**Q: What is JRE, and why is it required?**

JRE stands for "Java Runtime Environment" which you usually download as a Java software. The JRE comprises of the Java Virtual Machine, Java platform classes, and supporting libraries. The JRE is the runtime component of Java software and is all you need to run any Java application.

**Q: What is JDK, and why is it required?**

The JDK is a superset of the JRE and includes everything that the JRE contains. Additionally, it comes with the compilers and debuggers tools required for developing Java applications.

**Q: What is JVM, and why is it required?**

JVM stands for The Java Virtual machine. It translates and executes the Java bytecode. It's the entity which transforms Java to become a "portable language" (i.e., write once, run anywhere). Though, each platform has its implementation of JVM like the Windows, Linux, macOS, etc. have a distinct version of JVM to run bytecode.

**Q: Distinguish between the Path and Classpath?**

The <Path> and <Classpath> are OS-level environment variables. Path defines the location where the system can look up for the executables (.exe) files, and classpath specifies the location of the Java class files.

**Q: Distinguish between a constructor and method?**

A constructor gets automatically invoked to create an object, whereas the method gets called explicitly.

Q: Is it permissible for a constructor to have a different name than its class name in Java?

No, constructors in Java should have the same name as their classes. If the name is different, then it would behave like a standard method.

Q: Is there any difference between an argument and a parameter?

While defining methods, you pass variables which you refer as parameters. And when you call these methods and pass values for the variables, then they are phrased as arguments.

Q: How your program would behave if you declare the main method as private?

It would get compiled correctly but will throw the error "Main method not public." at runtime.

Q: What if an application get multiple classes having main() methods?

It's certainly possible to have multiple main methods in different classes. When you start the application, you've to provide the startup class name for execution. The JVM then looks up for the main method only in the class whose name you've supplied. Hence, you won't observe any conflict with the multiple classes having the <main()> definition.

Q: What difference you see between pass by reference and pass by value in Java?

Pass by reference indicates, passing the address itself rather than passing the value. Pass by value means is giving a copy of the value.

Q: What do you understand by Byte Code?

Java compiler generates bytecode for all the Java code and converts into class files. The bytecode is platform-independent and needs the platform-specific JVM for the execution.

**Q: What do you make of each keyword in public static void main(String args[])?**

Public- <main(...)> is the entry point method which the JVM calls when a program is run. So it is mandatory to be accessible from the Java environment. Hence, the access specifier has to be public.

Static- JVM must be capable of calling this method w/o creating an instance of the class. So the method has to be declared as static.

Void- <main()> doesn't return anything, so its return type must be void.

The argument string represents the argument type passed from the console, and the <args> is an array of strings specified at the command line.

Q: How to compare the final, finally, and finalize keywords?

final– It's used to declare a constant.

Variables defined in an interface are implicitly final.

You can't extend a final class.

finally– It makes you handle exceptions.

It's a keyword used for exception handling. The code under the <finally> block gets executed apparently.

finalize– It helps in garbage collection.

The <finalize()> method is used just before an object is destroyed and garbage collected.

Q: Can you compile a Java class successfully without having the main method?

Yes, we can compile, but it won't run. The main method works as the startup function for a Java class, and the JVM calls it for the program execution.

Q: What do you make of System, out and <println> in the function System.out.println()?

System -> A predefined final class,

out -> PrintStream object and,

The <println> -> built-in overloaded method of the out object.

Q: What do you understand by the explicit casting?

It's a process which instructs the complier about transforming the object into a different type.

e.g. long no = 99999;  
int new\_no = (int) no; // Explicit casting

Q: Would a Java program compile/run if we use <static public void> instead of <public static void>?

Yes, the program will compile and run as usual.

Q: How would you prove that an array is not null but is empty?

Call the <Print array.length>. It will print 0. That suggests that the array is empty. If it would've been null then, it would've thrown a NullPointerException on calling the <Print array.length>.

Q: What do you understand of Garbage Collection and how to call it explicitly?

If the object is no longer belong to any variable, Java automatically reclaims the memory. This process is known as garbage collection. You can use the <System.gc()> method to call it explicitly.

Q: How comes an unreachable object become reachable again, is it at all possible?

Yes, an unreachable object may get to reachable state. It can happen if the object <finalize> method gets called during the garbage collection, and there you have set an object referring to it. This situation would cause the garbage collection to skip and make the object reachable again.

Top 10 Java Interview Questions - Read about the internals of Java Classes

The concept of Java classes is a very vast and complex area. You can read the below set of Java interview questions on classes specially prepared for the Software test engineers.

Q: What is an interface, and why is it used?

An interface is similar to a class which may contain method’s signature only but not bodies, and it is a formal set of method and constant declarations that must be defined by the class that implements it.

Interfaces are useful for:

1- Declaring methods that one or more classes are expected to implement.

2- Capturing similarities between unrelated classes without forcing a class relationship.

3- Determining an object’s programming interface without revealing the actual body of the class.

Q: What is an abstract class, and why is it used?

The abstract class serves as a template, and you can't instantiate it. It may contain static data, and you've to extend it to make it functional.

Also, you must note that any class which has an abstract method automatically turns itself abstract.

Q: What difference you see between an Abstract class and an Interface?

An interface by definition has all public members without any implementation. While an abstract class may group different flavors of class members like private, protected, etc. but has at least one abstract method.

Every abstract class must provide an instance method that defines its default behavior. While in an Interface, only the declaration of constants and instance methods is permissible, you can't implement the default behavior, and all methods are abstract by default.

Q: Are there any performance implications of Interfaces over abstract classes, if yes, then specify?

Interfaces require additional indirections to find methods in the implementing class. This behavior makes them slower than in abstract classes. Another fact that a QA engineer should know that a class can only extend a single abstract class whereas it can implement multiple interfaces.

Also, with interfaces, one has to address all of its methods, which lead to extra efforts while developing an automation framework.

Q: What if you declare a class w/o any access modifiers, where can you use this class in your program?

When we don't specify any access modifier for a class, then Java assigns package level access to it. It implies that you can access such a class from other classes and interfaces within the boundary of that package.

Q: Explain the difference between function overloading and overriding?

1- Overloading talks about the relationship between methods of the same class, whereas the overriding focuses on the relationship between a superclass method and subclass method.

2- Overloading doesn't impact the inheritance from the superclass, whereas the overriding impedes inheritance from the superclass.

3- In overloading, you have different methods sharing the same name, whereas in overriding, the methods in subclasses replace the superclass versions.

4- Overloading requires the methods to be implemented with distinct signatures, whereas overriding limits to use the same markup.

Q: How would you describe the difference between this() and super()?

You can use this() for invoking the constructor of the class while super() helps to call the superclass constructor.

Q: What difference you see between superclass and subclass?

A superclass is the one that you inherit, whereas the subclass is a class that makes the inheriting.

Q: What do you know about anonymous and inner classes?

Inner class- are the ones having their definition within other classes, including those defined in methods. These can assume any accessibility, including private, public, etc.

Anonymous class- It is a class which has its definition inside a method with no name. You can instantiate or declare it in the same place, and it doesn't support explicit constructors.

Q: Describe the difference between a Sub-Class and an Inner Class?

A subclass is a class which gets inherited from another class termed as the superclass. It can easily access all public/protected methods and fields of its superclass.

The inner class is a class which gets cradled within another class. An Inner class can access all variables and methods provided by the outer class.

10 Best Java Interview Questions - Learn What Questions the Interviewers could ask from You on Java Strings?

Java Strings are one of the most preferred topics for the Interviewers. Read the most frequently asked Java interview questions on strings.

Q: Is String a primitive data type in Java?

The String is amongst the core Java classes, and it's not related to any basic data types like int, long or float. It's a derived class which comes with the <java.lang> Package. It makes use of a character array to manage its content.

Q: Is String in Java final by default, if yes then why?

Yes, the Java designers kept it as final to enable security, optimization and to manage string pool.

Q: What difference you see between String and String Buffer classes in Java?

You must note that both the String and String Buffer are two distinct classes. The most important difference between them is that every change in a String causes the creation of a new String; the String Buffer eliminates this bottleneck. String Buffer mainly used to support the concatenation of Strings.

Q: What do you understand by the string constant pool?

It belongs to a section of the memory that holds the string objects created using string literals. This pool doesn't allow any two string objects pointing to the same content.

JVM ensures no two string objects have duplicate content. If it receives any such request, it returns the reference of the object matching the content instead of creating the new one.

Q: Explain the difference between mutable and immutable objects?

Immutable objects work like constants. It means that they don't accept any changes once created.

They are final by design. On the contrary, the mutable objects allow modifications to them.

Q: Which of these classes are final?

String, StringBuffer, and StringBuilder.

All of these classes are final.

Q: What is the no. of objects getting created in the below Java code snippet?

String first = "TechBeamers";

String second = "TechBeamers";

The above code would only lead to the creation of one String object in the String constant pool as both the strings are referring to the same content.

Q: How would you create mutable string objects?

We can use String Buffer and String Builder classes for creating the mutable strings.

Q: What do you understand string intern?

Every string object in the string constant pool is termed as String Intern. You can create a replica of objects lying in string constant pool. We call this process as interning. Java provides <intern()> method for this purpose.

Q: What difference you see between String and StringBuffer classes?

First, let's see the similarity, which is both the String and StringBuffer class are thread-safe. Next, the primary difference between them is that String objects are immutable while the String Buffer objects are mutable.

Java Interview Questions - Don't Miss to Read 10 Must-Know Questions on Java Threads

[Java threading](https://www.techbeamers.com/java-multithreading-with-examples/" \t "_blank)/synchronization are the most complex areas for the Software testers. We recommend going through the below list of 10 essential Java interview questions for QA engineers.

Q: How do you define a Thread in Java?

The thread is the smallest unit of execution in a program. It optimally consumes the CPU and improves the performance of the application. Some of the characteristics of Java threads are as follows.

These are lightweight Java process.

The thread class is a part of the <java.lang> package.

You can create many threads in java, and even the main method runs on a thread.

Java supports the concurrent execution of multiple threads.

Threads manage their stack.

Q: What difference you see between Process and Thread in Java?

A single process can own multiple threads.

Threads are the smaller execution units of a Process.

Processes possess their individual copy of the data segment of the parent process while the threads share the data portion of its process.

Threads share the address space of the process, whereas processes have their respective addresses.

Processes can easily communicate with child processes using IPC (interprocess communication). While, the threads communicate with other threads of the same process using the wait(), notify(), notifyAll() methods.

Q: What are different ways to create Threads in Java?

There are two ways to create Threads, i.e., by implementing the <java.lang.Runnable> interface or by extending the <java.lang.Thread> class and then defining the run method.

e.g.

public class ImplementThreadByExtend extends Thread{

public void run(){

// Write your code here.

// Your code will get run by a new thread.

}

public static void main(String [] args){

ImplementThreadByExtend th = new ImplementThreadByExtend();

th.start();

}

}

public class ImplementThreadByRunnable implements Runnable{

public void run(){

//...

}

public static void main(String [] args){

ImplementThreadByRunnable th = new ImplementThreadByRunnable();

Thread thread = new Thread(th);

thread.start();

}

}

Q: How would you differentiate between a thread starting with the run() and start() method?

While using the start() method, the main thread internally invokes the run() method to launch newly created Thread.

When you invoke the run() method, the main thread starts the run() method by itself.

Q: What is the significance of using the <volatile> keyword?

Java permits threads to retrieve shared variables. If you declare a field as volatile, the Java memory model ensures that all threads see a consistent value of the variable.

Q: Does Java support the <volatile> methods?

No, <volatile> is a standard keyword which you can only use for variables.

**Q: Does Java enables support of synchronized variable?**

No, the synchronized keyword can be used only with methods, i.e., in the method declaration.

Q: Is it permissible to restart a Thread?

No, you can't start a thread again. If you do so, the system will throw the runtimeException <java.lang.IllegalThreadStateException>. It occurs because the thread goes into the absolute state after executing the run() method.

Q: What do you make of the daemon threads?

These are low priority threads that run intermittently in the background for the purpose of garbage collection.

Q: What is the process of Synchronization, and why is it used?

Synchronization is a way of managing the access of shared resources in such a manner that not more than one thread could lock a particular resource at a given time.

It is helpful in protecting your application resources when multiple threads are accessing them and may change their state that could lead to data corruption. Java supports it using the <synchronized> construct.