1.Why we use static key in a main method?

The **static keyword** in Java is mainly used for memory management. The static keyword in Java is used to share the same variable or method of a given class.

The reason the main() method is marked static so that it can be invoked by JVM without the need of creating an object.

2.Why strings are immutable in java?

* The String pool cannot be possible if String is not immutable in Java.
* If we don’t make the String immutable, it will pose a serious security threat to the application.
* The String is safe for multithreading because of its immutableness.
* Immutability gives the security of loading the correct class by Class loader.

3.Why constructors is required? Can we make constructor static ? Can we create objects to the constructor?

A constructor in Java is a **special method** that is used to initialize objects.

No,A static constructor is not allowed in Java programming. when we declare it as static, it can not be used by its subclasses other than the declaring classes.

Yes, We can create objects to the constructor.

The constructor is called when an object of a class is created. It can be used to set initial values for object attributes.

4.What is the main difference between abstract and interface? When to choose abstract and interface?

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 7) An **abstract class** can be extended using keyword "extends". | An **interface** can be implemented using keyword "implements". |
| 8) A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |

Interface is used when you only want to declare which methods and members a class must have. Anyone implementing the interface will have to declare and implement the methods listed by the interface.

If you also want to have a default implementation, use abstract class. Any class extending the abstract class will have to implement only its abstract methods and members, and will have some default implementation of the other methods of the abstract class, which you may override or not.

5.What is a polymorphism ? Can we override with static methods?  
Can we perform the static method with overloading?  
Polymorphism is a concept in object-oriented programming that allows objects of different classes to be treated as if they were objects of the same class.  
It is the ability of an object to take on many forms..

No,we cannot override static methods in Java.When a method is marked as static, it means that it is associated with the class itself, rather than with any particular instance of the class.ie,cannot be overridden on a per-instance basis.

Yes, we can perform method overloading with static methods in Java. Method overloading allows us to define multiple methods with the same name in a class, but with different parameters.We can overload static methods just like we can overload instance methods. The JVM determines which method to call based on the number and type of arguments passed to the method.

6. What are primitive data types and objects?. Which will give better performance?

Primitive data types are the most basic data types in Java, and they are used to represent simple values like numbers and characters.  
There are eight primitive data types in Java:  
byte: 8-bit integer  
short: 16-bit integer  
int: 32-bit integer  
long: 64-bit integer  
float: 32-bit floating point number  
double: 64-bit floating point number  
char: 16-bit Unicode character  
boolean: true or false

In Java, objects are instances of classes or arrays. An object is a self-contained unit that contains data and methods that manipulate that data.   
Each object has a unique identity and can interact with other objects through their methods and fields.  
To create an object in Java, you first need to define a class.

Java developers work with two kinds of values: primitives and objects.

Primitives offer better performance, because they are typically inlined—stored directly (without headers or pointers) in variables, on the computation stack, and, ultimately, in CPU registers.

7.How do we create a custom immutable and custom exception?

Creating Custom Immutable Class:  
An immutable class is a class whose objects cannot be modified after they are created. To create a custom immutable class, you need to follow these steps:

* Make the class final, so it cannot be extended.  
  Make all the fields private and final.  
  Do not provide any setter methods. Instead, provide getter methods to access the values of the fields.
* Ensure that any mutable objects used as fields are not modified after they are initialized.

Creating Custom Exceptions:  
Custom exceptions can be used to handle specific error cases in your code. To create a custom exception,

* you need to create a class that extends the built-in "Exception" or "RuntimeException" class.

8.Can you explain the execution/flow of java program?

There are three execution phases of a program. They are written, compile and run the program.

* Writing a program is done by a java programmer like you and me.
* The compilation is done by the JAVAC compiler which is a primary Java compiler included in the Java development kit (JDK). It takes the Java program as input and generates bytecode as output.
* In the Runningphase of a program,JVM executes the bytecode generated by the compiler.

9.Difference between Comparable and Comparator ?

|  |  |
| --- | --- |
| 1) Comparable provides a **single sorting sequence**. | The Comparator provides **multiple sorting sequences** |
| 2) Comparable **affects the original class**, i.e., the actual class is modified. | Comparator **doesn't affect the original class**, i.e., the actual class is not modified. |
| 3) Comparable provides **compareTo() method** to sort elements. | Comparator provides **compare() method** to sort elements. |
| 4) Comparable is present in **java.lang** package. | A Comparator is present in the **java.util** package. |
| 5) We can sort the list elements of Comparable type by **Collections.sort(List)** method. | We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method. |

10.How to avoid deadlocks in multithreading environment?

* **Avoid Unnecessary Locks:** We should use locks only for those members on which it is required.
* **Avoid Nested Locks:** Another way to avoid deadlock is to avoid giving a lock to multiple threads if we have already provided a lock to one thread. Since we must avoid allocating a lock to multiple threads.
* **Using Thread.join() Method:** You can get a deadlock if two threads are waiting for each other to finish indefinitely using thread join. If your thread has to wait for another thread to finish, it's always best to use join with the maximum time you want to wait for the thread to finish.
* **Use Lock Ordering:** Always assign a numeric value to each lock. Before acquiring the lock with a higher numeric value, acquire the locks with a lower numeric value.
* **Lock Time-out:** We can also specify the time for a thread to acquire a lock. If a thread does not acquire a lock, the thread must wait for a specific time before retrying to acquire a lock.