Theory Questions

Why are we using static keyword in main method of java?

In any Java program, the main() method is the starting point from where compiler starts program execution. So, the compiler needs to call the main() method. The main() method is declared static so that JVM can call it without creating an instance of the class containing the main() method. We must declare the main() function static as no class object is present when the java runtime starts.

Strings are immutable in java. Why?

The String is immutable in Java because of the security, synchronization and concurrency, caching, and class loading. The String objects are cached in the String pool, and it makes the String immutable. String pool is possible because String is immutable in Java. It increases security because any hacker can’t change its value and it’s used for storing sensitive information such as a database username or password.

Why is constructor required? Can we make constructor static? Can we create object to constructor?

Constructor in java is used to create the instance of the class. No, we cannot define a static constructor in Java, if we are trying to define a constructor with the static keyword a compile-time error will occur. A constructor will be used to assign initial values for the instance variables. We can create object with constructor like object creation, starting a thread, calling a method, etc. You can perform any operation in the constructor as you perform in the method.

Difference between abstract/interface. When to use them?

Abstract class can have abstract and non-abstract methods. Abstract class doesn't support multiple inheritance. Abstract class can have final, non-final, static, and non-static variables. Abstract class can provide the implementation of interface.

Interface can have only abstract methods. Since Java 8, it can have default and static methods also. Interface supports multiple inheritance. Interface has only static and final variables. Interface can't provide the implementation of abstract class.

An abstract class is a good choice if we are using the inheritance concept since it provides a common base class implementation to derived classes. An abstract class is also good if we want to declare non-public members. In an interface, all methods must be public. If we want to add new methods in the future, then an abstract class is a better choice.

If the functionality we are creating will be useful across a wide range of disparate objects, use an interface. Abstract classes should be used primarily for objects that are closely related, whereas interfaces are best suited for providing a common functionality to unrelated classes.

Interfaces are a good choice when we think that the API will not change for a while. Interfaces are also good when we want to have something like multiple inheritances since we can implement multiple interfaces. If we are designing small, concise bits of functionality, use interfaces. If we are designing large functional units, use an abstract class.

What is polymorphism in java? Can we override static methods? Can we perform static method in overloading?

Polymorphism in Java is a concept by which we can perform a single action in different ways. There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

We can't override static methods since method overriding relies on dynamic binding at runtime, but static methods are bonded at compile time with static binding. As a result, we are unable to override static methods.

We can overload static methods. We can have two or more static methods with the same name, but differences in input parameters.

What are primitive datatypes and objects? Which will give better performance and why we need to use them?

The primitive data types are the predefined data types of Java. They specify the size and type of any standard values. Java has 8 primitive data types namely byte, short, int, long, float, double, char and boolean.

These are also referred to as Non-primitive or Reference Data Type. They are so-called because they refer to any particular object. Unlike the primitive data types, the non-primitive ones are created by the users in Java. Examples include arrays, strings, classes, interfaces etc.

Creating an object using new isn’t very efficient because new will place objects on the heap. This approach would be very costly for small and simple variables. Instead of create variables using new, Java can use primitive types to create automatic variables that are not references. The variables hold the value, and it’s place on the stack so its much more efficient.

How to create a custom immutable class and custom exception in java?

* The class must be declared as final so that child classes can’t be created.
* Data members in the class must be declared private so that direct access is not allowed.
* Data members in the class must be declared as final so that we can’t change the value of it after object creation.
* A parameterized constructor should initialize all the fields performing a deep copy so that data members can’t be modified with an object reference.
* Deep Copy of objects should be performed in the getter methods to return a copy rather than returning the actual object reference)

We can create our own exceptions that are derived classes of the Exception class. Creating our own Exception is known as custom exception or user-defined exception.

public class WrongFileNameException extends Exception {

public WrongFileNameException(String errorMessage) {

super(errorMessage);

}

}

Explain flow of execution of java program

The class files generated by the compiler are independent of the machine or the OS, which allows them to be run on any system. To run, the main class file (the class that contains the method main) is passed to the JVM and then goes through three main stages before the final machine code is executed. These stages are:

These states do include:

1. ClassLoader
2. Bytecode Verifier
3. Just-In-Time Compiler

ClassLoader

The main class is loaded into the memory bypassing its ‘.class’ file to the JVM, through invoking the latter. All the other classes referenced in the program are loaded through the class loader.

Bytecode Verifier

After the bytecode of a class is loaded by the class loader, it must be inspected by the bytecode verifier, whose job is to check that the instructions don’t perform damaging actions. The following are some of the checks carried out:

* Variables are initialized before they are used.
* Method calls match the types of object references.
* Rules for accessing private data and methods are not violated.
* Local variable accesses fall within the runtime stack.
* The run-time stack does not overflow.
* If any of the above checks fail, the verifier doesn’t allow the class to be loaded.

JIT (Just-In-Time) Compiler

This is the final stage encountered by the java program, and its job is to convert the loaded bytecode into machine code. When using a JIT compiler, the hardware can execute the native code, as opposed to having the JVM interpret the same sequence of bytecode repeatedly and incurring the penalty of a relatively lengthy translation process. This can lead to performance gains in the execution speed unless methods are executed less frequently.

Difference between comparator and comparable

The Comparator provides multiple sorting sequences. In other words, we can sort the collection based on multiple elements such as id, name, and price etc. Comparator doesn't affect the original class, i.e., the actual class is not modified. Comparator provides compare() method to sort elements. A Comparator is present in the java.util package. We can sort the list elements of Comparator type by Collections. Sort(List, Comparator) method.

Comparable provides a single sorting sequence. In other words, we can sort the collection based on a single element such as id, name, and price. Comparable affects the original class, i.e., the actual class is modified. Comparable provides compareTo() method to sort elements. Comparable is present in java.lang package. We can sort the list elements of Comparable type by Collections. Sort(List) method.

How to resolve deadlocks in multithreading environment

* Avoid Nested Locks: A deadlock mainly happens when we give locks to multiple threads. Avoid giving a lock to multiple threads if we already have given to one.
* Avoid Unnecessary Locks: We can have a lock only those members which are required. Having a lock unnecessarily can lead to a deadlock.
* Using Thread.join(): A deadlock condition appears when one thread is waiting other to finish. If this condition occurs, we can use Thread.join() with the maximum time the execution will take.