**Java Questions**

**1. If we define data member as final, is there any use of static keyword for that data member?**

Yes, it will become a constant variable.

When you combine the final and static keywords, you create a variable that is constant and global to the class. This means that the value of the variable cannot be changed, and it can be accessed from anywhere in the program, regardless of whether an instance of the class has been created.

There are a few reasons why you might want to use a static final data member in Java.

Memory efficiency:

Static variables are allocated memory only once, during the execution of the program. This can result in significant memory savings for large programs.

Improved performance:

Because static variables are associated with the class rather than with individual instances, they can be accessed more quickly and efficiently than non-static variables.

Global accessibility:

Static variables can be accessed from anywhere in the program, regardless of whether an instance of the class has been created.

**2. Difference b/w Method Overloading & Overriding in java?**

The main difference between overloading and overriding is that overloading occurs when methods in the same class have the same method name but different parameters and return type may be same or different, whereas overriding occurs when two methods in two different class have the same method name and parameters but return type must be same or covariant (subclass of the object returned by the super class method).

**3. Difference b/w Abstract class 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. |
| 9)**Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

**4. What is Upcasting and Downcasting in Java?**

Upcasting and Downcasting are two types of casting in Java. Casting is a mechanism that allows you to convert a data type of one object to another. Upcasting is a type of casting that converts a subclass object to a superclass object It's also known as "Widening." . Downcasting is a type of casting that converts a superclass object to a subclass object This is also termed as "Narrowing."

**5. What is Binding in java?**

In Java, binding is the process of associating a method call to the method body. There are two types of binding in Java:

* Static binding,

also known as early binding, occurs when the type of an object is determined at compile time. Static binding occurs when there is a private, final, or static method in a class.

Static binding is used for: Private methods, Final methods, Static methods, and Variables.

* Dynamic binding,

also known as late binding, occurs when the type of an object is determined at runtime. Dynamic binding occurs when there is a virtual method in a class.

Dynamic binding is used for: Virtual methods.

**6. Why String is Immutable in java?**

When we create a string in Java like String s1="hello"; then an object will be created in **string pool(hello)** and **s1** will be pointing to **hello**. Now if again we do String s2="hello"; then another object will not be created, but **s2** will point to hello because **JVM** will first check if the same object is present in **string pool** or not. If not present, then only a new one is created, else not.

The **String is immutable** in Java because of the security, synchronization and concurrency, caching, and class loading.

**7. If we use new keyword it will create in heap memory not in SCP? Yes/No.**

Yes.

**8. Difference b/w String, StringBuilder & StringBuffer?**

* String: Immutable, meaning it can't be changed after it's declared.
* StringBuilder: Used to create mutable strings, but it's not thread-safe.
* StringBuffer: Used to create mutable strings, and it's thread-safe.
* Synchronization: StringBuffer is synchronized by default, while StringBuilder is not.
* Thread-safe: StringBuffer is thread-safe, while StringBuilder is not.
* Speed: StringBuilder is faster than StringBuffer.
* Memory consumption: StringBuffer consumes more memory than StringBuilder.

**9**. **What is polymorphism?**

**Polymorphism in Java** is a concept by which we can perform a single action in different ways. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

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.

**10**. **How can we create a thread, which is the best one and why?**

In Java, there are two ways to create a thread:

* Extending the Thread class

Create a thread by creating a new class that extends the Thread class and creating an instance of that class. The extending class must override the run() method, which is the entry point of the new thread.

* Implementing the Runnable interface

Create a class that implements the runnable interface, and then implement the run() method.

The Runnable interface approach is recommended because it only creates one instance of a class, which is shared by different threads. The Thread class approach requires creating a separate instance for every thread access.

**11. What is class level lock & object level lock?**

In Java, there are two types of locks: object level locks and class level locks.

Object level locks are used to synchronize access to individual objects. When a thread acquires an object level lock, it prevents other threads from accessing the object until the lock is released. Object level locks are typically used to protect shared data that is accessed by multiple threads.

Class level locks are used to synchronize access to all instances of a class. When a thread acquires a class level lock, it prevents other threads from accessing any instance of the class until the lock is released. Class level locks are typically used to protect static data that is shared by all instances of a class.

**12. What is DeadLockSituation & Synchronization?**

Deadlock in Java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Since, both threads are waiting for each other to release the lock, the condition is called deadlock.

**Here are some tips for avoiding deadlocks in Java:**

* Avoid locking multiple resources in different orders.
* Use timeouts when waiting for locks.
* Use lock hierarchies to avoid circular dependencies.
* Use semaphores to control the number of threads that can access a shared resource at the same time.

Synchronization is the process of controlling access to shared resources. This is important in multithreaded programming, where multiple threads are trying to access the same resources at the same time. Synchronization can be used to prevent deadlocks, as well as other problems such as data corruption. One common approach is to use the synchronized keyword.

**13. Difference b/w ArrayList & LinkedList?**

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList internally uses a **dynamic array** to store the elements. | LinkedList internally uses a **doubly linked list** to store the elements. |
| 2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory. |
| 3) An ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| 4) ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |
| 5) The memory location for the elements of an ArrayList is contiguous. | The location for the elements of a linked list is not contagious. |
| 6) Generally, when an ArrayList is initialized, a default capacity of 10 is assigned to the ArrayList. | There is no case of default capacity in a LinkedList. In LinkedList, an empty list is created when a LinkedList is initialized. |
| 7) To be precise, an ArrayList is a resizable array. | LinkedList implements the doubly linked list of the list interface. |

When the rate of addition or removal rate is more than the read scenarios, then go for the LinkedList. On the other hand, when the frequency of the read scenarios is more than the addition or removal rate, then ArrayList takes precedence over LinkedList.

**14. Difference b/w Map & Set?**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Set** | **Map** |
| 1. | Set is used to construct the mathematical Set in Java. | Map is used to do mapping in the database. |
| 2. | It cannot contain repeated values. | It can have the same value for different keys. |
| 3. | Set doesn't allow us to add the same elements in it. Each class that implements the Set interface contains only the unique value. | Map contains unique key and repeated values. In Map, one or more keys can have the same values, but two keys cannot be the same. |
| 4. | We can easily iterate the Set elements using the keyset() and the entryset() method of it. | Map elements cannot be iterated. We need to convert Map into Set for iterating the elements. |
| 5. | Insertion order is not maintained by the Set interface. However, some of its classes, like LinkedHashSet, maintains the insertion order. | The insertion order is also not maintained by the Map. However, some of the Map classes like TreeMap and LinkedHashMap does the same. |

**15. Add list to Set?**

Set<String> s= new Hashset<String>(list);

OR

Set<String> s= new Hashset<String>();

For(String x : list)

s.add(x);

**16. what is Singleton, How to make a class Singleton?**

A singleton is a design pattern that restricts the instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system. The singleton pattern is often used in multithreaded applications, where it can be used to ensure that only one thread accesses a shared resource at a time.

**To make a class singleton, you can follow these steps:**

Declare a private static variable of the class type.

Create a private constructor.

Create a public static method that returns the instance of the class. This method should check if the instance has already been created. If it has not, the method should create the instance and store it in the private static variable. If the instance has already been created, the method should simply return the instance.

**17. what is Encapsulation? how to make a class Encapsulation?**

Encapsulation is a process of wrapping data and code together into a single unit, called a class. This bundling of data and code together helps to protect the data from being accessed by unauthorized code. It also helps to make the code more reusable and maintainable.

To make a class encapsulated, you need to declare the data members of the class as private. This means that the data members can only be accessed by the methods of the class. You then need to provide public methods that allow other classes to access and modify the data members. These methods are called getter and setter methods.

**18. what is comparator & comparable?**

Comparable and Comparator are interfaces in the Java programming language that determine how classes are implemented. They are used to sort collections of objects.

|  |  |
| --- | --- |
| **Comparable** | **Comparator** |
| 1) Comparable provides a **single sorting sequence**. In other words, we can sort the collection on the basis of a single element such as id, name, and price. | The Comparator provides **multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc. |
| 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. |

**19. Difference between final, finally and finalize method in java?**

Final is a keyword that can be used to declare a variable, method, or class as final. This means that the value of a final variable cannot be changed, a final method cannot be overridden by a subclass, and a final class cannot be extended.

Finally is a keyword that is used to create a finally block. A finally block is a block of code that is always executed, regardless of whether an exception is thrown. This is useful for ensuring that resources are properly cleaned up, even if an error occurs.

Finalize is a method that is called by the garbage collector before an object is destroyed. This method can be used to perform any necessary cleanup tasks, such as closing files or releasing resources.

**20. What is meant by Exception?**

**In Java, Exception** is an unwanted or unexpected event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program’s instructions. Exceptions can be caught and handled by the program. When an exception occurs within a method, it creates an object. This object is called the exception object. It contains information about the exception, such as the name and description of the exception and the state of the program when the exception occurred.

21**. What are the types of Exception?**

There are two types of exceptions in Java: checked exceptions and unchecked exceptions.

**22. What are Checked and Unchecked Exceptions?**

Checked exceptions are the exceptions that are checked by the compiler at compile time. If a method throws a checked exception, then the caller of the method must either handle the exception or declare it in the throws clause.

Unchecked exceptions are the exceptions that are not checked by the compiler at compile time. They include runtime exceptions and errors.

**Here are some examples of checked exceptions:**

* FileNotFoundException: This exception is thrown when a file cannot be found.
* IOException: This exception is thrown when an input or output operation fails.
* ClassNotFoundException: This exception is thrown when a class cannot be found.
* SQLException: This exception is thrown when a database operation fails.

**Here are some examples of unchecked exceptions:**

* NullPointerException: This exception is thrown when a null reference is used.
* ArrayIndexOutOfBoundsException: This exception is thrown when an array index is out of bounds.
* ArithmeticException: This exception is thrown when an arithmetic operation fails.
* IllegalArgumentException: This exception is thrown when an illegal argument is passed to a method.

**23. How can you create Customized Exception?**

In order to create custom exception, we need to extend Exception class that belongs to java.lang package.

Consider the following example, where we create a custom exception named WrongFileNameException:

**public** **class** WrongFileNameException **extends** Exception {

**public** WrongFileNameException(String errorMessage) {

**super**(errorMessage);

     }

}

**24. Throw & Throws keyword in Exception?**

In Java, the throw keyword is used to explicitly throw an exception from a method or block of code, while the throws keyword is used to declare that a method might throw an exception. The throw keyword is used inside a method, while the throws keyword is used in the method signature. The throw syntax is followed by an instance of exception to be thrown, while the throws syntax is followed by the class names of exceptions to be thrown.

**25. What are the new features added in java 8.**

Lambda expressions:

Lambda expressions are a new way to write concise and readable code. They allow you to create anonymous functions that can be passed as arguments to other methods.

Functional interfaces:

Functional interfaces are interfaces that have only one abstract method. They are used to represent functions that can be passed as arguments to other methods.

Stream API:

The Stream API is a new way to process collections of data. It provides a number of operations that can be used to filter, map, reduce, and collect data.

Optional class:

The Optional class is a new way to represent optional values. It provides a number of methods that can be used to check if a value is present, get the value if it is present, or get a default value if the value is not present.

Date and time API:

The Date and time API is a new way to represent and manipulate dates and times. It provides a number of classes and methods that make it easier to work with dates and times.

Default methods:

Default methods are methods that can be added to interfaces. They allow you to add new functionality to interfaces without breaking existing code.

**26. What is SpringBoot?**

Spring Boot is an open-source Java-based framework that is used to create stand-alone, production-grade Spring-based Applications that you can "just run". It is built on the Spring Framework and provides a number of features that make it easier to get started with Spring, such as:

Embedded servers:

Spring Boot includes embedded servers such as Tomcat, Jetty, and Undertow, so you don't need to deploy your application to a separate server.

Auto-configuration:

Spring Boot can automatically configure your application based on the dependencies that you have added to your classpath. This means that you don't need to write a lot of boilerplate code to get your application up and running.

Starters:

Spring Boot provides a number of starters, which are pre-configured sets of dependencies that you can add to your application to get started with specific features, such as web development, data access, and messaging.

**27. What are the Annotations used in SpringBoot?**

* @SpringBootApplication

This annotation is used to mark the main class of a Spring Boot application. It encapsulates @Configuration, @EnableAutoConfiguration, and @ComponentScan annotations with their default attributes.

* @Bean

This annotation indicates that a method produces a bean that is managed by the Spring container.

* @Service

This annotation indicates that the annotated class is a service class. Service classes are responsible for business logic.

* @Repository

This annotation indicates that the annotated class is a repository. Repositories are responsible for interacting with data stores.

* @Controller

This annotation indicates that the annotated class is a controller class. Controller classes are responsible for handling HTTP requests.

* @RestController

This annotation combines the functionality of @ResponseBody and @Controller annotations. It tells Spring Boot to return the return value of a method as the response body of an HTTP request.

* @Autowired

This annotation tells Spring Boot to automatically inject the required dependencies into a bean.

@RequestMapping

annotation is used to map incoming HTTP requests to specific handler methods in Spring Boot.

* @Qualifier

This annotation is used to qualify a bean when there are multiple beans of the same type.

* @PropertySource

This annotation tells Spring Boot to load properties from a file or an environment variable.

* @Value

This annotation indicates that a field or parameter should be injected with a value from a property source.

**28. What are the Http methods used?**

**i) GET**

This method retrieves information from the given server using a given URI. GET request can retrieve the data. It can not apply other effects on the data.

**iii) POST**

The POST request sends the data to the server. For example, file upload, customer information, etc. using the HTML forms.

**iv) PUT(update)**

The PUT method is used to replace all the current representations of the target resource with the uploaded content.

**v) DELETE**

The DELETE method is used to remove all the current representations of the target resource, which is given by URI.

**29. What is diff b/w @pathvariable & @requestparam?**

The main difference between @PathVariable and @RequestParam is that @PathVariable is used to extract values from the URI path, while @RequestParam is used to extract values from the query string.

**Here are some more details about each annotation:**

* @PathVariable
  + Used to extract values from the URI path.
  + Values are enclosed in curly braces ({}).
  + For example, in the URI /users/{id}, the id value would be extracted using @PathVariable.
  + @PathVariable is required, meaning that the URI path must contain a value for the variable.
* @RequestParam
  + Used to extract values from the query string.
  + Values are appended to the URI after a question mark (?).
  + For example, in the URI /users?id=123, the id value would be extracted using @RequestParam.
  + @RequestParam is optional, meaning that the query string may or may not contain a value for the variable.

**30. Sql Query to find 2nd highest salary?**

select \* from employee

group by salary

order by salary desc limit 1,1;