**Created By Vijayalayan Prabhakaran**

1.What is java?

**Java is a programming language and computing platform first released by Sun Microsystems in 1995. It is designed to have as few implementation dependencies as possible, allowing developers to "write once, run anywhere" (WORA). Java applications are typically compiled to bytecode (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture. It is widely used for building enterprise software and mobile applications, as well as video games and other interactive content.**

2.What are classes and methods and members variables in java

**In Java, a class is a blueprint for an object, which defines its properties (also called member variables or attributes) and methods.**

**A method is a block of code that performs a specific task, and can be called (invoked) by other parts of the program.**

**A member variable (also called a field or attribute) is a variable that is associated with an instance of a class, and stores the state of the object.**

3.What are constructors in java

**In Java, a constructor is a special type of method that is automatically called when an object of a class is created. It is used to initialize the state of the object, and can also be used to perform any other setup that the object needs before it can be used.**

**Constructors have the same name as the class, and do not have a return type. They can also have different parameters, which allows for multiple ways to create an object. For example, a class called "Person" could have constructors that take different sets of arguments, such as a name and age, or just a name.**

**The default constructor is a constructor with no argument,if we don't define any constructor in the class, then the compiler will provide a default constructor with no arguments, which will call the super class constructor with no argument**

**Java also provides a special constructor called the "copy constructor", which creates a new object and copies the state of an existing object into it.**

**4.Explain the 4 OOPS concepts in java-Polymorphism,inheritance,Encapsulation and abstraction**

**Polymorphism: It is the ability of an object to take on multiple forms. In Java, polymorphism allows a single variable or method to refer to objects of different types at different times. This is achieved through method overriding and method overloading.**

**Inheritance: It is the mechanism by which one class acquires the properties and methods of another class. In Java, a class can inherit from a single superclass using the "extends" keyword. This allows for a hierarchy of classes, where a subclass inherits the characteristics of its parent class, and can also add new properties and methods of its own.**

**Single Inheritance: A class inherits the properties and methods of a single parent class.**

**Multi-level Inheritance: A class inherits the properties and methods of a parent class and that parent class also inherits properties and methods from its parent class and so on.**

**Hierarchical Inheritance: Multiple classes inherit the properties and methods of a single parent class.**

**Multiple Inheritance: A single class inherits the properties and methods of multiple classes. However, Java does not support multiple inheritance directly, but it can be achieved by interfaces.**

**Hybrid Inheritance: A combination of two or more types of inheritance.**

**Encapsulation: It is the process of hiding the internal details of an object from the outside world. In Java, encapsulation is achieved through the use of private variables and methods, which can only be accessed by other members of the same class. This helps to protect the internal state of an object and ensures that it can only be modified in a controlled way.**

**Abstraction: It is the process of abstracting away the complexity of an object to reveal its essential features. In Java, abstraction is achieved through the use of interfaces and abstract classes. An interface defines a set of methods that a class must implement, while an abstract class provides a common set of properties and methods that its subclasses can inherit and extend. This allows for a simplified view of an object, and helps to reduce the complexity of the code.**

**4.What is method overriding and method overloading**

**Method Overriding: It is a feature that allows a subclass to provide a specific implementation of a method that is already provided by its superclass. When a method in a subclass has the same name, return type and arguments as a method in its superclass, it is said to be overriding the method in the superclass. The use of the @override annotation is optional but it's a good practice to use it. The overriding method must have the same method signature as the method in the superclass and it must be at least as accessible or more accessible than the method it overrides.**

**Method Overloading: It is a feature that allows a class to have multiple methods with the same name but different arguments. These methods are said to be overloaded. Method overloading allows a class to have multiple methods with the same name but different argument lists. This feature is also called method polymorphism. The overloading method must have a different method signature, i.e different number of parameters or different type of parameters or both.**

**The key point to note is that in method overriding the method should have the same method signature and return type and the subclass should have the same or more accessibility but in method overloading the method should have the same name but different argument list.**

**5.What are static methods?explain static keyword in Java**

**This means that it can be called on the class itself, rather than on a specific object of that class. The static keyword is used to define a static method.**

**The main difference between a static method and a non-static (also called instance) method is that a static method does not have access to the non-static variables and methods of the class, because it does not have an instance of the class to work with. Instead, it can only access static variables and methods, which belong to the class itself.**

**A common use case for static methods is utility methods, which do not need access to the internal state of an object, and can be used without creating an instance of the class. For example, a class that contains a utility method for converting a temperature from Celsius to Fahrenheit could have a static method called "celsiusToFahrenheit", which takes a temperature in Celsius as an argument and returns the equivalent temperature in Fahrenheit.**

**Another use case for static methods is when a method needs to be shared by multiple instances of a class but does not need to access the instance data. In this case, it would be wasteful to create a new copy of the method for each object, so a static method is used instead.**

**It's also worth noting that static variables and methods are shared across all instances of a class,static fields and methods are associated with a class rather than with an object,therefore they are only created once per class, no matter how many instances of the class are created.**

**6.What is a JVM in Java?**

**Java Virtual Machine (JVM) is a virtual machine that enables a computer to run Java programs as well as programs written in other languages that are also compiled to Java bytecode. The JVM is the heart of the Java platform and provides the environment in which Java bytecode can be executed.**

**When a Java program is compiled, the source code is converted into bytecode, which is a platform-independent code that can be run on any operating system that has a JVM installed. When the Java program is executed, the JVM interprets the bytecode and converts it into machine code that can be executed by the computer's processor.**

**The JVM also provides important features such as memory management, security, and exception handling. It includes a built-in garbage collector, which automatically frees up memory that is no longer needed by the program, and a security manager, which enforces a set of rules to protect the computer from malicious code.**

**It also provides a platform-independent way of executing the code, it means the same code can run on different operating systems without modification.**

**In summary, the JVM is a key component of the Java platform that provides the environment for executing Java bytecode and provides important features such as memory management, security, and exception handling.**

**7.Explain about exception handling in java with thow,throws keyword.**

**In Java, exception handling is a mechanism for handling errors and exceptional conditions that might occur during the execution of a program. The main goal of exception handling is to separate the error-handling code from the normal code, so that the program can continue to execute even in the presence of errors.**

**The two main keywords used in Java for exception handling are "throw" and "throws".**

**The "throw" keyword is used to throw an exception, which means to create a new exception object and then immediately pass it to the runtime system. This can be done by calling the constructor of the exception class and then passing the new object to the runtime system. For example:**

**javascript**

**Copy code**

**throw new IllegalArgumentException("Invalid argument");**

**The "throws" keyword is used in a method signature to indicate that the method may throw an exception. When a method throws an exception, it means that the method cannot handle the exception itself and it must be handled by the code that calls the method. For example:**

**java**

**Copy code**

**public void readFile(String fileName) throws IOException {**

**// code to read a file**

**}**

**It's important to know that the throws keyword indicates that the method might throw an exception but it doesn't mean it will. The exception will only be thrown if an error occurs during the execution of the method.**

**When a method throws an exception, the runtime system looks for an exception handler that can handle the exception. If it finds one, it passes the exception object to the handler, otherwise, the program is terminated.**

**It's also worth noting that there's a try-catch block statement, which allows you to catch an exception thrown by a method and handle it in a specific way. The try block contains the code that may throw an exception and the catch block contains the code that handles the exception.**

**In summary, exceptions are events that occur during the execution of a program that disrupts the normal flow of instructions. Java provides a robust exception handling mechanism using throw and throws keywords, try-catch block, which allow you to handle exceptions in a controlled and predictable way, and make your code more robust and maintainable.**

**8.Explain Try,catch and finally block**

**In Java, the try-catch-finally block is used to handle exceptions that might occur during the execution of a program. The try block contains the code that may throw an exception, the catch block contains the code that handles the exception, and the finally block contains the code that is always executed, whether an exception is thrown or not.**

**The try block is used to enclose the code that may throw an exception. If an exception is thrown within the try block, the runtime system will immediately transfer the control to the appropriate catch block. For example:**

**javascript**

**Copy code**

**try {**

**// code that may throw an exception**

**}**

**The catch block is used to handle the exception that was thrown within the try block. It contains the code that is executed when an exception is thrown. The catch block is executed immediately after the exception is thrown and before the finally block. You can have multiple catch blocks to handle different types of exceptions. For example:**

**javascript**

**Copy code**

**catch (IOException e) {**

**// code to handle IOException**

**}**

**catch (NumberFormatException e) {**

**// code to handle NumberFormatException**

**}**

**The finally block is used to enclose the code that is always executed, whether an exception is thrown or not. The code in the finally block is executed after the try block and any catch blocks have completed. It is typically used to release resources that have been acquired within the try block, such as file handles, database connections, etc. For example:**

**javascript**

**Copy code**

**finally {**

**// code to release resources**

**}**

**In summary, the try-catch-finally block is a powerful mechanism for handling exceptions in Java. The try block contains the code that may throw an exception, the catch block contains the code that handles the exception, and the finally block contains the code that is always executed, whether an exception is thrown or not. This mechanism allows you to handle exceptions in a controlled and predictable way and make your code more robust and maintainable.**

**9.What is Finalize method in Java**

**In Java, the finalize() method is a protected method that belongs to the Object class. It is called by the garbage collector just before an object is garbage collected. The finalize() method is used to perform cleanup activities such as releasing resources (e.g. file handles, database connections) that the object may be holding.**

**The finalize() method is typically overridden by the subclass to include cleanup code. The overridden method should be made public and should not throw any exceptions. The method should also be called only once for an object.**

**java**

**Copy code**

**protected void finalize() throws Throwable {**

**// cleanup code here**

**}**

**It's important to note that the finalize() method is not guaranteed to be called by the garbage collector, and its use is generally discouraged in favor of cleaner and more efficient alternatives such as try-finally or try-with-resources. The finalize() method is considered as a legacy method and it's not recommended to use it as it can have a negative impact on the performance of the application.**

**In summary, the finalize() method is a protected method that belongs to the Object class, which is called by the garbage collector just before an object is garbage collected. It's typically used to perform cleanup activities such as releasing resources that the object may be holding, but it's not guaranteed to be called by the garbage collector and its use is generally discouraged in favor of cleaner and more efficient alternatives such as try-finally or try-with-resources.**

**10.What are the ways to create a custom exception**

**Extend the Exception class: You can create a custom exception by creating a new class that extends the Exception class. The new class should have a constructor that takes a string argument as the error message.**

**java**

**Copy code**

**public class MyException extends Exception {**

**public MyException(String message) {**

**super(message);**

**}**

**}**

**Extend the RuntimeException class: You can create a custom exception by creating a new class that extends the RuntimeException class. The new class should have a constructor that takes a string argument as the error message.**

**java**

**Copy code**

**public class MyRuntimeException extends RuntimeException {**

**public MyRuntimeException(String message) {**

**super(message);**

**}**

**}**

**Implement the Exception interface: You can create a custom exception by creating a new class that implements the Exception interface. The new class should have a constructor that takes a string argument as the error message.**

**typescript**

**Copy code**

**public class MyCustomException implements Exception {**

**private String message;**

**public MyCustomException(String message) {**

**this.message = message;**

**}**

**public String getMessage() {**

**return message;**

**}**

**}**

**In summary, creating a custom exception in Java can be done by creating a new class that extends the Exception class, RuntimeException class or implements the Exception interface. The new class should have a constructor that takes a string argument as the error message and can also have other methods for handling the exception.**

**11. What is multithreading**

**Multithreading is a technique in which a single process is divided into multiple threads that can run independently and concurrently. Each thread has its own instruction pointer, stack and local variables, allowing it to run its own task simultaneously with other threads. This allows for more efficient use of CPU resources and can significantly improve the performance of a program.**

**Java, as well as many other programming languages, provides support for multithreading through the Thread class and the Runnable interface. A thread in Java can be created by either extending the Thread class or implementing the Runnable interface and then calling the start() method on the new thread object.**

**The Runnable interface is more preferred way of creating a thread because it does not inherit the Thread class, but just implements the runnable interface. If a class extends the Thread class, it cannot extend any other class, but if it implements the Runnable interface it can still extend other base classes.**

**java**

**Copy code**

**public class MyThread extends Thread {**

**public void run() {**

**// code to be executed in the new thread**

**}**

**}**

**typescript**

**Copy code**

**public class MyThread implements Runnable {**

**public void run() {**

**// code to be executed in the new thread**

**}**

**}**

**Threads can communicate with each other using various synchronization mechanisms such as locks, semaphores and monitors. The synchronized keyword can be used to ensure that only one thread can execute a specific block of code at a time, preventing race conditions and other synchronization issues.**

**In summary, multithreading is a technique in which a single process is divided into multiple threads that can run independently and concurrently. This allows for more efficient use of CPU resources and can significantly improve the performance of a program. Java provides support for multithreading through the Thread class and the Runnable interface and threads can communicate with each other using various synchronization mechanisms such as locks, semaphores and monitors.**

**12.What are the methods in a thread**

**Java provides several methods in the Thread class that can be used to control and manage threads. Some of the most commonly used methods are:**

**start(): This method is used to start a new thread. It creates a new thread of execution and starts it running.**

**run(): This method contains the code that will be executed by the new thread. It is important to note that the run() method is called when the start() method is invoked, and not when the thread is instantiated.**

**sleep(long millis): This method causes the current thread to suspend execution for the specified number of milliseconds.**

**join(): This method causes the current thread to wait for the completion of another thread.**

**yield(): This method causes the current thread to temporarily pause its execution, allowing other threads to run.**

**getName(): This method returns the name of the thread.**

**setName(String name): This method sets the name of the thread.**

**getPriority(): This method returns the priority of the thread.**

**setPriority(int priority): This method sets the priority of the thread.**

**isAlive(): This method checks if the thread is still running.**

**interrupt(): This method interrupts the thread and causes it to throw an InterruptedException.**

**isInterrupted(): This method checks if the thread has been interrupted.**

**It is worth noting that, the run() method and the start() method are the two important methods for a thread. The run() method contains the code that will be executed by the thread, while the start() method is used to start the thread. The other methods are used to control and manage the thread's execution.**

**13.Ways to create a thread**

**There are two ways to create a thread in Java:**

**Extending the Thread class:**

**You can create a new thread by extending the Thread class and overriding its run() method. The run() method contains the code that will be executed by the new thread. Once you have overridden the run() method, you can create an instance of your new thread class and call its start() method to begin execution.**

**java**

**Copy code**

**public class MyThread extends Thread {**

**public void run() {**

**// code to be executed in the new thread**

**}**

**}**

**scss**

**Copy code**

**MyThread myThread = new MyThread();**

**myThread.start();**

**Implementing the Runnable interface:**

**Another way to create a thread is by implementing the Runnable interface. This interface has a single method run() that you must implement. Once you have implemented the run() method, you can create a new thread by instantiating a Thread object and passing an instance of your class to its constructor. Then call the start() method on the Thread object to begin execution.**

**typescript**

**Copy code**

**public class MyThread implements Runnable {**

**public void run() {**

**// code to be executed in the new thread**

**}**

**}**

**scss**

**Copy code**

**MyThread myRunnable = new MyThread();**

**Thread thread = new Thread(myRunnable);**

**thread.start();**

**It is worth noting that, The Runnable interface is more preferred way of creating a thread because it does not inherit the Thread class, but just implements the runnable interface. If a class extends the Thread class, it cannot extend any other class, but if it implements the Runnable interface it can still extend other base classes.**

**In summary, there are two ways to create a thread in Java: by extending the Thread class and overriding its run() method, and by implementing the Runnable interface and passing an instance of the class to a Thread object's constructor. Both methods allow you to create a new thread of execution and run code concurrently with the main thread of the program.**

**14.What is a diffence between wait and sleep method**

**wait() and sleep() are both methods in Java that are used to pause the execution of a thread. However, they are used in different situations and have some key differences.**

**wait() method :**

**The wait() method is used for inter-thread communication, and it is defined in the Object class. When a thread calls the wait() method, it releases the lock it holds on the object and goes into a wait state. It can only be resumed by another thread calling the notify() or notifyAll() method on the same object. When the thread is resumed, it must re-acquire the lock before it can continue execution.**

**sleep() method :**

**The sleep() method is used to pause the execution of the current thread for a specified amount of time. It is defined in the Thread class. When a thread calls the sleep() method, it temporarily suspends execution for the specified time, but it still retains the lock it holds on any objects. After the specified time has elapsed, the thread resumes execution where it left off.**

**In summary, The sleep() method is used to temporarily pause a thread's execution for a specified amount of time, while the wait() method is used for inter-thread communication. The sleep() method is defined in the Thread class, while the wait() method is defined in the Object class. sleep() method does not release the lock, it holds on any objects.**

**15. What is a Executor service in thread**

**In Java, the ExecutorService interface is part of the java.util.concurrent package and it is used to manage and execute a collection of threads. It provides an easy way to execute tasks concurrently and provides several useful features such as thread pooling and task scheduling.**

**An ExecutorService can be created using one of the factory methods provided by the Executors class. For example, the newFixedThreadPool(int nThreads) method creates an Executor with a fixed number of threads in its thread pool.**

**Once an ExecutorService is created, you can submit tasks for execution using one of its submit() methods. For example, the submit(Runnable task) method submits a Runnable task for execution and returns a Future object that can be used to check the status of the task and retrieve its results.**

**java**

**Copy code**

**ExecutorService executor = Executors.newFixedThreadPool(10);**

**Future<Integer> future = executor.submit(new MyTask());**

**Additionally, the ExecutorService interface also provides several other features such as:**

**Task scheduling: schedule() method can be used to execute a task at a specified time or after a specified delay.**

**Task canceling: cancel() method can be used to cancel the execution of a task.**

**Shutting down: shutdown() method can be used to terminate all running threads and release resources.**

**Waiting for completion: awaitTermination() method can be used to wait for the completion of all tasks.**

**In summary, The ExecutorService interface provides a higher-level framework for managing and executing threads. It allows you to submit tasks for execution, manage a pool of threads, schedule tasks, cancel tasks, and shut down the service when it is no longer needed. This makes it an easy and convenient way to perform concurrent operations in your application.**

**16. What is synchronised and volatile keyword in threads**

**In Java, the synchronized and volatile keywords are used to control access to shared resources in a multi-threaded environment.**

**synchronized keyword :**

**The synchronized keyword is used to control access to shared resources in a multi-threaded environment. When a method or block is marked as synchronized, only one thread can execute it at a time. This ensures that multiple threads don't access the same resource simultaneously, which can cause race conditions and other problems.**

**java**

**Copy code**

**public class MyClass {**

**private int count;**

**public synchronized void increment() {**

**count++;**

**}**

**}**

**volatile keyword :**

**The volatile keyword is used to indicate that a variable's value can be modified by multiple threads. When a variable is marked as volatile, the JVM will ensure that all threads see the latest value of the variable.**

**csharp**

**Copy code**

**public class MyClass {**

**private volatile boolean running;**

**public void start() {**

**running = true;**

**// rest of the code**

**}**

**public void stop() {**

**running = false;**

**// rest of the code**

**}**

**}**

**In summary, The synchronized keyword is used to control access to shared resources in a multi-threaded environment, while the volatile keyword is used to ensure that multiple threads see the latest value of a variable. The synchronized keyword ensures that only one thread can execute a synchronized method or block at a time, while the volatile keyword ensures that all threads see the latest value of a volatile variable.**

**JAva 8:(Very important topic all these are,Read everything)**

**17.Lambada expression**

**In Java, a lambda expression is a way to define a function or behavior as a first-class citizen, similar to a variable. It is a concise and functional way to represent a single method interface, such as Runnable or Comparator, using an expression rather than an anonymous inner class.**

**A lambda expression is defined with the following syntax:**

**css**

**Copy code**

**(parameters) -> {body}**

**For example, the following lambda expression represents a Runnable that prints "Hello, World!" when run:**

**java**

**Copy code**

**Runnable r = () -> System.out.println("Hello, World!");**

**Lambda expressions can also have multiple parameters and a return value, like a regular function. The following lambda expression represents a Comparator that compares two strings based on their length:**

**scss**

**Copy code**

**Comparator<String> comparator = (s1, s2) -> s1.length() - s2.length();**

**Lambda expressions are often used with functional interfaces, which are interfaces with a single abstract method. The Java API provides several functional interfaces, such as Runnable, Callable, Consumer, Supplier, Predicate, and Function, that can be used with lambda expressions.**

**In summary, Lambda expressions are a concise and functional way to represent a single method interface using an expression. It is a functional way to represent a piece of code as a value, similar to a variable. It allows you to pass behavior as an argument to a method or construct an instance of an interface with an anonymous inner class. With the introduction of lambda expressions, Java 8 enables functional programming constructs like closures, which were previously not possible in Java.**

**18. Stream API**

**In Java 8, the Stream API is a powerful set of tools for processing collections of data. A stream is a sequence of elements that can be processed in parallel or sequentially. The Stream API provides a functional and declarative way to manipulate collections of data, similar to the way SQL queries work on a database.**

**A stream can be created from a collection, an array, an I/O channel, or even by generating elements on the fly. Once created, a stream can be transformed, filtered, and processed in various ways. The Stream API provides several intermediate operations, such as map, filter, flatMap, distinct, sorted, and peek, that can be used to transform a stream. Terminal operations, such as forEach, count, min, max, reduce, and collect, are used to produce a final result or a side-effect.**

**For example, the following code uses a stream to filter a list of integers, square them, and find the sum:**

**scss**

**Copy code**

**List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);**

**int sum = numbers.stream()**

**.filter(n -> n % 2 == 0)**

**.map(n -> n \* n)**

**.reduce(0, (a, b) -> a + b);**

**The Stream API also provides a parallel version of streams, which can be used to perform operations in parallel and take advantage of multi-core processors. Parallel streams can be created by calling the parallelStream() method on a collection or by calling the stream().parallel() method on a stream.**

**In summary, the Stream API is a powerful set of tools for processing collections of data in Java 8. It provides a functional and declarative way to manipulate collections of data, similar to the way SQL queries work on a database. It can be used to filter, transform, and process collections of data in parallel or sequentially, with a lot of built-in intermediate and terminal operations that can be combined to perform complex operations.**

**19.Functional interface and its types**

**In Java 8, a functional interface is an interface that contains exactly one abstract method. A functional interface can have any number of default and static methods, but it must have only one abstract method. Functional interfaces are used as the target type for lambda expressions and method references.**

**Java 8 introduces several functional interfaces in the java.util.function package, such as:**

**Predicate<T>: Represents a boolean-valued function of one argument. It has a single abstract method, test(T t), that returns a boolean value.**

**Consumer<T>: Represents an operation that accepts a single input argument and returns no result. It has a single abstract method, accept(T t), that performs a void operation on the given argument.**

**Function<T, R>: Represents a function that accepts one argument and produces a result. It has a single abstract method, apply(T t), that takes a single argument of type T and returns a result of type R.**

**Supplier<T>: Represents a supplier of results. It has a single abstract method, get(), that returns a result of type T.**

**UnaryOperator<T>: Represents an operation on a single operand that produces a result of the same type as its operand. It extends the Function interface, and it has a single abstract method, apply(T t), that takes a single argument of type T and returns a result of the same type T.**

**BinaryOperator<T>: Represents an operation upon two operands of the same type, producing a result of the same type as the operands. It has a single abstract method, apply(T t1, T t2), that takes two arguments of the same type T and returns a result of the same type T.**

**For example, the following code defines a functional interface MyFunctionalInterface:**

**java**

**Copy code**

**@FunctionalInterface**

**interface MyFunctionalInterface {**

**int myMethod(int a, int b);**

**}**

**And following code shows its usage in lambda expression**

**css**

**Copy code**

**MyFunctionalInterface addition = (a,b)->a+b;**

**int result = addition.myMethod(5,3);**

**In summary, functional interfaces are a new feature in Java 8 that allow you to use lambda expressions and method references to create anonymous functions. They are interfaces that have exactly one abstract method and can be used as the target type for lambda expressions. Java 8 provides several functional interfaces in the java.util.function package for common functional interface types such as predicates, consumers, functions, suppliers, unary operators, and binary operators.**

**20.Method reference**

**Method references in Java 8 are a concise way of specifying a method to be invoked in a lambda expression. They are used when a lambda expression is used to call an existing method, without modifying its behavior.**

**Method references are created using the :: operator, and can be of four types:**

**Reference to a static method: The method reference is created by specifying the class name followed by :: and the method name, for example Math::pow.**

**Reference to an instance method of a particular object: The method reference is created by specifying the object followed by :: and the method name, for example "hello"::length**

**Reference to an instance method of an arbitrary object of a particular type: The method reference is created by specifying the class name or interface name followed by :: and the method name, for example String::length**

**Reference to a constructor: The method reference is created by specifying the class name followed by :: and the keyword new, for example ArrayList::new**

**For example, the following code shows how to use a method reference to sort a list of strings in descending order using the sort method of the List interface:**

**javascript**

**Copy code**

**List<String> words = Arrays.asList("apple", "banana", "cherry");**

**words.sort(Comparator.reverseOrder());**

**We can also use a method reference in place of a lambda expression**

**less**

**Copy code**

**words.sort(Comparator.comparing(String::length).reversed());**

**In summary, method references are a new feature in Java 8 that allow you to use existing methods in lambda expressions, without modifying their behavior. They are created using the :: operator and can be of four types: reference to a static method, reference to an instance method of a particular object, reference to an instance method of an arbitrary object of a particular type, and reference to a constructor. Using method references can make your code more readable, concise, and expressive.**

**Springboot:**

**21.What is spring boot**

**Spring Boot is a framework for building standalone, production-grade Java applications using the Spring Framework. It is designed to simplify the process of creating and deploying Spring-based applications by providing a set of pre-configured, opinionated defaults and a set of tools for working with Spring, including an embedded web server, a command-line interface, and a set of starter dependencies for common use cases.**

**Spring Boot provides a number of features that make it easier to develop and deploy Spring-based applications, such as:**

**Auto-configuration: Spring Boot automatically configures your application based on the dependencies it finds on the classpath, making it easy to get started with minimal configuration.**

**Embedded web server: Spring Boot includes an embedded web server (such as Tomcat, Jetty, or Undertow), so you don't need to worry about deploying your application to a separate web server.**

**Command-line interface: Spring Boot provides a command-line interface (CLI) that makes it easy to create and run Spring-based applications.**

**Starter dependencies: Spring Boot provides a set of starter dependencies, which are pre-configured sets of libraries that you can use to add functionality to your application, such as data access, security, and web services.**

**Actuator: Spring Boot includes Actuator, a set of tools for monitoring and managing your application in production.**

**Spring Boot is particularly useful when you need to create a simple, stand-alone Java application quickly, without spending a lot of time on configuration. It can also be used to create microservices, web applications, and command-line tools.**

**In summary, Spring Boot is a framework that is built on top of the Spring Framework and provides a set of tools and features for creating and deploying Spring-based applications. It is designed to make it easy to get started with minimal configuration, and provides an embedded web server, a command-line interface, and a set of starter dependencies for common use cases. Spring Boot is particularly useful for creating simple, stand-alone Java applications quickly, without spending a lot of time on configuration.**

**22..What are starter projects and pom.xml**

**Starter projects in Spring Boot are pre-configured sets of dependencies that you can use to add functionality to your application. They are called "starters" because they provide a starting point for building a specific type of application or adding a specific feature to an application. For example, the "spring-boot-starter-web" starter includes all of the dependencies required to build a web application, such as the Spring Web MVC framework and the embedded Tomcat web server.**

**POM stands for "Project Object Model" and is an XML file that contains information about a project, including its dependencies, build information, and more. When you create a Spring Boot application, a POM file is automatically generated for you. The POM file for a Spring Boot application typically includes information about the Spring Boot parent project, as well as the starter dependencies for the application.**

**In the POM file, you can specify the dependencies that your application needs by adding them to the <dependencies> section of the file. Spring Boot provides a number of starter dependencies that you can use to add functionality to your application, such as the "spring-boot-starter-web" dependency for building web applications. You can also specify additional dependencies that are not provided by Spring Boot.**

**The POM file also contains the build information for your application, such as the version of the Java compiler to use, the location of the source code, and the location of the output files.**

**In summary, Starter projects are pre-configured sets of dependencies that you can use to add functionality to your Spring Boot application. The POM file (Project Object Model) is an XML file that contains information about your project, including its dependencies, build information and more. It's automatically generated when you create a Spring Boot application and it specifies the dependencies that your application needs to run and the build information for your application.**

**22.What are actuators in spring and what is the default embedded server in spring**

**Actuators in Spring Boot are a set of tools that allow you to monitor and manage your application. They provide endpoints that expose information about the application's health, configuration, and runtime metrics. Some examples of the information that you can retrieve from actuator endpoints include:**

**the current state of the application (e.g., whether it's up and running, or if there are any errors)**

**the current configuration of the application (e.g., the properties that are set)**

**the current performance metrics of the application (e.g., the number of requests that have been processed, the amount of memory that's being used)**

**Actuator endpoints can be accessed through HTTP or JMX and are secured by default.**

**The default embedded server in Spring Boot is Apache Tomcat. It is a Java Servlet container that is lightweight and easy to use. Spring Boot will automatically configure and start an instance of Tomcat for you when you run your application, so you don't have to do any additional configuration. However, you can also use other embedded servers like Jetty and Undertow.**

**In summary, Actuators in Spring Boot are a set of tools that allow you to monitor and manage your application by exposing endpoints that provide information about the application's health, configuration, and runtime metrics. The default embedded server in Spring Boot is Apache Tomcat which is a Java Servlet container that is lightweight and easy to use. It's automatically configured and started by Spring Boot when you run your application, but you can also use other embedded servers like Jetty and Undertow.**

**23.Explain Dependency injection in spring**

**Dependency injection (DI) is a design pattern that allows a class to have its dependencies (i.e., other objects it needs to perform its work) provided to it, rather than creating them itself. In Spring, this is done through the use of the Spring IoC container, which is responsible for instantiating and configuring objects, and then injecting them into other objects as needed.**

**There are two ways to configure dependency injection in Spring: XML-based configuration and annotation-based configuration.**

**XML-based configuration:**

**In this approach, you define the beans (i.e., objects that will be managed by the Spring IoC container) and their dependencies in an XML file called applicationContext.xml. The Spring framework reads this file and creates the objects that are defined in it.**

**Annotation-based configuration:**

**In this approach, you use annotations to define the beans and their dependencies directly in the Java code. The most common annotations used for this purpose are @Autowired and @Component.**

**Here is an example of how to use @Autowired to inject a dependency:**

**kotlin**

**Copy code**

**class Service{**

**@Autowired**

**private Dependency dependency;**

**//...**

**}**

**In summary, Dependency Injection (DI) is a design pattern that allows a class to have its dependencies provided to it, rather than creating them itself. Spring Framework uses the Inversion of Control (IoC) container to accomplish this. There are two ways to configure Dependency Injection in Spring, XML-based configuration and annotation-based configuration. Annotation-based configuration is more popular because it is less verbose and more readable.**

**24.Types of dependency injection**

**There are two main types of dependency injection in Spring: constructor injection and setter injection.**

**Constructor Injection:**

**This type of injection is done by passing the required dependencies as arguments to the class's constructor. The Spring framework will then use reflection to instantiate the class and pass in the required dependencies.**

**Here is an example of constructor injection:**

**kotlin**

**Copy code**

**class Service{**

**private Dependency dependency;**

**public Service(Dependency dependency){**

**this.dependency = dependency;**

**}**

**//...**

**}**

**Setter Injection:**

**This type of injection is done by calling setter methods on the class after it has been instantiated. The Spring framework will instantiate the class and then call the setter methods to set the required dependencies.**

**Here is an example of setter injection:**

**typescript**

**Copy code**

**class Service{**

**private Dependency dependency;**

**@Autowired**

**public void setDependency(Dependency dependency){**

**this.dependency = dependency;**

**}**

**//...**

**}**

**Both types of injection have the same end goal, which is to provide the class with its required dependencies. The choice between the two often comes down to personal preference or the specific requirements of the project.**

**In summary, There are two types of dependency injection in Spring: Constructor injection and Setter injection. Constructor injection is done by passing the required dependencies as arguments to the class's constructor, while setter injection is done by calling setter methods on the class after it has been instantiated. Both types of injection have the same end goal, which is to provide the class with its required dependencies.**

**25.What is auto wiring and its types**

**Auto-wiring is a feature in Spring Framework that allows the framework to automatically wire dependencies between beans. This means that the framework will automatically create and set the dependencies for a bean without the developer having to explicitly define the relationships.**

**There are five types of auto-wiring in Spring:**

**no: This is the default setting, it means that the developer has to explicitly define the relationships between beans.**

**byName: This type of auto-wiring looks for a bean with the same name as the property in the class and wires it.**

**byType: This type of auto-wiring looks for a bean of the same type as the property in the class and wires it.**

**constructor: This type of auto-wiring looks for a constructor argument of the same type as the property in the class and wires it.**

**autodetect: This type of auto-wiring first tries to use constructor injection and if it is not possible, it falls back to byType injection.**

**Here is an example of auto-wiring byType:**

**java**

**Copy code**

**@Autowired**

**private Dependency dependency;**

**It's important to note that auto-wiring is not always the best option and can lead to issues such as ambiguity if more than one bean of the same type is present in the application context. In such cases, explicit wiring should be used instead.**

**In summary, Auto-wiring is a feature of Spring Framework which allows the framework to automatically wire dependencies between beans. There are five types of auto-wiring: no, byName, byType, constructor and autodetect. Auto-wiring can lead to issues such as ambiguity if more than one bean of the same type is present in the application context. In such cases, explicit wiring should be used.**

**26.Explain these spring**

**Spring Framework: Spring is an open-source framework for building Java applications. It provides a comprehensive set of features for developing enterprise applications, including support for dependency injection, data access, transaction management, and more.**

**Spring Boot: Spring Boot is a framework built on top of the Spring Framework that makes it easy to create stand-alone, production-grade Spring-based applications. It provides an opinionated approach to configuration and eliminates much of the boilerplate code typically required for a Spring application.**

**Spring MVC: Spring MVC is a module of the Spring Framework that provides a Model-View-Controller (MVC) architecture for building web applications. It enables developers to build web applications using a familiar, annotation-based programming model.**

**Spring Data: Spring Data is a set of libraries and modules that make it easy to access and manipulate data using Spring. It provides a consistent, high-level programming model for working with different data access technologies, such as JDBC, JPA, and MongoDB.**

**Spring Security: Spring Security is a module of the Spring Framework that provides authentication and authorization for web applications. It enables developers to secure their applications by declaratively specifying security rules and providing a flexible, pluggable architecture for authentication and authorization.**

**Spring AOP: Spring AOP is a module of the Spring Framework that provides aspect-oriented programming (AOP) capabilities. It enables developers to modularize cross-cutting concerns, such as security and logging, and apply them declaratively to their application.**

**In summary, Spring Framework is an open-source framework for building Java applications, Spring Boot is a framework built on top of the Spring Framework that makes it easy to create stand-alone, production-grade Spring-based applications. Spring MVC is a module of the Spring Framework that provides a Model-View-Controller (MVC) architecture for building web applications. Spring Data is a set of libraries and modules that make it easy to access and manipulate data using Spring. Spring Security is a module of the Spring Framework that provides authentication and authorization for web applications and Spring AOP is a module of the Spring Framework that provides aspect-oriented programming (AOP) capabilities.**

**26.Explain profiling in springboot**

**Profiling in Spring Boot refers to the process of measuring the performance of a Spring Boot application in order to identify and troubleshoot any potential performance bottlenecks. Profiling can be used to gather data on various aspects of an application's performance, such as CPU usage, memory usage, database queries, and more.**

**Spring Boot provides several built-in tools for profiling, including the Spring Boot Actuator, which provides a set of endpoints for monitoring and managing a Spring Boot application. The Actuator provides several endpoints for gathering performance data, such as /metrics, /health, and /trace.**

**In addition to the Actuator, Spring Boot also provides support for external profiling tools, such as JProfiler, VisualVM, and Java Mission Control. These tools can be used to gather detailed performance data on a Spring Boot application, such as CPU and memory usage, database queries, and more.**

**To use profiling in Spring Boot, you will need to configure your application to enable the Actuator endpoints or connect to external profiling tools. You can enable the Actuator by adding the spring-boot-starter-actuator dependency to your project, and then configure the endpoints you want to use in your application's properties file.**

**It's important to note that profiling should only be used in a development or staging environment and not in production environment as it may affect the performance of the system.**

**26.Explain scope of spring bean**

**In Spring Framework, a bean is an object that is managed by the Spring IoC container. The scope of a bean defines the lifecycle and visibility of the bean within the container. Spring Framework provides several different scopes for beans, including:**

**Singleton: Only one instance of the bean is created by the container and is shared by all clients. This is the default scope for a bean.**

**Prototype: A new instance of the bean is created for each client.**

**Request: A new instance of the bean is created for each HTTP request. This scope is only available in web-aware Spring ApplicationContexts.**

**Session: A new instance of the bean is created for each HTTP session. This scope is only available in web-aware Spring ApplicationContexts.**

**Global session: A new instance of the bean is created for a global HTTP session. This scope is only available in web-aware Spring ApplicationContexts.**

**Application: A new instance of the bean is created for the entire application. This scope is only available in web-aware Spring ApplicationContexts.**

**WebSocket: A new instance of the bean is created for each WebSocket session. This scope is only available in web-aware Spring ApplicationContexts.**

**You can specify the scope of a bean in the configuration file or by using the @Scope annotation on the class or by using scope attribute in @Bean annotation.**

**It's important to understand that the scope of a bean affects how the container creates, manages, and destroys the bean and also affects the availability and visibility of the bean to other beans in the container. Choosing the appropriate scope for a bean can have a significant impact on the performance, scalability, and maintainability of an application.**

**27.What is properties file in spring and what details will it have**

**In Spring, a properties file is a file that contains key-value pairs used to configure various aspects of the application, such as database connections, email settings, and more. These properties can be accessed and used throughout the application using the Spring's Environment abstraction.**

**A typical properties file in Spring would have the following details:**

**Database connection details: This includes properties like the JDBC URL, username, password, and other connection-related properties.**

**Email server settings: This includes properties like the SMTP server host, port, username, and password used for sending emails.**

**Application-specific properties: This includes properties like the application name, version, and other custom properties that are specific to the application.**

**Logging properties: This includes properties for configuring the logging framework, such as the log level, appenders, and log file location.**

**Security properties: This includes properties related to security, such as encryption keys, passwords, and other sensitive information.**

**External Services: This includes properties related to External services such as AWS S3, google cloud storage, external APIs etc**

**The properties file can be in any format such as .properties or .yml, and can be located in the classpath or external location. Spring can automatically detect and load properties from the properties file and make them available to the application through the Environment abstraction.**

**It's important to note that it's good practice to use properties files in spring to keep the sensitive information like passwords, keys or any other sensitive information out of the source code and in a separate location which can be easily managed and updated.**

**Spring boot profiles**

**Spring Boot Profiles are a way to manage different configurations for different environments in your application. For example, you may have different configurations for development, testing, and production environments. With Spring Boot Profiles, you can create separate configuration files for each environment and activate the appropriate profile based on the environment in which the application is running.**

**To use profiles in Spring Boot, you can create a application.properties or application.yml file in the src/main/resources directory of your project. Within that file, you can define properties specific to a certain profile by prefixing the property with the profile name, like spring.datasource.username=admin for dev profile and spring.datasource.username=prod for prod profile.**

**You can also create separate properties files for each profile, such as application-dev.properties and application-prod.properties, and place them in the src/main/resources directory.**

**To activate a specific profile, you can use the spring.profiles.active property in the application.properties file or pass in the --spring.profiles.active=dev as JVM argument when starting the application.**

**You can also use the @Profile annotation to specify that a bean should only be created when a specific profile is active. This can be useful for creating environment-specific beans, such as different data sources for development and production environments.**

**Overall, Spring Boot Profiles provide an easy way to manage different configurations for different environments in your application, and make it easy to switch between configurations as needed.**

**27. What IDE are you using to develop java-Answer Eclipse**

**annotations @springbootapplication,@autowire,@Controller,@Restcontroller@Service,@repository,@requestmapping@qualifier**

**@SpringBootApplication: This is a convenience annotation that is used to configure and enable various features of a Spring Boot application, such as auto-configuration, component scanning, and more.**

**@Autowired: This annotation is used to inject a bean from the Spring application context into a class field, setter method, or constructor. It is used for dependency injection.**

**@Controller: This annotation is used to mark a class as a Spring MVC controller. A controller class is responsible for handling incoming HTTP requests and returning the appropriate response.**

**@RestController: This is a specialized version of the @Controller annotation that is used to create RESTful web services. It is a combination of @Controller and @ResponseBody annotations.**

**@Service: This annotation is used to mark a class as a Spring service bean. A service class typically contains business logic and is used to perform actions on the data model.**

**@Repository: This annotation is used to mark a class as a Spring data repository. A repository class is typically used to interact with a database and perform CRUD operations on the data model.**

**@RequestMapping: This annotation is used to map an HTTP request to a specific method in a controller class. It can be used at the class level or the method level.**

**@Qualifier: This annotation is used to specify which implementation of a particular interface should be used when multiple implementations are present in the application context. It is used in conjunction with the @Autowired annotation.**

**In summary, these annotations are used to configure, enable and control the behavior of different components in a Spring Boot application such as controllers, services, repositories, and more. They make it easy to manage the dependencies between different components and to structure the application in a consistent and maintainable way.**

**Rest API:**

**27.Expalin rest Api and its uses**

**A REST API (Representational State Transfer API) is a type of web service that uses the HTTP protocol to allow clients (typically web or mobile applications) to interact with a server. RESTful APIs use standard HTTP methods (such as GET, POST, PUT, and DELETE) to perform operations on resources, which are identified by URIs.**

**A resource in the context of a RESTful API is a piece of data or functionality that can be accessed or manipulated by the API. Examples of resources might include a specific user, a collection of users, or a specific item in an e-commerce application.**

**The main use of REST APIs is to provide a way for different applications to communicate with each other over the internet. For example, a mobile app might use a REST API to communicate with a server in order to retrieve data about users, or to send data about a new user registration. A web application might use a REST API to communicate with a server in order to retrieve data about products, or to send data about a new purchase.**

**Additionally, REST APIs can be used to expose the functionality of an existing system to other developers or organizations. For example, a company might create a REST API that allows external developers to access data from the company's database, or to send data to the company's system.**

**In summary, REST APIs allows different systems to communicate with each other, retrieve or send data, and perform actions on the resources, using standard HTTP methods.**

**28.What are some of the rest api error messages**

**Some common error messages that might be returned by a REST API include:**

**400 Bad Request: This error message is typically returned when the client has provided invalid or malformed data in the request.**

**401 Unauthorized: This error message is returned when the client is not properly authenticated to access the requested resource.**

**403 Forbidden: This error message is returned when the client is authenticated but does not have the necessary permissions to access the requested resource.**

**404 Not Found: This error message is returned when the client is trying to access a resource that does not exist.**

**405 Method Not Allowed: This error message is returned when the client is trying to use an HTTP method that is not supported for the requested resource.**

**406 Not Acceptable: This error message is returned when the client has specified an unacceptable content type in the request headers.**

**408 Request Timeout: This error message is returned when the server is taking too long to respond to the client's request.**

**500 Internal Server Error: This error message is returned when the server encounters an unexpected error while processing the client's request.**

**502 Bad Gateway: This error message is returned when the server is acting as a proxy and the upstream server returned an error.**

**503 Service Unavailable: This error message is returned when the server is temporarily unable to handle the client's request.**

**Keep in mind that these are general examples and actual error messages may vary depending on the specific API and its implementation.**

**29.Error codes in rest api**

**In a REST API, error codes are typically represented by HTTP status codes. These codes indicate the outcome of the request and provide additional information about the error. Some common HTTP status codes used in REST APIs include:**

**200 OK: The request was successful and the response contains the requested data.**

**201 Created: The request was successful and a new resource was created as a result.**

**204 No Content: The request was successful, but there is no data to return in the response.**

**400 Bad Request: The request was invalid or malformed. The client should check the request and try again.**

**401 Unauthorized: The request requires authentication and the client did not provide valid credentials.**

**403 Forbidden: The client does not have the necessary permissions to access the requested resource.**

**404 Not Found: The requested resource could not be found.**

**405 Method Not Allowed: The requested HTTP method is not supported for the requested resource.**

**500 Internal Server Error: An unexpected error occurred on the server.**

**503 Service Unavailable: The server is temporarily unable to handle the request.**

**Keep in mind that this is not a comprehensive list of HTTP status codes, but rather a list of some of the most commonly used status codes in REST APIs. Additionally, APIs may define their own custom error codes, which can be included in the response body along with the HTTP status code.**

**30.Get,Put,Post,Delete and Patch methods in Rest api(Important)**

**31-What is Junit(Get a brief idea)**

**JUnit is a popular open-source unit testing framework for Java. It provides annotations and assertions to help developers write and run repeatable tests for their Java code. JUnit tests are typically written as plain Java methods, with annotations and assertions used to define the test's behavior and expected outcomes.**

**JUnit is compatible with most Java development tools and can be easily integrated into a wide variety of development workflows. It provides a set of annotations and assertions to help developers write and run repeatable tests for their Java code. It allows test-driven development, which means that the developer writes the test first, before the actual implementation of the code.**

**JUnit tests can be run from the command line or integrated into a build tool such as Maven or Gradle, and it also has a plugin for development environments like Eclipse, IntelliJ IDEA, and NetBeans. JUnit tests are easy to run and debug, and it provides a consistent and clear way of verifying that the code is working as expected.**

**JUnit provides a lot of functionalities like test execution, test reporting, test debugging and test management. It's widely used for unit testing in Java projects, and it's easy to learn and use.**

**31-What is GIT version control tool,What are the commands you are aware(Learn fully about GIT,its important for storing the source code)**

**Git is a distributed version control system (VCS) that allows multiple developers to collaborate on a project by keeping track of all changes made to the source code over time. Git allows developers to create a local copy of the entire project, called a repository, which can be used to track changes, revert to previous versions, and merge changes made by different developers.**

**Some of the most commonly used Git commands include:**

**git init: Initializes a new Git repository.**

**git clone: Creates a local copy of a remote repository.**

**git add: Adds changes to the staging area, in preparation for a commit.**

**git commit: Saves changes to the repository, along with a message describing the changes.**

**git status: Shows the current status of the repository, including which files have been modified and whether they have been added to the staging area.**

**git log: Shows a history of all commits made to the repository.**

**git diff: Shows the differences between the current version of a file and the last committed version.**

**git branch: Shows a list of all branches in the repository, and highlights the current branch.**

**git checkout: Allows developers to switch between branches or switch to a specific commit.**

**git merge: Merges changes from one branch into another.**

**git pull: Fetches and merges changes from a remote repository.**

**git push: Pushes changes to a remote repository.**

**Git is widely used by developers and organizations to manage their source code and collaborate on projects. It's particularly useful when working on large projects with multiple contributors, as it allows developers to easily see who made changes and when, and to revert to previous versions if necessary.**

**In addition, Git is also useful for Open-source projects, as it allows developers to contribute to the project easily. Git also allows for easy integration with other tools, such as continuous integration servers, code review tools, and issue tracking systems.**

**It's also available on all major operating systems, and it's free and open-source software.**

**It is important to learn Git as it is a widely used tool in software development. It allows developers to collaborate, track changes, and maintain a history of their work. It also makes it easy to revert to previous versions of the code, and to collaborate with other developers on the same project.**

**Java Collections:**

**Java Collections is a framework that provides various classes and interfaces for storing and manipulating a group of objects. It contains several classes and interfaces for different types of collections, such as lists, sets, maps, etc.**

**Some of the most commonly used classes and interfaces in the Java Collections framework include:**

**ArrayList: An implementation of the List interface that uses an array to store elements.**

**LinkedList: An implementation of the List interface that uses a linked list to store elements.**

**HashSet: An implementation of the Set interface that uses a hash table to store elements.**

**TreeSet: An implementation of the SortedSet interface that uses a tree to store elements.**

**HashMap: An implementation of the Map interface that uses a hash table to store key-value pairs.**

**TreeMap: An implementation of the SortedMap interface that uses a tree to store key-value pairs.**

**The Java Collections framework also includes several interfaces that define the basic operations that collections should provide, such as the Collection interface, which defines operations like add, remove, and contains, and the Map interface, which defines operations like put and get.**

**The framework also provides several utility classes for working with collections, such as the Collections class, which provides methods for sorting, reversing, and filling collections.**

**Overall, the Java Collections framework is a powerful and flexible tool for working with groups of objects in Java, and is an essential part of many Java programs.**

**32.What is Hierarchy of Java collection**

**The hierarchy of the Java collection framework starts with the Collection interface, which is the root of the collection hierarchy. The Collection interface declares the basic operations that all collections should have, such as adding and removing elements, and checking the number of elements in a collection.**

**The Collection interface is then extended by the List and Set interfaces, which provide additional operations specific to those types of collections. Lists are ordered collections of elements, while Sets are collections of unique elements.**

**The Map interface is also a part of the Java collection hierarchy, but it is not a subinterface of Collection. Maps store key-value pairs, where each key is unique.**

**The Java collection hierarchy also includes several concrete classes, such as ArrayList, LinkedList, HashSet, and HashMap, which implement the interfaces and provide a specific implementation of the operations. Additionally, the framework also provides several utility classes, such as the Collections class, which provides various utility methods for collections.**

**33.Difference between array and arraylist**

**An array and an ArrayList are both used to store collections of data, but there are some important differences between the two:**

**Size: An array has a fixed size, while an ArrayList can grow and shrink dynamically.**

**Type: An array can store items of a specific data type, such as int or String, while an ArrayList can store items of any data type.**

**Syntax: An array is created using the syntax "type[] arrayName = new type[size]", while an ArrayList is created using the syntax "ArrayList<type> listName = new ArrayList<>()".**

**Element Access: To access an element in an array, you use the array name followed by the index in square brackets. In an ArrayList, you use the get() method.**

**Memory: An array is allocated in contiguous memory locations and is stored in stack memory where as arraylist is dynamic and is stored in heap memory.**

**Performance: An array is faster than an ArrayList for certain operations, such as accessing a specific element, because the memory is contiguous. However, an ArrayList can be faster for certain operations, such as adding or removing elements from the middle of the list, because the elements do not need to be shifted.**

**In summary, arrays are useful when you need a fixed-size collection of items of a specific data type, while ArrayList is useful when you need a dynamic collection of items of any data type.**

**34.Difference between Arraylist and linked list**

**An ArrayList is a dynamic array implementation of the List interface. It is a part of the Java collection framework and is backed by an array. It is implemented as a resizable array and is relatively fast when it comes to element insertion and retrieval. However, when it comes to element deletion or insertion at a specific index, it is relatively slow as it requires shifting all the elements after the insertion or deletion point.**

**A LinkedList, on the other hand, is a doubly-linked list implementation of the List interface. It is also a part of the Java collection framework. It is relatively slow when it comes to element retrieval but is relatively fast when it comes to element insertion or deletion, especially at a specific index as it doesn't require shifting elements like ArrayList.**

**In summary, ArrayList is better for situations where you need to access elements frequently, whereas LinkedList is better for situations where you need to frequently add or remove elements.**

**35.Difference between set and list**

**A Set is a collection that cannot contain duplicate elements, while a List is a collection that can contain duplicate elements.**

**A Set is implemented using a hash table, which provides fast lookups and element tests. Sets are typically used when you need to keep track of a collection of unique elements, such as a list of unique words in a document.**

**A List, on the other hand, is implemented using an array or a linked list, and it provides fast element access but slower element tests. Lists are typically used when you need to keep track of a collection of elements in a specific order, such as a list of customer orders or a list of items in a shopping cart.**

**In summary, Sets are used when you need to keep track of unique elements and do not care about the order of the elements, while Lists are used when you need to keep track of elements in a specific order.**

**35.Difference between Hashmap and hashtable**

**HashMap and Hashtable are both classes in Java that implement the Map interface, but they have some key differences:**

**HashMap is not synchronized, while Hashtable is. This means that HashMap is not thread-safe and can be faster than Hashtable in a single-threaded environment. Hashtable can be used in a multi-threaded environment without the need for explicit synchronization, but at the cost of reduced performance.**

**HashMap allows null keys and values, while Hashtable does not. In HashMap, you can have one null key and multiple null values, but in Hashtable, you cannot have any null key or value.**

**HashMap is a part of the Java Collection Framework since Java 2, and is found in the java.util package. Hashtable is also a part of the Collection Framework since Java 1.0, and is found in the java.util package.**

**In summary, HashMap is more flexible and efficient than Hashtable, but Hashtable is more thread-safe. If you don't need the thread-safety and can handle the null values, then HashMap should be the preferred choice. In most cases, HashMap is the better choice for performance and flexibility.**

**36.Explain internal working of Hashmap**

**A HashMap is a data structure that stores key-value pairs. It uses a hash function to compute an index into an array of buckets, from which the desired value can be found.**

**When a new key-value pair is added to a HashMap, the hash function is used to compute the index of the bucket where the pair should be stored. If the bucket is empty, the pair is simply added to it. If the bucket is not empty, the pair is added to a linked list of key-value pairs that are stored in that bucket.**

**When a value is retrieved from a HashMap, the hash function is used to compute the index of the bucket where the desired key-value pair is stored. The linked list in that bucket is then searched for the key, and if it is found, its associated value is returned.**

**The internal working of HashMap uses an array of Entry objects, where each Entry represents a key-value pair. When a new key-value pair is added to a HashMap, the key's hashcode is used to calculate the index of the array element where the pair should be added. If the element is empty, the pair is added to that element.**

**If the element is already occupied, the pair is added to a linked list of Entry objects that are stored at that element. This linked list is called as a bucket. When a value is retrieved from a HashMap, the key's hashcode is used to calculate the index of the array element where the pair should be. The linked list in that element is then searched for the key, and if it is found, its associated value is returned.**

**It allows one null key and multiple null values. It is non-synchronized means multiple threads can access it at the same time.**

**37.Explain internal working of Hashset**

**A HashSet is a collection that stores unique elements, and it is implemented using a HashMap. The internal working of a HashSet is based on the internal working of a HashMap.**

**When an element is added to a HashSet, it is first passed through a hashCode() method to generate a unique key. This key is then passed through the equals() method to check if the element already exists in the set. If the element does not exist, it is added to the HashMap as a key, with a dummy value (usually the constant PRESENT) as its value.**

**When an element is searched for in the HashSet, the hashCode() method is used to generate the key and the equals() method is used to check if the element is present in the HashMap. If the element is present, it is considered to be in the HashSet.**

**When an element is removed from the HashSet, it is removed from the HashMap as well, using the key generated by the hashCode() method.**

**In summary, HashSet uses a HashMap to store unique elements with a constant value as the value of the key, and uses the hashCode() and equals() methods to determine if an element already exists in the set and to locate elements in the set.**

**Hibernate and uses**

**Hibernate is an open-source object-relational mapping (ORM) tool for the Java programming language. It provides a framework for mapping an object-oriented domain model to a relational database. Hibernate automatically generates the SQL statements required to interact with the database, and it also provides a way to express queries using a high-level, object-oriented language. This allows developers to work with the data in a more natural and intuitive way, without having to write complex SQL statements.**

**Hibernate is commonly used in the development of Java-based enterprise applications, particularly those that use a relational database. It is also widely used in the development of web applications, as it allows for easy integration with popular web frameworks such as Spring and Struts. Hibernate also provides support for caching and lazy loading of data, which can improve the performance of an application.**