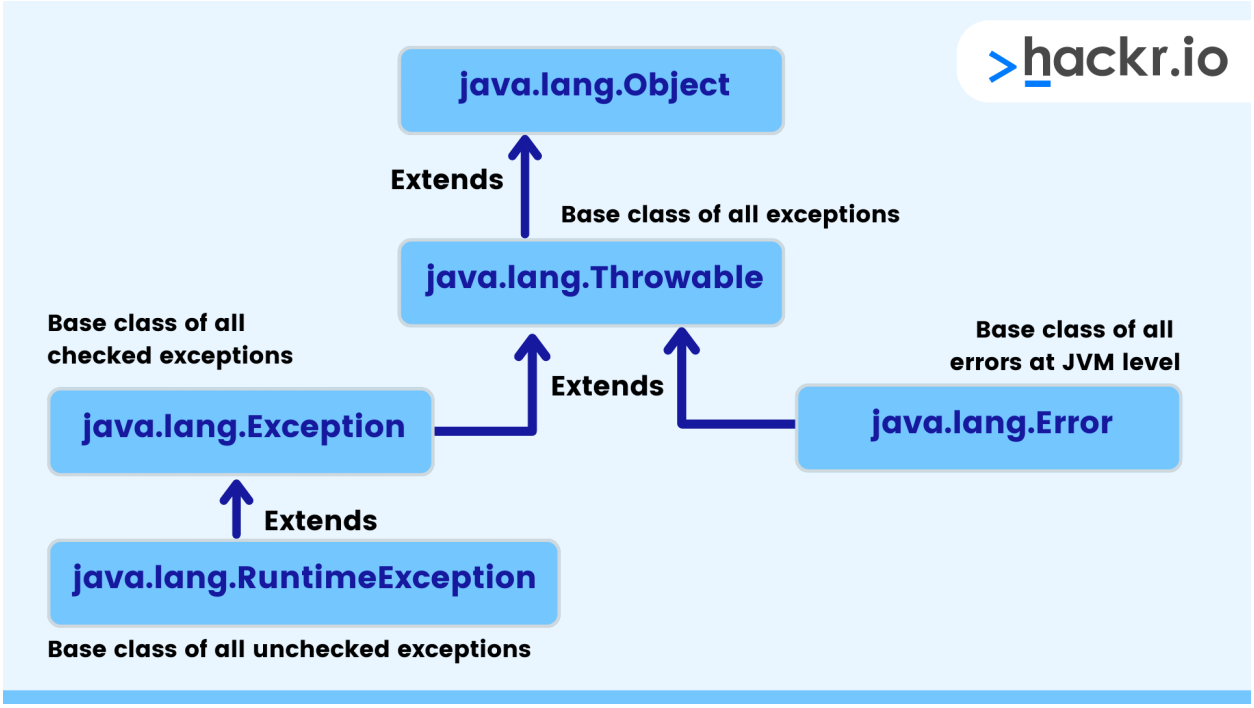
Using try/catch: Any code segment that is expected to yield an exception is surrounded by the try block. Upon the occurrence of the exception, it is caught by the catch block that follows the try block. For example:

```
class ExceptionCheck{

public static void main (String[] args) {

add();

}

public void add(){

try{

addition();

}

catch(Exception e)

{

e.printStacktrace();

}
```

```
}
```

```
}
```

41. Draw the Java Exception Hierarchy.



42. Is it possible to write multiple catch blocks under a single try block?

Yes, it is possible to write several catch blocks under a single try block. However, the approach needs to be from specific to general. The following example demonstrates it:

```
public class Example {
```

```
public static void main(String args[]) {
```

```
try {
```

```
int a[]= new int[10];
```

```
a[10]= 10/0;
```

```
}
catch(ArithmeticException e)
{
System.out.println("Arithmetic exception in first catch block");
}
catch(ArrayIndexOutOfBoundsException e)
{
System.out.println("Array index out of bounds in second catch block");
}
catch(Exception e)
{
System.out.println("Any exception in third catch block");
}
}
```

43. How does the throw keyword differ from the throws keyword?

While the throws keyword allows declaring an exception, the throw keyword is used to explicitly throw an exception.

Checked exceptions can't be propagated with throw only, but throws allow doing so without the need for anything else.

The throws keyword is followed by a class, whereas the throw keyword is followed by an instance. The throw keyword is used within the method, but the throws keyword is used with the method signature.

Furthermore, it is not possible to throw multiple exceptions, but it is possible to declare multiple exceptions.

44. Explain various exceptions handling keywords in Java.

There are two crucial exception handling keywords in Java, followed by the third keyword final, which may or may not be used after handling exceptions.

try:

If and when a code segment has chances of having an abnormality or an error, it is placed within a try block. When the exception is raised, it is handled and caught by the catch block.

The try block must have a catch() or a final() or both blocks after it.

catch:

When an exception is raised in the try block, it is handled in the catch block.

final:

This block is executed regardless of the exception. It can be placed either after try{} or catch {} block.

45. Explain exception propagation.

The method at the top of the stack throws an exception if it is not caught. It moves to the next method and goes on until caught. Example:

```
public class Sum()

{

public static void main(String args[])

{

addition()
```

```
}

public void addition()

{

add();

}

}
```

The stack of the above code is:

```
add()

addition()

main()
```

If an exception occurred in the add() method is not caught, then it moves to the method addition(). It is then moved to the main() method, where the flow of execution stops. It is called Exception Propagation.

File Handling Java Interview Questions

46. Is an empty file name with .java extension a valid file name?

Yes, Java permits us to save our java file by .java only. It is compiled by javac and run by the java class name. Here's an example:

```
public class Any()

{

public static void main(String args[])

{
```

```
System.out.println("Hello Java File here!");

}

}
```

To compile: javac.java

To run: Java Any

Collections Java Interview Questions

47. What are collections? What are their constituents?

A group of objects in Java is known as collections. Java.util package contains, along with date and time facilities, internationalization, legacy collection classes, etc., the various classes and interfaces for collecting. Alternatively, collections can be considered as a framework designed for storing the objects and manipulating the design in which the objects are stored. You can use collections to perform the following operations on objects:

- Deletion
- Insertion
- Manipulation
- Searching
- Sorting

Following are the various constituents of the collections framework:

- **Classes:** Array List, Linked List, Lists, and Vector
- **Interfaces:** Collection, List, Map, Queue, Set, Sorted Map, and Sorted Set
- **Maps:** HashMap, HashTable, LinkedHashMap, and TreeMap
- **Queues:** Priority Queue
- **Sets:** Hash Set, Linked Hash Set, and Tree Set

48. How do you differentiate HashMap and HashTable?

HashMap in Java is a map-based collection class, used for storing key & value pairs. It is denoted as HashMap<Key, Value> or HashMap<K, V> HashTable is an array of a list, where each list is called a bucket.

Values contained in a HashTable are unique and depend on the key. Methods are not synchronized in HashMap, while key methods are synchronized in HashTable.

However, HashMap doesn't have thread safety, while HashTable has the same.

For iterating values, HashMap uses an iterator and HashTable uses an enumerator. HashTable doesn't allow anything that is null, while HashMap allows one null key and several null values.

In terms of performance, HashTable is slow. Comparatively, HashMap is faster.

49. What is a Map and what are the types?

A Java Map is an object that maps keys to values. It can't contain duplicate keys, and each key can map to only one value. In order to determine whether two keys are the same or distinct, Map makes use of the equals() method. There are 4 types of Map in Java, described as follows:

- **HashMap:** It is an unordered and unsorted map and hence, is a good choice when there is no emphasis on the order. A HashMap allows one null key and multiple null values and doesn't maintain any insertion order.
- **HashTable:** Doesn't allow anything null and has methods that are synchronized. As it allows for thread safety, the performance is slow.
- **LinkedHashMap:** Slower than a HashMap but maintains insertion order and has a faster iteration.
- **TreeMap:** A sorted Map providing support for constructing a sort order using a constructor.

50. What is a Priority Queue?

A priority queue, like a regular queue, is an abstract data type, but it has a priority associated with each element contained by it.

The element with the high priority is served before the element with low priority in a priority queue. Elements in a priority queue are ordered either according to the comparator or naturally. The order of the elements in a priority queue represents their relative priority.

51. What is a Set? Explain the types in Java Collections.

In Java, a set is a collection of unique objects. It uses the *equals()* method to determine whether two objects are the same or not. The various types of set in Java Collections are:

1. **Hash Set:** An unordered and unsorted set that uses the hash code of the object for adding values. Used when the order of the collection isn't important
2. **Linked Hash Set:** This is an ordered version of the hash set that maintains a doubly-linked list of all the elements. Used when iteration order is mandatory. Insertion order is the same as that of how elements are added to the Set.
3. **Tree Set:** One of the two sorted collections in Java, it uses Read-Black tree structure and ensures that the elements are present in the ascending order.

52. What is ordered and sorted when it comes to collections?

- **Ordered:** Values are stored in a collection in a specific order, but the order is independent of the value. Example: List
- **Sorted:** The collection has an order which is dependent on the value of an element. Example: SortedSet

Miscellaneous Java Interview Questions

53. What are the various types of garbage collectors in Java?

The Java programming language has four types of garbage collectors:

1. **Serial Garbage Collector:** Using only a single thread for garbage collection, the serial garbage collector works by holding all the application threads. It is designed especially for single-threaded environments. Because serial garbage collector freezes all application threads while performing garbage collection, it is most suitable for command-line programs only. For using the serial garbage collector, one needs to turn on the -XX:+UseSerialGC JVM argument.
2. **Parallel Garbage Collector:** Also known as the throughput collector, the parallel garbage collector is the default garbage collector of the JVM. It uses multiple threads for garbage collection, and like a serial garbage collector freezes all application threads during garbage collection.
3. **CMS Garbage Collector:** Short for Concurrent Mark Sweep, CMS garbage collector uses multiple threads for scanning the heap memory for marking instances for eviction, followed by sweeping the marked instances. There are only two scenarios when the CMS garbage collector holds all the application threads:
    - When marking the referenced objects in the tenured generation space.
    - If there is a change in the heap memory while performing the garbage collection, CMS garbage collector ensures better application throughput over parallel garbage collectors by using more CPU resources. For using the CMS garbage collector, the XX:+USeParNewGC JVM argument needs to be turned on.
4. **G1 Garbage Collector:** Used for large heap memory areas, G1 garbage collector works by separating the heap memory into multiple regions and then executing garbage collection in them in parallel. Unlike the CMS garbage collector that compacts the memory on STW (Stop The World) situations, G1 garbage collector compacts the free heap space right after reclaiming the

memory. Also, the G1 garbage collector prioritizes the region with the most garbage. Turning on the –XX:+UseG1GC JVM argument is required for using the G1 garbage collector.

54. What do you understand by synchronization? What is its most significant disadvantage?

If several threads try to access a single block of code, then there is an increased chance of producing inaccurate results. Synchronization is used to prevent this. Using the synchronization keyword makes a thread need a key to access the synchronized code. Simply, synchronization allows only one thread to access a block of code at a time. Each Java object has a lock, and every lock has only one key. A thread can access a synchronized method only if it can get the key to the lock of the object. The following example demonstrates synchronization:

```
public class ExampleThread implements Runnable {

public static void main (String[] args){

Thread t = new Thread();

t.start();

}

public void run(){

synchronized(object){

{

}

}
```

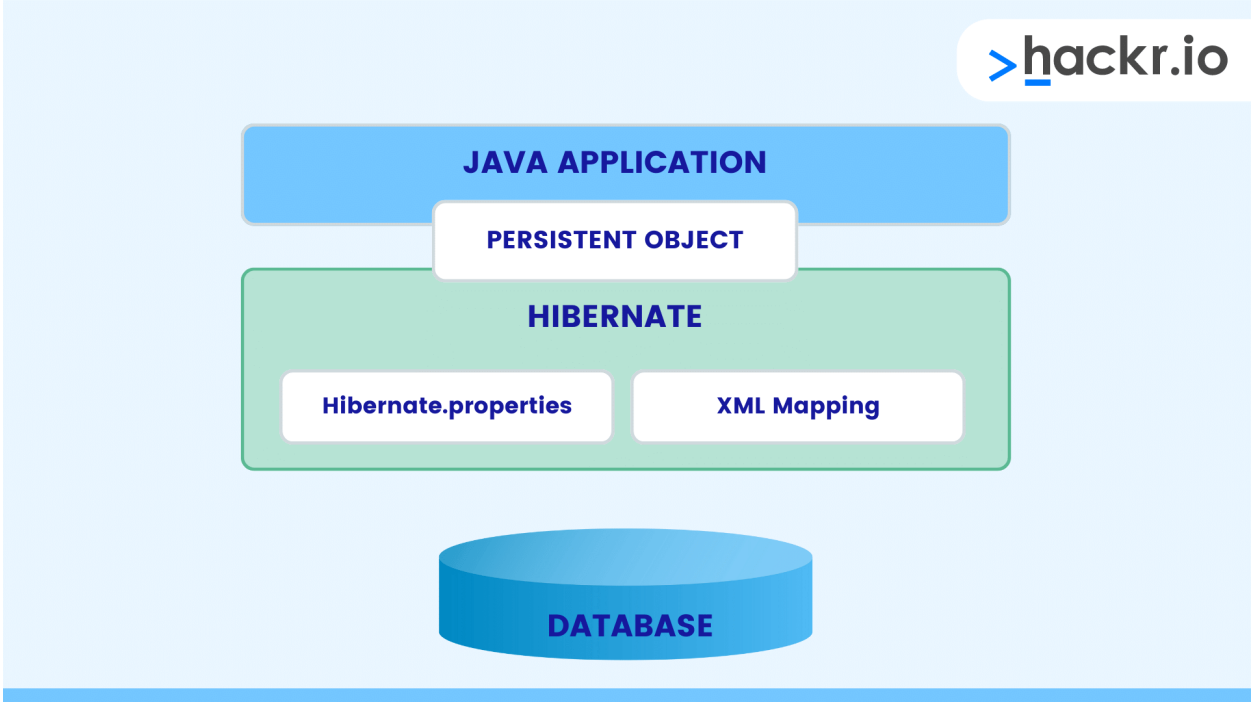
Note: It is recommended to avoid implementing synchronization for all methods. This is because when only one thread can access the synchronized code, the next thread needs to wait. Consequently, it results in slower performance of the program.

55. What is the difference between execute(), executeQuery(), and executeUpdate()?

- **execute():** Used for executing an SQL query. It returns TRUE if the result is a ResultSet, like running Select queries, and FALSE if the result is not a ResultSet, such as running an Insert or an Update query.
- **executeQuery():** Used for executing Select queries. It returns the ResultSet, which is not null, even if no records are matching the query. The executeQuery() method must be used when executing select queries so that it throws the java.sql.SQLException with the 'executeQuery method cannot be used for update' message when someone tries to execute an Insert or Update statement.
- **executeUpdate():** Used for executing Delete/Insert/Update statements or DDL statements that return nothing. The output varies depending on whether the statements are Data Manipulation Language (DML) statements or Data Definition Language (DDL) statements. The output is an integer and equals the total row count for the former case, and 0 for the latter case.

Note: The execute() method needs to be used only in a scenario when there is no certainty about the type of statement. In all other cases, either use executeQuery() or executeUpdate() method.

56. Provide an example of the Hibernate architecture.



57. Could you demonstrate how to delete a cookie in JSP with a code example?

The following code demonstrates deleting a cookie in JSP:

```
Cookie mycook = new Cookie("name1","value1");

response.addCookie(mycook1);

Cookie killmycook = new Cookie("mycook1","value1");

killmycook . set MaxAge ( 0 );

killmycook . set Path ("/");

killmycook . addCookie ( killmycook 1 );
```

58. Write suitable code examples to demonstrate the use of final, final, and finalize.

Final: The final keyword is used for restricting a class, method, and variable. A final class can't be inherited, a final method is disabled from overriding, and a final variable becomes a constant i.e., its value can't be changed.

```
class FinalVarExample {

public static void main( String args[])

{

final int a=10;

a=50; /* Will result in an error as the value can't be changed now*/

}
```

Finally: Any code inside the final block will be executed, irrespective of whether an exception is handled or not.

```
class FinallyExample {

public static void main(String args[]){

try {

int x=100;

}

catch(Exception e) {

System.out.println(e);

}

finally {

System.out.println("finally block is executing");}
```

```
}

}

}
```

Finalize: The finalize method performs the clean up just before the object is garbage collected.

```
class FinalizeExample {

public void finalize() {

System.out.println("Finalize is called");

}

public static void main(String args[])

{

FinalizeExample f1=new FinalizeExample();

FinalizeExample f2=new FinalizeExample();

f1= NULL;

f2=NULL;

System.gc();

}

}
```

59. What purpose does the Volatile variable serve?

The value stored in a volatile variable is not read from the thread's cache memory but from the main memory. Volatile variables are primarily used during synchronization.

60. Please compare serialization and deserialization.

Serialization is the process by which Java objects are converted into the byte stream.

Deserialization is the exact opposite process of serialization where Java objects are retrieved from the byte stream.

A Java object is serialized by writing it to an ObjectOutputStream and deserialized by reading it from an ObjectInputStream.

61. What is OutOfMemoryError?

Typically, the OutOfMemoryError exception is thrown when the JVM is not able to allocate an object due to running out of memory. In such a situation, no memory could be reclaimed by the garbage collector.

There can be several reasons that result in the OutOfMemoryError exception, of which the most notable ones are:

- Holding objects for too long
- Trying to process too much data at the same time
- Using a third-party library that caches strings
- Using an application server that doesn't perform a memory cleanup post the deployment
- When a native allocation can't be satisfied

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

The execution Java program starts with public static void main(String args[ ]), also called the main() method.

- **public:** It is an access modifier defining the accessibility of the class or method. Any class can access the main() method defined public in the program.
- **static:** The keyword indicates the variable, or the method is a class method. The method main() is made static so that it can be accessed without creating the instance of the class. When the method main() is not made static, the compiler

throws an error because the 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 that does not return any type of value.
- **main:** JVM searches this method when starting the execution of any program, with the particular signature only.
- **String args[]:** The parameter passed to the main method.

63. What are wrapper classes in Java?

Wrapper classes are responsible for converting the Java primitives into the reference types (objects). A class is dedicated to every primitive data type. They are known as wrapper classes because they wrap the primitive data type into an object of that class. It is present in the Java.lang package. The table below displays the different primitive types and wrapper classes.

Simple Type	Wrapper Class
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer
long	Long

64. Explain the concept of boxing, unboxing, autoboxing, and auto unboxing.

- **Boxing:** The concept of putting a primitive value inside an object is called boxing.
- **Unboxing:** Getting the primitive value from the object.
- **Autoboxing:** Assigning a value directly to an integer object.
- **Auto unboxing:** Getting the primitive value directly into the integer object.

```
public class BoxUnbox

{

public static void main(String args[])

{

int i = 5;

Integer ii = new Integer(i);              /*Boxing*/

Integer jj = i; /*Unboxing*/

int j = jj.intValue();       /*Unboxing*/

int k = jj; /*AutoUnboxing*/

}

}
```

65. Define the Singleton class. How can a class be made Singleton?

A Singleton class allows only one instance of the class to be created. A class can be made singleton with the following steps:

1. Creating a static instance of the class with the class.
2. By not allowing the user to create an instance with default constructor by defining a private constructor.

3. Create a static method to return the object of an instance of A.

```
public class Singleton

{

public static void main(String args[])

{

Single obj1 = Single.getInstance();  /* both would point to one and same
instance of the class */

Single obj2 = Single.getInstance();

}

}

class Single

{

static Single obj = new Single(); /* step a*/

private Single() /* step b*/

{

}

public static Single getInstance()

{

return obj; /* step c*/

}

}
```

66. What if the public static void is replaced by static public void, will the program still run?

Yes, the program would compile and run without any errors as the order of the specifiers doesn't matter.

67. What is the difference between == and equals()?

Equals()	==
It is a method of String class	It is an operator
Content comparison	Address comparison

```
class Operator {

public static void main(String[] args)

{

/* integer-type*/

System.out.println(10 == 20);

/* char-type*/

System.out.println('a' == 'b');

/* char and double type*/
```

```
public class Equals{

public static void main(String[] args)

{

String s1 = new String("HELLO");

String s2 = new String("HELLO");

System.out.println(s1 == s2);

System.out.println(s1.equals(s2));

}
```

```
System.out.println('a' == 97.0);

/* boolean type*/

System.out.println(true == true);

}

}
```

68. Why don't we use pointers in Java?

Pointers are considered to be unsafe, and increase the complexity of the program, adding the concept of pointers can be contradicting. Also, JVM is responsible for implicit memory allocation; thus, to avoid direct access to memory by the user, pointers are discouraged in Java.

69. What is the difference between this() and super()?

this()	super()
Represents the current instance of the class	Represents the current instance of the parent/base class
It is used to call the default constructor of the same class	It is used to call the default constructor of the parent/base class.
Accesses method of the current class	Accesses method of the base class
Points current class instance	Points to the superclass instance.
Must be the first line of the block	It must be the first line of the block.

Java Coding Interview Questions

Apart from having good knowledge about concepts of Java programming, you are also tested for your skills in coding in Java programming language. The following are Java coding interview questions that are relevant for freshers and are quite popular amongst Java programming interviews.

70. Take a look at the two code snippets below. What is the important difference between the two?

i.

```
class Adder {

Static int add(int a, int b)

{

return a+b;

}

Static double add( double a, double b)

{

return a+b;

}

public static void main(String args[])

{

System.out.println(Adder.add(11,11));

System.out.println(Adder.add(12.3,12.6));

}}
```

ii.

```
class Car {

void run(){

System.out.println("car is running");

}

Class Audi extends Car{

void run()

{

System.out.prinltn("Audi is running safely with 100km");

}

public static void main( String args[])

{

Car b=new Audi();

b.run();

}

}
```

Code snippet **i.** is an example of method overloading while the code snippet **ii.** demonstrates method overriding.

71. Write a program for string reversal without using an inbuilt function.

```
public class Reversal

{
```

```
public static void main(String args[])

{

String input = "Java Interview";

System.out.println("Given String -> " + "Java Interview");

char charArray[] = input.toCharArray();

System.out.println("Reversed String -> ");

for(int i = charArray.length-1;i>=0; i--)

{

System.out.print(charArray[i]);

}

System.out.println();

}

}
```

72. Write a program to delete duplicates from an array.

```
import java.util.ArrayList;

import java.util.LinkedHashSet;

import java.util.List;

import java.util.Set;

class RemoveDuplicates

{
```

```
public static void main(String args[])

{

/*create ArrayList with duplicate elements*/

ArrayList duplicate = new ArrayList();

duplicate.add(5);

duplicate.add(7);

duplicate.add(1);

duplicate.add(4);

duplicate.add(1);

duplicate.add(7);

System.out.println("Given array: "+ duplicate);

Set <Integer> withoutDuplicates = new LinkedHashSet<Integer>(duplicate)

duplicate.clear();

duplicate.addAll(withoutDuplicates);

System.out.println("Array without duplicates: "+ duplicate);

}

}
```

73. Write a program to reverse a number.

```
import java.util.Scanner;

public class NumberReversal
```

```
{

public static void main(String args[])

{

System.out.println("Please enter the number to be reversed");

Scanner sc = new Scanner (System.in);

int number = sc.nextInt();

int reverse = reverse(number);

System.out.println("Reverse of number: " + number + " is " + reverse(number));

}

public static int reverse(int number){

int reverse = 0;

int remainder = 0;

do{

remainder = number%10;

reverse = reverse*10 + remainder;

number = number/10;

}while(number > 0);

  return reverse;

}

}
```

74. Write a program that implements binary search.

```
import java.util.Scanner;

import java.util.Arrays;

public class Binary {

public static void main(String[] args) {

System.out.println("Enter total number of elements : ");

Scanner s = new Scanner (System.in);

int length = s.nextInt();

int[] input = new int[length];

System.out.printf("Enter %d integers", length);

for (int i = 0; i < length; i++) {

input[i] = s.nextInt();

}

/* binary search requires the input array to be sorted so we must sort the
array first*/

Arrays.sort(input);

System.out.print("the sorted array is: ");

for(int i= 0; i<= length-1;i++)

{

System.out.println(input[i] + " ,");

}
```

```
System.out.println("Please enter number to be searched in sorted array");

int key = s.nextInt();

int index = BSearch(input, key);

if (index == -1) {

System.out.printf("Sorry, %d is not found in array %n", key);

} else {

System.out.printf("%d is found in array at index %d %n", key,

index);

}

}

public static int BSearch(int[] input, int number) {

int low = 0;

int high = input.length - 1;

while (high >= low) {

int middle = (low + high) / 2;

if (input[middle] == number) {

return middle;

} else if (input[middle] < number) {

low = middle + 1;

} else if (input[middle] > number) {
```

```
high = middle - 1;

}

}

return -1;

}

}
```

75. Write a program to check if a number is prime.

```
import java.util.Scanner;

public class Prime

{

public static void main(String args[])

{

System.out.println("Enter the number to check: ");

Scanner sc = new Scanner(System.in);

int num = sc.nextInt();

boolean isPrime = false;

 if(num!=0)

 {

 isPrime = checkPrime(num);

 }else
```

```
 {
 System.out.println("Enter valid number");
 }
 if(isPrime == false)
 {
 System.out.println(" NOT PRIME!!");
 }
 else
 {
 System.out.println("PRIME!!");
 }
}
public static boolean checkPrime(int number)
{
int sqrt = (int) Math.sqrt(number) + 1;
for(int i = 2; i<sqrt; i++)
{
if(number % i== 0)
{
return false;
```

```
}

}

return true;

}

}
```

76. Write a program to print Fibonacci series.

```
import java.util.Scanner;

public class Fibo

{

public static void main(String args[])

{

System.out.println("Enter the number upto which Fibonacci series should be
printed ");

Scanner sc = new Scanner(System.in);

int num = sc.nextInt();

System.out.println("Fibonacci Series upto %d is" + num);

for(int i=1; i<=num; i++)

{

System.out.print(fib(i) + " ");

}

}
```

```
public static int fib(int n)

{

if(n ==1 || n==2)

{

return 1;

}

return fib(n-1) + fib(n-2);

}

}
```

77. Write a program to check if the given string is a palindrome.

```
import java.util.Scanner;

public class PalinDrome

{

public static void main(String args[])

{

System.out.println("Enter the string to check");

Scanner sc = new Scanner(System.in);

String str = sc.nextLine();

boolean isPalindrome;

isPalindrome = checkPalindrome(str);
```

```
if(str.equals(" "))

{

System.out.println("Enter valid string");

}

else

{

if(isPalindrome)

{

System.out.println("PALINDROME!!");

}

else

{

System.out.println("NOT A PALINDROME!!");

}

}

}

public static boolean checkPalindrome(String input)

{

int str_length = input.length();

int i=0, j= str_length-1;
```

```
while(i<j)

{

if(input.charAt(i) != input.charAt(j))

return false;

i++;

j--;

}

return true;

}

}
```

78. Write a program to print the following pattern.

```
*

* *

* * *

* * * *

* * * * *
```

Answer:

```
public class Pattern

{

public static void main(String args[])

{
```

```
for(int i=5; i>=0; i--)

{

System.out.println();

for(int j=i; j<5;j++)

{

System.out.print(" * ");

}

}

System.out.println();

}

}
```

79. Write a program to swap two numbers.

```
import java.util.Scanner;

public class Swap

{

public static void main(String args[])

{

Scanner s = new Scanner(System.in);

System.out.println("Enter a number: ");

int a = s.nextInt();
```

```
System.out.println("Enter second number: ");

int b = s.nextInt();

System.out.println("Value of a and b before swapping: " + "a = " +a  + " b = "
+ b);

swap(a,b);

}

public static void swap(int a , int b)

{

int swap_variable;

swap_variable = a;

a = b;

b = swap_variable;

System.out.println("Value of a and b after swapping: " + "a = " +a  + " b = " +
b);

}

}
```

80. Write a program to check if the given number is an Armstrong number.

```
import java.util.Scanner;

public class Armstrong

{

public static void main(String args[])
```

```
{

Scanner s = new Scanner(System.in);

System.out.println("Enter a number: ");

int number = s.nextInt();

int a=number, sum = 0, num=0;

 while(a%10 !=0)

 {

 num = a%10;

 sum = sum + (num*num*num);

a = a/10;

 }

 if(sum == number)

 {

 System.out.println("Armstrong Number!");

 }

 else

 {

 System.out.println("Not an Armstrong Number!");

 }

}
```

```
}
```

Summary

These core Java Interview Questions and Java Programming Interview Questions are a great way to prepare you for the interview.

You can also take a look at this course for further reading and preparing for a Java-based interview: Java interview Guides: 200+ Interview Question and Answer

We recommend this book to help you succeed in your future Java interviews: Elements of Programming Interviews in Java: The insider guide second edition

Frequently Asked Questions

What are the basic Java questions asked in an interview?

There are several basic Java interview questions that can appear in an interview. Look at the ones we've listed above to get a sense of them.

How should I prepare for a Java interview?

You should prepare for a Java interview by learning both the theory and practicing coding. There are several related questions above.

What are the advanced Java interview questions?

Advanced Java interview questions can be of many kinds, including both theory and coding-based questions. Check out the list of Java programming questions above to see what they are like.