Explore More

Subcription : Premium CDAC NOTES & MATERIAL @99

Contact to Join
Premium Group

Click to Join
Telegram Group

For More E-Notes

Join Our Community to stay Updated

TAP ON THE ICONS TO JOIN!

	codewitharrays.in freelance project available to buy contact on 8007592194	
SR.NO	Project NAME	Technology
1	Online E-Learning Platform Hub	React+Springboot+MySql
2	PG Mates / RoomSharing / Flat Mates	React+Springboot+MySql
3	Tour and Travel management System	React+Springboot+MySql
4	Election commition of India (online Voting System)	React+Springboot+MySql
5	HomeRental Booking System	React+Springboot+MySql
6	Event Management System	React+Springboot+MySql
7	Hotel Management System	React+Springboot+MySql
8	Agriculture web Project	React+Springboot+MySql
9	AirLine Reservation System / Flight booking System	React+Springboot+MySql
10	E-commerce web Project	React+Springboot+MySql
11	Hospital Management System	React+Springboot+MySql
12	E-RTO Driving licence portal	React+Springboot+MySql
13	Transpotation Services portal	React+Springboot+MySql
14	Courier Services Portal / Courier Management System	React+Springboot+MySql
15	Online Food Delivery Portal	React+Springboot+MySql
16	Muncipal Corporation Management	React+Springboot+MySql
17	Gym Management System	React+Springboot+MySql
18	Bike/Car ental System Portal	React+Springboot+MySql
19	CharityDonation web project	React+Springboot+MySql
20	Movie Booking System	React+Springboot+MySql

	freelance_Project available to buy contact on 8007592194	
21	Job Portal web project	React+Springboot+MySql
22	LIC Insurance Portal	React+Springboot+MySql
23	Employee Management System	React+Springboot+MySql
24	Payroll Management System	React+Springboot+MySql
25	RealEstate Property Project	React+Springboot+MySql
26	Marriage Hall Booking Project	React+Springboot+MySql
27	Online Student Management portal	React+Springboot+MySql
28	Resturant management System	React+Springboot+MySql
29	Solar Management Project	React+Springboot+MySql
30	OneStepService LinkLabourContractor	React+Springboot+MySql
31	Vehical Service Center Portal	React+Springboot+MySql
32	E-wallet Banking Project	React+Springboot+MySql
33	Blogg Application Project	React+Springboot+MySql
34	Car Parking booking Project	React+Springboot+MySql
35	OLA Cab Booking Portal	React+NextJs+Springboot+MySql
36	Society management Portal	React+Springboot+MySql
37	E-College Portal	React+Springboot+MySql
38	FoodWaste Management Donate System	React+Springboot+MySql
39	Sports Ground Booking	React+Springboot+MySql
40	BloodBank mangement System	React+Springboot+MySql

41	Bus Tickit Booking Project	React+Springboot+MySql
42	Fruite Delivery Project	React+Springboot+MySql
43	Woodworks Bed Shop	React+Springboot+MySql
44	Online Dairy Product sell Project	React+Springboot+MySql
45	Online E-Pharma medicine sell Project	React+Springboot+MySql
46	FarmerMarketplace Web Project	React+Springboot+MySql
47	Online Cloth Store Project	React+Springboot+MySql
48	Train Ticket Booking Project	React+Springboot+MySql
49	Quizz Application Project	JSP+Springboot+MySql
50	Hotel Room Booking Project	React+Springboot+MySql
51	Online Crime Reporting Portal Project	React+Springboot+MySql
52	Online Child Adoption Portal Project	React+Springboot+MySql
53	online Pizza Delivery System Project	React+Springboot+MySql
54	Online Social Complaint Portal Project	React+Springboot+MySql
55	Electric Vehical management system Project	React+Springboot+MySql
56	Online mess / Tiffin management System Project	React+Springboot+MySql
57		React+Springboot+MySql
58		React+Springboot+MySql
59		React+Springboot+MySql
60		React+Springboot+MySql

Spring Boot + React JS + MySQL Project List

Sr.No	Project Name	YouTube Link
1	Online E-Learning Hub Platform Project	https://youtu.be/KMjyBaWmgzg?si=YckHuNzs7eC84-IW
2	PG Mate / Room sharing/Flat sharing	https://youtu.be/4P9clHg3wvk?si=4uEsi0962CG6Xodp
3	Tour and Travel System Project Version 1.0	https://youtu.be/-UHOBywHaP8?si=KHHfE_A0uv725f12
4	Marriage Hall Booking	https://youtu.be/VXz0kZQi5to?si=llOS-QG3TpAFP5k7
5	Ecommerce Shopping project	https://youtu.be/vJ_C6LkhrZ0?si=YhcBylSErvdn7paq
6	Bike Rental System Project	https://youtu.be/FIzsAmIBCbk?si=7ujQTJqEgkQ8ju2H
7	Multi-Restaurant management system	https://youtu.be/pvV-pM2Jf3s?si=PgvnT-yFc8ktrDxB
8	Hospital management system Project	https://youtu.be/IynIouBZvY4?si=CXzQs3BsRkjKhZCw
9	Municipal Corporation system Project	https://youtu.be/cVMx9NVyI4I?si=qX0oQt-GT-LR_5jF
10	Tour and Travel System Project version 2.0	https://youtu.be/_4u0mB9mHXE?si=gDiAhKBowi2gNUKZ

Sr.No	Project Name	YouTube Link
11	Tour and Travel System Project version 3.0	https://youtu.be/Dm7nOdpasWg?si=P_Lh2gcOFhlyudug
12	Gym Management system Project	https://youtu.be/J8_7Zrkg7ag?si=LcxV51ynfUB7OptX
13	Online Driving License system Project	https://youtu.be/3yRzsMs8TLE?si=JRI_z4FDx4Gmt7fn
14	Online Flight Booking system Project	https://youtu.be/m755rOwdk8U?si=HURvAY2VnizIyJlh
15	Employee management system project	https://youtu.be/lD1iE3W_GRw?si=Y_jv1xV_BljhrD0H
16	Online student school or college portal	https://youtu.be/4A25aEKfei0?si=RoVgZtxMk9TPdQvD
17	Online movie booking system project	https://youtu.be/Lfjv_U74SC4?si=fiDvrhhrjb4KSlSm
18	Online Pizza Delivery system project	https://youtu.be/Tp3izreZ458?si=8eWAOzA8SVdNwlyM
19	Online Crime Reporting system Project	https://youtu.be/0UlzReSk9tQ?si=6vN0e70TVY1GOwPO
20	Online Children Adoption Project	https://youtu.be/3T5HC2HKyT4?si=bntP78niYH802I7N

Q - 1) What is the default value of an instance variable in Java?

The default value of an instance variable depends on its data type.For numeric types (`byte`, `short`, `int`, `long`, `float`, `double`), the default value is `0`.For `boolean`, it is `false`, and for object references, it is `null`.

Q - 2) How do you declare a variable in Java?

To declare a variable in Java, you specify the data type followed by the variable name.For example, `int age;` declares a variable named `age` of type `int`.

Q - 3) What is the purpose of the `main` method in Java?

The `main` method serves as the entry point for a Java application. It is where the program starts executing. Its signature is `public static void main(String[] args)`.

Q - 4) What is a constructor in Java?

A constructor is a special method used to initialize objects. It has the same name as the class and does not have a return type. Constructors are called when an object is created.

Q - 5) What is the difference between == and `.equals()` in Java?

== compares the references of two objects to see if they point to the same memory location. `.equals()` compares the actual content or values of the objects to determine equality.

Q - 6) What is an `interface` in Java?

An `interface` in Java is a reference type that can contain only constants, method signatures, default methods, static methods, and nested types. Interfaces are used to achieve abstraction and multiple inheritance. A class implements an interface to provide the behavior defined by the interface.

Q - 7) How do you create a new object in Java?

To create a new object in Java, use the `new` keyword followed by the class constructor. For example, `MyClass obj = new MyClass();` creates a new instance of `MyClass`.

Q - 8) What is a package in Java?

A package in Java is a namespace that organizes a set of related classes and interfaces. It helps in grouping related types and managing their access, avoiding naming conflicts, and providing a modular structure.

Q - 9) How do you handle exceptions in Java?

Exceptions in Java are handled using `try`, `catch`, `finally`, and `throw` blocks. You place the code that may throw an exception inside a `try` block and handle the exception in the

`catch` block. The `finally` block contains code that will execute regardless of whether an exception was thrown or not.

Q - 10) What is the purpose of the `static` keyword in Java?

The `static` keyword defines class-level methods and variables that can be accessed without creating an instance of the class. It belongs to the class rather than instances of the class.

Q - 11) What is an abstract class in Java?

An abstract class in Java is a class that cannot be instantiated on its own and may contain abstract methods (methods without implementation) that must be implemented by subclasses. It is used to provide a common base for subclasses to extend and provide specific implementations.

Q - 12) What is the difference between `int` and `Integer` in Java?

`int` is a primitive data type, while `Integer` is a wrapper class that encapsulates an `int` value in an object. `Integer` provides methods to convert between `int` and other types and offers useful utility functions.

Q - 13) What is a `for-each` loop in Java?

The `for-each` loop is used to iterate over elements in an array or a collection. It simplifies the loop syntax and avoids the need for an index variable. For example: `for (String item : collection) { // process item }`.

Q - 14) What is the use of the `final` keyword in Java?

The `final` keyword is used to declare constants, prevent method overriding, and inheritance. When applied to a variable, it makes the value immutable; when applied to a method, it prevents overriding; and when applied to a class, it prevents inheritance.

Q - 15) How do you implement inheritance in Java?

Inheritance in Java is implemented using the `extends` keyword. A subclass inherits from a superclass using this keyword, allowing it to access the superclass's methods and fields. For example: `class SubClass extends SuperClass`.

Q - 16) What is method overloading in Java?

Method overloading occurs when multiple methods in a class have the same name but different parameter lists (different type or number of parameters). Overloaded methods must differ in their parameter lists.

Q - 17) What is method overriding in Java?

Method overriding occurs when a subclass provides a specific implementation of a method that is already defined in its superclass. The method in the subclass must have the same name, return type, and parameters as the method in the superclass.

Q - 18) What is the purpose of the `this` keyword in Java?

The `this` keyword refers to the current object instance. It is used to access instance variables and methods from within the class, and to distinguish between instance variables and parameters with the same name.

Q - 19) How do you define a constant in Java?

To define a constant in Java, use the `final` keyword with a variable declaration. For example: `public static final int MAX_VALUE = 100;`. The `final` keyword ensures that the value cannot be changed after initialization.

Q - 20) How does Java achieve platform independence?

Java achieves platform independence through the use of the Java Virtual Machine (JVM). Java code is compiled into bytecode, which is platform-independent. The JVM interprets or compiles this bytecode into machine code specific to the operating system and hardware, allowing Java applications to run on any platform that has a compatible JVM.

Q - 21) What are the differences between `ArrayList` and `LinkedList` in Java?

ArrayList: `ImplementsList` interface using a dynamic array. It provides fast random access (O(1) time complexity) but slow insertion and deletion (O(n) time complexity) due to array resizing.

LinkedList: `ImplementsListinterface using a doubly linked list. It provides faster insertion and deletion (O(1) time complexity) compared to`ArrayList`, but slower random access (O(n) time complexity).

Q - 22) How do you perform type casting in Java?

Type casting in Java involves converting a variable from one type to another. For primitive types, you can use explicit casting: `int i = (int) 3.14;`. For object references, use explicit casting: `Object obj = "Hello"; String str = (String) obj;`. Be cautious with object casting to avoid `ClassCastException`.

Q - 23) What is the use of the `super` keyword in Java?

The `super` keyword refers to the superclass of the current object. It is used to call superclass methods and constructors, and to access superclass fields. It helps in invoking overridden methods or constructors in the parent class.

Q - 24) What is the difference between `throw` and `throws` in Java?

`throw`: Used to explicitly throw an exception from a method or a block of code. Example: `throw new IOException("Error message");`.

`throws`: Used in a method declaration to indicate that the method may throw one or more exceptions, and the caller must handle or declare them. Example: `public void readFile() throws IOException`.

Q - 25) What is a `HashMap` in Java, and how does it work?

`HashMap` is a part of Java's Collections Framework and implements the `Map` interface. It stores key-value pairs, allowing fast retrieval based on the key. It uses a hash table to store data, where keys are hashed to determine their storage location. It allows null values and one null key.

Q - 26) How do you synchronize a method in Java?

Synchronize a method in Java using the `synchronized` keyword to ensure that only one thread can execute the method at a time. Example: `public synchronized void myMethod() { // method body }`.

Q - 27) What is the difference between `String`, `StringBuilder`, and `StringBuffer` in Java?

`String`: Immutable and thread-safe. Any modification results in a new `String` object.

`StringBuilder`: Mutable and not thread-safe. Provides better performance for single-threaded scenarios where the string is modified frequently.

`StringBuffer`: Mutable and thread-safe. Suitable for multi-threaded scenarios where the string is modified frequently.

Q - 28) How does the Java garbage collector work?

The Java garbage collector automatically manages memory by identifying and disposing of objects that are no longer reachable or needed. It uses algorithms like generational garbage collection, which divides objects into generations (young, old) and collects them based on their age and usage patterns.

Q - 29) What is the difference between `abstract` and `interface` in Java?

`abstract` class: Can have both abstract methods (without implementation) and concrete methods (with implementation). A class can extend only one abstract class due to single inheritance.

`interface`: Can only have method signatures (before Java 8) and final variables. From Java 8 onwards, it can have default and static methods. A class can implement multiple interfaces, supporting multiple inheritance.

Q - 30) How do you read and write files in Java?

Reading a file: Use `FileReader` and `BufferedReader` for text files, or `FileInputStream` for binary files.

Example:

BufferedReader reader = new BufferedReader(new FileReader("file.txt"));

String line = reader.readLine();

reader.close();

Writing a file: Use `FileWriter` and `BufferedWriter` for text files, or `FileOutputStream` for binary files.

Example:

BufferedWriter writer = new BufferedWriter(new FileWriter("file.txt"));

writer.write("Hello, World!");

writer.close();

Q - 31) What is the purpose of the `try-with-resources` statement?

The `try-with-resources` statement ensures that resources like files, sockets, or database connections are automatically closed after use, even if an exception occurs. Resources that implement the `AutoCloseable` interface can be used within this statement.

Example:

try (BufferedReader reader = new BufferedReader(new FileReader("file.txt"))) {

```
} catch (IOException e) {

}
```

Q - 32) What is a `enum` in Java?

An `enum` (short for enumeration) is a special class that represents a group of constants (unchangeable variables). Enums provide a way to define a collection of named values that are related, making code more readable and less error-prone.

Example:

```
public enum Day {

    SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY

}
```

Q - 33) What are the differences between `public`, `protected`, and `private` access modifiers?

Public : Accessible from any other class.

Protected : Accessible within the same package and by subclasses.

Private : Accessible only within the same class.

Q - 34) How do you implement a singleton pattern in Java?

The Singleton pattern ensures that a class has only one instance and provides a global point of access to that instance. A common implementation is:

```
public class Singleton {

    private static Singleton instance;

    private Singleton() { }

    public static synchronized Singleton getInstance() {

        if (instance == null) {

           instance = new Singleton();

        }

        return instance;

    }

}
```


Q - 35) How do you use the Java Collections Framework?

The Java Collections Framework provides data structures like `List`, `Set`, `Map`, and their implementations (`ArrayList`, `HashSet`, `HashMap`, etc.). Use these collections to store, manipulate, and retrieve groups of objects. Example:

```
List<String> list = new ArrayList<>();

list.add("Item1");
```

Q - 36) What is a `Comparator` and how is it used in Java?

A `Comparator` is an interface used to define custom sorting orders for objects. It has a single method, `compare(T o1, T o2)`. Example usage:

```
  Comparator<String> lengthComparator = (s1, s2) ->
Integer.compare(s1.length(), s2.length());

  Collections.sort(list, lengthComparator);
```

Q - 37) What is the difference between == and `equals()` when comparing objects?

== compares object references to check if they point to the same memory location, while `.equals()` compares the content or state of the objects. For custom objects, you should override the `equals()` method to compare their state.

Q - 38) What is the purpose of `hashCode()` method in Java?

The `hashCode()` method returns an integer hash code value for an object. It is used in hashing-based collections like `HashMap` to quickly locate objects. The `hashCode()` method should be overridden whenever `equals()` is overridden to maintain consistency.

Q - 39) How do you create and use a custom exception in Java?

To create a custom exception, extend the `Exception` class or one of its subclasses. Example:

```
  public class CustomException extends Exception {

      public CustomException(String message) {

          super(message);

      }

  }

  To use it:

  public void someMethod() throws CustomException {

      throw new CustomException("This is a custom exception
message");

  }
```

Q - 40) What are Java annotations, and how are they used?

Java annotations provide metadata about the code but do not affect the program's execution. They are used to give information to the compiler, runtime, or tools. Examples include `@Override`, `@Deprecated`, and `@Entity`. You can define custom annotations with the `@interface` keyword and use them to add metadata to classes, methods, or fields.

Q - 41) What is the Java Memory Model, and how does it affect concurrency?

The Java Memory Model (JMM) defines how threads interact through memory and ensures visibility and ordering of variables. It affects concurrency by specifying how and when changes made by one thread become visible to others, ensuring proper synchronization and preventing issues like stale or inconsistent data.

Q - 42) How does Java implement polymorphism?

Java implements polymorphism through method overriding and interfaces. Method overriding allows a subclass to provide a specific implementation of a method defined in its superclass. Interfaces allow a class to implement multiple behaviors, and polymorphism ensures that the correct method implementation is called based on the object's runtime type.

Q - 43) What is the difference between `synchronized` and `Lock` in Java concurrency?

`synchronized`: Provides implicit locking at the method or block level, automatically releasing the lock when the block exits. It is simpler to use but less flexible.

`Lock`: An explicit lock from the `java.util.concurrent.locks` package, offering more control and additional features like try-lock, timed lock, and interruptible lock. Example: `ReentrantLock`.

Q - 44) How do Java `Callable` and `Future` differ from `Runnable`?

`Callable`: Represents a task that can return a result or throw an exception. It has a `call()` method and returns a `Future` object.

`Runnable`: Represents a task that does not return a result or throw a checked exception. It has a `run()` method.

`Future`: Represents the result of an asynchronous computation and provides methods to check if the computation is complete or to retrieve the result.

Q - 45) What is the Java Reflection API, and how is it used?

The Java Reflection API allows inspection and manipulation of classes, methods, fields, and constructors at runtime. It is used for dynamically loading classes, creating instances, and accessing or modifying fields and methods. Example:

```
Class<?> clazz = Class.forName("com.example.MyClass");

Object instance = clazz.getDeclaredConstructor().newInstance();

Method method = clazz.getMethod("myMethod");

method.invoke(instance);
```

Q - 46) How does Java handle method resolution in case of method overloading?

Java handles method resolution at compile-time based on the method signature, which includes the method name and parameter types. The compiler selects the appropriate method based on the number and types of arguments provided during the method call.

Q - 47) What are `volatile` variables, and when would you use them?

`volatile` variables ensure visibility of changes across threads. When a variable is declared as `volatile`, changes made by one thread are immediately visible to other threads. Use `volatile` for variables that are shared among threads but do not require atomicity or synchronization.

Q - 48) What is the difference between `ConcurrentHashMap` and `HashMap`?

`HashMap`: Not thread-safe. It can cause data inconsistency when accessed by multiple threads concurrently.

`ConcurrentHashMap`: Thread-safe. It allows concurrent access by multiple threads with high performance and provides finer-grained locking to minimize contention.

Q - 49) How do you implement a thread-safe singleton in Java?

Use the `Bill Pugh Singleton Design` which leverages the `static inner helper class` approach:

```
public class Singleton {

    private Singleton() { }

    private static class Holder {

        private static final Singleton INSTANCE = new Singleton();

    }

    public static Singleton getInstance() {

        return Holder.INSTANCE;

    }

}
```

Q - 50) What is the `fork/join` framework in Java, and how does it work?

The `fork/join` framework, introduced in Java 7, is designed for parallel processing. It splits tasks into smaller sub-tasks (`fork`), processes them in parallel, and then combines the results (`join`). It uses a `ForkJoinPool` to manage and execute the tasks efficiently.

Q - 51) What are `SoftReference`, `WeakReference`, and `PhantomReference`?

`SoftReference`: Used for memory-sensitive caches. An object with a soft reference is collected only when memory is low.

`WeakReference`: Used for objects that can be collected more eagerly, typically used in caching where the reference should not prevent garbage collection.

`PhantomReference`: Used to determine when an object is about to be reclaimed. It is used for post-mortem cleanup, as it does not prevent garbage collection.

Q - 52) What is the difference between a `HashSet` and a `TreeSet`?

`HashSet`: Implements `Set` interface using a hash table. It provides constant-time performance for basic operations like add, remove, and contains, but does not guarantee order.

`TreeSet`: Implements `Set` interface using a red-black tree. It provides ordered elements and logarithmic-time performance for basic operations. It maintains elements in a sorted order.

Q - 53) . How does Java handle memory management and garbage collection?

Java handles memory management through automatic garbage collection, which reclaims memory used by objects that are no longer reachable. The garbage collector identifies and disposes of unused objects to free up memory. Java uses algorithms like generational garbage collection to manage different memory regions efficiently.

Q - 54) . What is the `java.util.Optional` class, and how is it used?

The `Optional` class represents a container that may or may not contain a non-null value. It is used to avoid null references and to handle optional values more gracefully. Example:

```
Optional<String> optional = Optional.of("Hello");

optional.ifPresent(value -> System.out.println(value));
```

Q - 55) What is the significance of `java.util.function` package?

The `java.util.function` package provides functional interfaces that support lambda expressions and method references. It includes commonly used interfaces like `Function`, `Predicate`, `Consumer`, and `Supplier`, enabling functional programming patterns and improving code readability.

Q - 56) How do you implement custom serialization in Java?

To implement custom serialization, you need to override the `writeObject` and `readObject` methods in your class. Example:

```
  private void writeObject(ObjectOutputStream oos) throws IOException
{

      oos.defaultWriteObject();

      // custom serialization logic

  }

  private void readObject(ObjectInputStream ois) throws
ClassNotFoundException, IOException {

      ois.defaultReadObject();

      // custom deserialization logic

  }
```

Q - 57) What are the differences between `CyclicBarrier` and `CountDownLatch`?

`CyclicBarrier`:** Allows a set of threads to wait for each other to reach a common barrier point before continuing. It can be reused after the barrier is reached.

`CountDownLatch`:** Allows threads to wait until a count reaches zero. It cannot be reset or reused once the count reaches zero.

Q - 58) What is the `Stream` API in Java, and how is it used?

The `Stream` API provides a functional approach to processing sequences of elements (e.g., collections) in a declarative way. It supports operations like filter, map, reduce, and collect. Example:

```
  List<String> names = Arrays.asList("Alice", "Bob", "Charlie");

  names.stream()

       .filter(name -> name.startsWith("A"))

       .forEach(System.out::println);
```

Q - 59) How do you implement a `ConcurrentLinkedQueue`?

`ConcurrentLinkedQueue` is a thread-safe, non-blocking queue implementation. It is part of the `java.util.concurrent` package and is used to handle concurrent access by multiple threads.

Example:

```
ConcurrentLinkedQueue<String> queue = new ConcurrentLinkedQueue<>();

queue.offer("Item1");

String item = queue.poll();
```

Q - 60) What is the `java.nio` package, and how does it differ from the traditional `java.io` package?

The `java.nio` (New I/O) package provides a more efficient way to perform I/O operations by using buffers and channels, enabling non-blocking I/O operations and better control over data processing. It differs from `java.io` by offering features like direct memory access (`ByteBuffer`), asynchronous I/O, and file I/O operations with `FileChannel`.

Q - 61) How do you use `CompletableFuture` in asynchronous programming?

`CompletableFuture` is used to write asynchronous, non-blocking code. You create a `CompletableFuture` instance, perform asynchronous computations, and then use methods like `thenApply`, `thenAccept`, and `thenCompose` to handle results or chain multiple asynchronous operations.

Example:

```
CompletableFuture.supplyAsync(() -> {

    return "Hello";

}).thenAccept(result -> {

    System.out.println(result);

});
```

Q - 62) What are `CompletableFuture`'s main methods and their uses?

`supplyAsync : Runs a task asynchronously and returns aCompletableFuture` with the result.

`thenApply : Transforms the result of the current stage using a function.

`thenAccept : Performs an action using the result of the current stage.

`thenCompose : Chains asynchronous computations.

`exceptionally : Handles exceptions that occur during the computation.

Q - 63) How do `Phaser` and `Exchanger` work in Java concurrency?

`Phaser`: A synchronization aid that allows multiple parties to wait for each other to reach a common barrier point. It supports dynamic registration and deregistration of parties.

`Exchanger`: Allows two threads to exchange data between them. Each thread provides an item, and the `Exchanger` ensures that both items are exchanged simultaneously.

Q - 64) What are the key differences between `LinkedBlockingQueue` and `ArrayBlockingQueue`?

`LinkedBlockingQueue`: Uses a linked list for storage, allowing for an unbounded queue unless a limit is explicitly set. It supports fair and non-fair access policies.

`ArrayBlockingQueue`: Uses a fixed-size array for storage. It is bounded by the size of the array and provides fair and non-fair access policies.

Q - 65) How do you handle deadlock in Java?

: To handle deadlock, follow strategies like:

Avoid Nested Locks: Minimize the locking scope and avoid acquiring multiple locks.

Lock Ordering: Always acquire locks in a consistent, predefined order.

Timeouts: Use lock timeouts to prevent indefinite waiting.

Deadlock Detection: Use tools or techniques to detect and recover from deadlocks.

Q - 66) What is the `java.util.concurrent` package, and what are its main classes?

The `java.util.concurrent` package provides classes and interfaces for concurrency and parallelism. Main classes include:

`ExecutorService`: Manages a pool of threads for task execution.

`Future`: Represents the result of an asynchronous computation.

`CountDownLatch`: Allows threads to wait for a count to reach zero.

`Semaphore`: Controls access to a resource by a fixed number of threads.

`ConcurrentHashMap`: A thread-safe map implementation.

Q - 67) How does Java handle class loading and classpath?

Java handles class loading through the `ClassLoader` class, which loads classes at runtime. The classpath is a parameter that specifies the locations (directories or JAR files) where the Java runtime looks for classes and resources. Java uses different class loaders (`BootstrapClassLoader`, `PlatformClassLoader`, `ApplicationClassLoader`) to load classes from the classpath.

Q - 68) What is the purpose of `java.lang.invoke` package?

The `java.lang.invoke` package provides APIs for dynamically invoking methods and creating method handles, which allow for low-level operations and optimizations related to method calls and dynamic language support. It includes classes like `MethodHandles` and `MethodType`.

Q - 69) How do you use Java's `ForkJoinPool` for parallel tasks?

The `ForkJoinPool` is used for parallel processing of tasks by recursively splitting tasks into smaller sub-tasks. Use `RecursiveTask` for tasks that return a result and `RecursiveAction` for tasks that do not.

Example:

```
  ForkJoinPool pool = new ForkJoinPool();

  RecursiveTask<Integer> task = new MyRecursiveTask();

  Integer result = pool.invoke(task);
```

Q - 70) What is the difference between `FutureTask` and `CompletableFuture`?

FutureTask: Represents a task that can be cancelled and that returns a result. It is usually used with `ExecutorService` for synchronous processing.

CompletableFuture: Supports more complex asynchronous programming with features like chaining, combining, and handling exceptions. It allows non-blocking computations and functional-style handling.

Q - 71) How do you create a custom `ClassLoader` in Java?

: Extend the `ClassLoader` class and override methods such as `findClass(String name)` to define how classes are loaded. Example:

```
  public class MyClassLoader extends ClassLoader {

      @Override

      protected Class<?> findClass(String name) throws
ClassNotFoundException {
```

```
        // Custom class loading logic
        return super.findClass(name);
    }
}
```

Q - 72) What is the significance of `Class.forName()` method?

`Class.forName(String className)` loads the class with the specified name and returns its `Class` object. It is commonly used to dynamically load classes, especially in contexts like JDBC where class names are provided at runtime.

Q - 73) What are `Spanning Tree Protocol` and `Design Patterns` in Java?

Spanning Tree Protocol (STP): A network protocol used to prevent loops in Ethernet networks by creating a spanning tree to ensure a loop-free topology.

Design Patterns : Standard solutions to common software design problems. Examples include Singleton, Factory, Observer, and Strategy patterns. They provide proven approaches to solving recurring design issues.

Q - 74) How do you implement and use a `PriorityQueue` in Java?

`PriorityQueue` is a queue that orders elements based on their natural ordering or a specified comparator. Example:

```
PriorityQueue<Integer> queue = new PriorityQueue<>();
queue.add(3);
queue.add(1);
queue.add(2);
while (!queue.isEmpty()) {
    System.out.println(queue.poll()); // Outputs: 1, 2, 3
}
```

Q - 75) What are the differences between `CopyOnWriteArrayList` and `ConcurrentLinkedQueue`?

CopyOnWriteArrayList: A thread-safe variant of `ArrayList` that creates a new copy of the underlying array whenever it is modified. Suitable for scenarios where reads are more frequent than writes.

ConcurrentLinkedQueue : A thread-safe, non-blocking queue that uses a linked node structure. Suitable for scenarios with frequent concurrent insertions and deletions.

Q - 76) How do `Semaphores` and `Barriers` work in Java concurrency?

Semaphore : Controls access to a resource by maintaining a set number of permits. Threads acquire and release permits to access the resource.

Barrier : (Typically `CyclicBarrier` or `Phaser`) allows multiple threads to wait until they all reach a common barrier point before proceeding.

Q - 77) What are the different types of class loaders in Java?

BootstrapClassLoader : Loads core Java classes from the JDK.

PlatformClassLoader : Loads platform classes and classes from the JRE.

ApplicationClassLoader : Loads application classes and libraries from the classpath.

Custom ClassLoader : User-defined class loaders for specific needs.

Q - 78) How do you implement a `CircularBuffer` in Java?

A `CircularBuffer` (or ring buffer) is a data structure that uses a fixed-size buffer with wrap-around behavior. Example:

```
  public class CircularBuffer<T> {

      private final Object[] buffer;

      private int head = 0;

      private int tail = 0;

      private boolean isFull = false;


      public CircularBuffer(int size) {

          buffer = new Object[size];

      }


      public synchronized void put(T item) throws InterruptedException
{

          while (isFull) {
```

```
            wait();

        }

        buffer[tail] = item;

        tail = (tail + 1) % buffer.length

}
```

Q - 79) What is the `java.util.ServiceLoader` and how does it work?

: `ServiceLoader` is a utility class in the `java.util` package that provides a simple way to load service provider implementations. It uses the Java Service Provider Interface (SPI) to discover and load service implementations specified in configuration files. Example usage:

```
  ServiceLoader<MyService> loader =
ServiceLoader.load(MyService.class);

  for (MyService service : loader) {

      service.performService();

  }
```

Q - 80) How do you use `Unsafe` class in Java for low-level operations?

The `Unsafe` class provides low-level, unsafe operations such as direct memory access and manipulation. Accessing `Unsafe` typically involves reflection and is not recommended for general use due to its potential risks. Example usage:

```
  Unsafe unsafe = Unsafe.getUnsafe();

  long address = unsafe.allocateMemory(100);

  unsafe.putInt(address, 123);

  int value = unsafe.getInt(address);

  unsafe.freeMemory(address);
```

Q - 81) What are `ThreadLocal` variables, and when should you use them?

`ThreadLocal` variables provide thread-local storage, meaning each thread accessing the variable has its own independent copy. They are used to store data that is specific to a thread and should not be shared with other threads, such as user sessions or database connections. Example:

```
  ThreadLocal<Integer> threadLocalValue = ThreadLocal.withInitial(() -
> 1);

  threadLocalValue.set(10);

  Integer value = threadLocalValue.get();
```

Q - 82) How do `Dynamic Proxies` work in Java?

Dynamic proxies in Java are used to create proxy instances that implement one or more interfaces at runtime. They are created using the `Proxy` class and a `InvocationHandler` to handle method invocations on the proxy.

Example:

```
  MyInterface proxy = (MyInterface) Proxy.newProxyInstance(

      MyInterface.class.getClassLoader(),

      new Class<?>[]{MyInterface.class},

      (proxy, method, args) -> {

          // Handle method invocation

          return null;

      });
```

Q - 83) What is `InvokeDynamic` and its role in Java?

`InvokeDynamic` is a JVM instruction introduced in Java 7 to support dynamic languages and improve the performance of dynamic method calls. It allows method calls to be dynamically linked and optimized at runtime, providing more flexibility and efficiency for dynamic language features.

Q - 84) How does Java's `SoftReference` class help with memory management?

`SoftReference` is a type of reference that allows the Java garbage collector to reclaim memory more aggressively when needed. Objects referenced by `SoftReference` are kept in memory as long as there is sufficient memory; otherwise, they are eligible for garbage collection.

Q - 85) What is `JMX`, and how can you use it to manage Java applications?

Java Management Extensions (JMX) is a framework for managing and monitoring Java applications. It provides a way to access and control resources like applications, devices, and services. You use MBeans (Managed Beans) to expose resources and JMX clients to interact with them.

Q - 86) What are the differences between `AbstractQueuedSynchronizer` and `AbstractQueuedLongSynchronizer`?

`AbstractQueuedSynchronizer`: Provides a framework for implementing blocking locks and other synchronization primitives. It uses an integer to represent the synchronization state.

AbstractQueuedLongSynchronizer :Extends AbstractQueuedSynchronizer to use a `long` state field, allowing for additional bits to be used for representing state and features.

Q - 87) How do `ReadWriteLocks` work, and when should you use them?

`ReadWriteLock` allows concurrent access to resources with separate read and write locks. Multiple threads can acquire read locks simultaneously, but write locks are exclusive. Use `ReadWriteLock` when you have frequent reads and infrequent writes, to improve concurrency.

Q - 88) What are `MethodHandles` and how do they differ from reflection?

`MethodHandles` are part of the `java.lang.invoke` package and provide a more efficient and flexible way to perform method calls compared to reflection. They are used for dynamic method invocation, with lower overhead than traditional reflection.

Q - 89) What are the benefits and drawbacks of `Java's Generics`?

Benefits:

Type Safety: Generics enable compile-time type checking, reducing runtime type errors.

Code Reusability: You can create more flexible and reusable classes and methods.

Drawbacks:

Complexity: Generics can make code more complex and harder to understand.

Type Erasure: Generics use type erasure, which can lead to loss of type information at runtime.

Q - 90) How do you implement `Custom Thread Pools` in Java?

You implement custom thread pools by extending `ThreadPoolExecutor` and configuring it with custom parameters. Example:

```
  public class CustomThreadPool extends ThreadPoolExecutor {

      public CustomThreadPool() {

          super(10, 20, 60L, TimeUnit.SECONDS, new
LinkedBlockingQueue<>());
```

```
    }


    @Override
    protected void beforeExecute(Thread t, Runnable r) {
        // Custom logic before executing a task
        super.beforeExecute(t, r);
    }


    @Override
    protected void afterExecute(Runnable r, Throwable t) {
        // Custom logic after executing a task
        super.afterExecute(r, t);
    }
}
```

Q - 91) What is the role of `Java Virtual Machine (JVM) options`?

: JVM options are parameters passed to the Java Virtual Machine at runtime to control various aspects of its behavior and performance. They include settings for memory allocation, garbage collection, debugging, and performance tuning. Examples are `-Xmx` for setting maximum heap size and `-XX:+UseG1GC` for enabling the G1 garbage collector.

Q - 92) How does `JIT Compilation` optimize Java code?

Just-In-Time (JIT) compilation optimizes Java code by converting bytecode into native machine code at runtime. This allows frequently executed code paths to be optimized for better performance, as the JIT compiler applies optimizations such as inlining and loop unrolling.

Q - 93) How does Java's `Synchronization` work under the hood?

Synchronization in Java works by using intrinsic locks (or monitors) associated with objects. When a synchronized block or method is accessed, the thread must acquire the lock associated with the object. If another thread holds the lock, the current thread will wait until the lock is released.

Q - 94) What are `Method Area` and `Heap Area` in JVM memory structure?

Method Area: Part of the JVM memory where class structures, method metadata, and static variables are stored. It is shared among all threads and used for loading class definitions.

Heap Area: Memory area where Java objects are allocated. It is managed by the garbage collector and is divided into young and old generations.

Q - 95) How do you handle `Memory Leaks` in Java applications?

Memory leaks in Java can be handled by:

Profiling: Using tools like VisualVM or YourKit to identify memory leaks.

Monitoring: Tracking object creation and garbage collection behavior.

Code Review: Ensuring proper resource management and avoiding unintentional object retention.

Q - 96) What is `Java Module System` and how does it affect code organization?

The Java Module System, introduced in Java 9, allows for modularization of Java applications. It enables developers to define modules with explicit dependencies and encapsulate code, improving maintainability, modularity, and versioning. It helps in organizing large codebases and controlling access to module internals.

Q - 97) . How does `Java's Exception Handling` model work with error recovery?

Java's exception handling model uses `try`, `catch`, and `finally` blocks to handle runtime errors. When an exception is thrown, control is transferred to the nearest `catch` block that matches the exception type. The `finally` block, if present, is executed regardless of whether an exception was thrown, ensuring resource cleanup.

Q - 98) What is `Java's ForkJoinPool`, and how does it facilitate parallelism?

`ForkJoinPool` is a special type of `ExecutorService` designed for parallelism with a work-stealing algorithm. It efficiently manages a pool of threads that can split tasks into smaller sub-tasks (fork) and later combine results (join). It is suitable for tasks that can be broken down into smaller parallelizable units.

Q - 99) . How do `Java's Type Erasure` and `Generics` affect type safety?

Type Erasure: Java's generics use type erasure to remove generic type information during compilation, replacing it with raw types. This means that generic type information is not available at runtime, affecting operations like type checking and casting.

Type Safety: Generics provide type safety at compile-time, but type erasure can lead to potential type-related issues at runtime, such as `ClassCastException`.

https://www.youtube.com/@codewitharrays

https://www.instagram.com/codewitharrays/

https://t.me/codewitharrays Group Link: https://t.me/ccee2025notes

+91 8007592194 +91 9284926333

codewitharrays@gmail.com

https://codewitharrays.in/project