1. **Why we use static in main method?**

Ans) The **static** is a keyword which we use in the main() method to define it as static. There is no object of the class available at the time of starting java runtime, and because of that, we have to define the main() method as **static**.

By doing that, JVM can load the class into the main memory and call the main() method. So, if we define main() method as non-static method, JVM would not be able to call it and throws the following error.

2)**why String are immutable in java?**

Ans) The **String is immutable** in [Java](https://www.javatpoint.com/java-tutorial) because of the security, synchronization and concurrency, caching, and class loading. The reason of making string final is to destroy the immutability and to not allow others to extend it.

The String objects are cached in the String pool, and it makes the [String immutable](https://www.javatpoint.com/immutable-string). The cached String literals are accessed by multiple clients. So, there is always a risk, where action performs by one client affects all other clients.

3)**why constructor is required can we make a constructor to static? can we create a objects to the constructor?**

Ans) We know static keyword belongs to a class rather than the object of a class. A constructor is called when an object of a class is created, so no use of the static constructor. Another thing is that if we will declare static constructor then we cannot access/call the constructor from a subclass. Because we know static is allowed within a class but not by a subclass.

**A constructor does not create an object**. It just initializes the state of the object. It's the new operator which creates the object.

4)**what is the main difference between abstract and interface. when we need to use this?**

Ans) Abstract

1. Can only have final static variables. An interface can never change its own state.

2. A class can implement multiple interfaces.

3. Can be implemented with the implements keyword. An interface can also extend interfaces.

Interface:

1. Can have any kind of instance or static variables, mutable or immutable.
2. A class can extend only one abstract class.
3. Can only be extended.

5)**what is a polymorphism can be override static methods. Can we overload in static methods?**

Ans) **No, we cannot override static methods** because method overriding is based on dynamic binding at runtime and the static methods are bonded using static binding at compile time. So, we cannot override static methods.

We can overload static methods. But remember that the method signature must be different.

6)**what are primitive datatype and objects which will give the better performance why we use it?**

Ans) **Primitive data types are predefined.** **Object data types are user-defined**. These data types are held in a stack. In these data types, the original object is kept in the heap, and the reference variable is kept in the stack.

7)**how to create a customer immutable and customer exception?**

**Ans)**  Immutable objects are those objects whose states cannot be changed.

 we have to **extend the java.lang.Exception class**. Note that we also have to provide a constructor that takes a String as the error message and called the parent class constructor. This is all we need to do to define a custom exception.

8)**explain the flow of java programming?**

Ans) Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements.

9**) Comparable and Comparator differences?**

**Ans) comparable:**

1. a **single sorting sequence**.
2. **affects the original class**, i.e., the actual class is modified.
3. **compareTo() method** to sort elements.

**Comparator:**

1. **multiple sorting sequences**.
2. **doesn't affect the original class**, i.e., the actual class is not modified.
3. **compare() method** to sort elements.

10**)how to avoid deadlocks in multithreading environment**?

Ans) The [multithreading](https://www.javatpoint.com/multithreading-in-java) environment allows us to run multiple threads simultaneously for multitasking. Sometimes the threads find themselves in the waiting state, forever that is a **deadlock** situation. The **deadlock** is a situation when two or more threads try to access the same object that is acquired by another thread.