Technical Interview Repository

MASTER LEARNING GUIDE

Total: 76 Questions | Generated: 16 June 2026

JAVA BACKEND DEVELOPER

1.

Coding Question – Merge Overlapping Time Slots

Given the following time slots:

`[(1,3), (2,6), (8,10), (15,18)]`

Each time slot contains:

- First value # Start time

- Second value # End time

Question:

Write a Java program to merge the overlapping time slots and print the final merged intervals. Also, determine how many final merged slots remain after merging.

Additionally, explain your complete approach in detail, including:

1: The logic used to identify overlapping intervals

2: Step-by-step execution flow

3: Low-level implementation details

4: Time and space complexity of the solution

```
# MODEL ANSWER
```

```
import java.util.*;

public class MergeIntervals {
    public static void main(String[] args) {
        // Input time slots
        int[][] intervals = { {1,3}, {2,6}, {8,10}, {15,18} };

        // Step 1: Sort intervals by start time
        Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));

        // Step 2: Use a list to store merged intervals
        List<int[]> merged = new ArrayList<>();

        for (int[] interval : intervals) {
            // If merged list is empty OR no overlap, add interval
            if (merged.isEmpty() ||
                        merged.get(merged.size()-1)[1] < interval[0]) {
                merged.add(interval);
            } else {
                // Overlap: merge by updating end time
                merged.get(merged.size()-1)[1] =
                 Math.max(merged.get(merged.size()-1)[1], interval[1]);
            }
        }

        // Printing merged intervals
        System.out.print("Merged intervals: ");
        for (int[] m : merged) {
            System.out.print("[" + m[0] + "," + m[1] + "] ");
        }
        System.out.println("\nNumber of merged slots: "
                                        + merged.size());
    }
}
```

Approach Explanation (Simple Steps):

1: **Sort the intervals** by their start time.

2: This makes it easy to compare each interval with the previous one.

3: **Iterate through the sorted intervals**:

- If the current interval does **not overlap** with the previous (merged) interval, add it to the result.
- If it **overlaps** (i.e., current start # previous end), merge them by updating the end time to the maximum of both ends.

4: **Continue merging** until all intervals are checked.

5: **Print the merged intervals** and count.

Logic to Identify Overlapping Intervals:

- Two intervals `[a, b]` and `[c, d]` **overlap** if `c # b`.
- If they overlap, update the end to `max(b, d)`.

Step-by-Step Execution Flow:

1: Input: `[(1,3), (2,6), (8,10), (15,18)]`

2: After sorting: `[(1,3), (2,6), (8,10), (15,18)]` (already sorted)

3: Merge:

- `(1,3)` and `(2,6)` overlap # merge to `(1,6)`
- `(1,6)` and `(8,10)` do not overlap # add `(8,10)`
- `(8,10)` and `(15,18)` do not overlap # add `(15,18)`

4: Final merged intervals: `[(1,6), (8,10), (15,18)]`

5: Number of merged slots: **3**

Low-Level Implementation Details:

- **Sorting:** Use `Arrays.sort()` with a custom comparator.
- **Merging:** Use an `ArrayList` to dynamically add or update intervals.
- **Output:** Print intervals and their count.

Time and Space Complexity:

- **Time Complexity:**Sorting: `O(n log n)`
- Merging: `O(n)`
- **Total:** `O(n log n)`
- **Space Complexity:**`O(n)` for storing merged intervals.

Summary:

Sort the intervals, merge overlapping ones, and print the result. This method is easy to remember and efficient for interview or practical use.

JAVA BACKEND DEVELOPER

2.

Microservices Communication

Question:

How do two microservices communicate with each other? Please explain the complete process in detail, including common communication patterns and implementation approaches.

MODEL ANSWER

Microservices Communication – Simple Explanation

Microservices communicate with each other mainly in two ways:

1: **Synchronous Communication (Direct)**

- One microservice sends a request to another and waits for a response.

- This is usually done using **HTTP/REST APIs** or **gRPC**.

Example: Service A calls Service B's API to get some data.

2: **Asynchronous Communication (Indirect)**

- Microservices send messages without waiting for a reply.

- This is done using **Message Brokers** like RabbitMQ, Kafka, or AWS SQS.

Example: Service A sends a message to a queue, Service B picks it up and processes it later.

Common Patterns:

1: **Request/Response -** Direct API calls between services.

2: **Publish/Subscribe -** One service publishes messages, others subscribe and receive them.

3: **Event-Driven -** Services react to events/messages from other services.

Implementation Steps:

1: Service A needs something from Service B.

2: Service A either:

- Calls Service B's API (synchronous), or

- Sends a message to a queue/topic (asynchronous).

3: Service B responds directly or processes the message and sends a reply/event if needed.

Key Points to Remember:

1: Use APIs for quick, direct communication.

2: Use message brokers for scalable, decoupled communication.

3: Choose the pattern based on your needs (real-time vs. batch, reliability, scalability).

Summary:

Microservices talk to each other using APIs (direct) or messages (indirect), depending on how fast and reliable the communication needs to be.

JAVA BACKEND DEVELOPER

3.

Asynchronous Programming in Microservices

Question:

Do you have experience with asynchronous programming?

How would you implement:

1: Synchronous communication

2: Asynchronous communication

between APIs in a Spring Boot microservices architecture?

MODEL ANSWER

Yes, I have experience with asynchronous programming.

In Spring Boot microservices

1: **Synchronous Communication**

- APIs directly call each other and wait for a response.

Example: Using RestTemplate or WebClient to make HTTP calls.

- The client waits until it gets the result before continuing.

2: **Asynchronous Communication**

- APIs send messages/events and do not wait for a reply.

Example: Using message brokers like RabbitMQ, Kafka, or JMS.

- The sender puts the message on a queue/topic, and the receiver processes it later.

Summary:

1: Use synchronous for quick, direct responses (HTTP calls).

2: Use asynchronous for decoupled, scalable, and non-blocking operations (messaging).

JAVA BACKEND DEVELOPER

4.

Many-to-Many Relationship in Spring JPA/Hibernate

Suppose there are two entities:

· `Student`

· `Course`

You need to implement a many-to-many relationship between them.

 # MODEL ANSWER

Many-to-Many Relationship in Spring JPA/Hibernate (Student & Course)

A **many-to-many relationship** means that a student can enroll in multiple courses, and each course can have multiple students.

Step-by-Step Simple Implementation:

1. **Student Entity**

```
@Entity
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @ManyToMany
    @JoinTable(
        name = "student_course",
        joinColumns = @JoinColumn(name = "student_id"),
        inverseJoinColumns = @JoinColumn(name = "course_id")
    )
    private Set<Course> courses = new HashSet<>();
}
```

2. **Course Entity**

```
@Entity
public class Course {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;

    @ManyToMany(mappedBy = "courses")
    private Set<Student> students = new HashSet<>();
}
```

Key Points to Remember

1: Use @ManyToMany annotation in both entities.

2: Use @JoinTable in one entity to define the join table (here, in Student).

3: Use mappedBy in the other entity (here, in Course) to make it bidirectional.

4: The join table (e.g., student_course) links both entities using their IDs.

Summary

This setup allows you to easily add/remove courses for students and vice versa. Spring JPA/Hibernate will automatically manage the join table for you.

This is the easiest way to implement a many-to-many relationship in Spring JPA /Hibernate!

JAVA BACKEND DEVELOPER

5.

Exception Handling with `@Transactional`

Question:

If an exception occurs inside a method annotated with `@Transactional`, what type of exception handling behavior will occur?

Will the transaction rollback for:

1: Checked exceptions?

2: Unchecked exceptions?

Please explain.

MODEL ANSWER

If an exception occurs inside a method annotated with @Transactional:

1: **Unchecked exceptions (like RuntimeException and Error)** - The transaction will automatically rollback.

2: **Checked exceptions (like Exception but not RuntimeException)** - By default, the transaction will NOT rollback.

Simple way to remember:

1: Rollback happens for unchecked exceptions.

2: No rollback for checked exceptions unless you specify it manually with rollbackFor in @Transactional.

Example:

```
@Transactional(rollbackFor = Exception.class)
```

This will make the transaction rollback for checked exceptions too.

JAVA BACKEND DEVELOPER

6.

Transaction Management in Spring Boot

Question:

Do you have experience with transaction management in Spring Boot?

Please explain:

1: What is transaction propagation?

2: What is transaction isolation?

3: What are the different propagation and isolation levels?

How are they used with the `@Transactional` annotation?

MODEL ANSWER

Transaction Management in Spring Boot

1: **What is transaction propagation?**

Propagation defines how transactions behave when one transactional method calls another. It decides if the called method should run in the same transaction or start a new one.

2: **What is transaction isolation?**

Isolation controls how one transaction sees data changes made by other transactions. It prevents problems like dirty reads, non-repeatable reads, and phantom reads.

3: **Different propagation levels**

REQUIRED: Uses current transaction or creates a new one if none exists (default).

REQUIRES_NEW: Always starts a new transaction, suspending the current one.

SUPPORTS: Uses current transaction if exists, else runs non-transactional.

NOT_SUPPORTED: Runs non-transactional, suspends any existing transaction.

MANDATORY: Must run within an existing transaction; throws error if none.

NEVER: Must NOT run within a transaction; throws error if one exists.

NESTED: Runs within a nested transaction if current exists.

4: **Different isolation levels**

DEFAULT: Uses database default.

READ_UNCOMMITTED: Can see uncommitted changes (least safe).

READ_COMMITTED: Only sees committed changes.

REPEATABLE_READ: Same data for repeated reads in a transaction.

SERIALIZABLE: Highest isolation; transactions run one after another.

5: Using **@Transactional** annotation

You can set propagation and isolation like this:

```
@Transactional(
    propagation = Propagation.REQUIRED,
    isolation = Isolation.READ_COMMITTED
)
public void yourMethod() {
    // business logic
}
```

Summary:

1: **Propagation:** Controls transaction flow between methods.

2: **Isolation:** Controls data visibility between transactions.

3: **@Transactional:** Lets you set both using simple attributes.

JAVA BACKEND DEVELOPER

7.

JWT – Stateless or Stateful

Question:

Are JWT (JSON Web Tokens) stateless or stateful? Explain the reasoning behind your answer.

MODEL ANSWER

JWT (JSON Web Tokens) are **stateless**.

Reason:

When you use JWT, all the information needed to verify the user is stored inside the token itself. The server does not need to keep any session or user data for each token. This means the server does not remember anything between requests—each request is independent and can be verified using the JWT alone.

In simple words:

JWTs are stateless because the server does not store any data about the user session; everything is inside the token. This makes JWT fast and easy to scale.

JAVA BACKEND DEVELOPER

8.

Machine Token vs User Token

In a Spring Boot-based web application:

Question:

What is the difference between a Machine Token and a User Token? Where and why would each be used?

MODEL ANSWER

Machine Token

1: Used by applications or services (machines), not by real people.

Example: When one backend service talks to another backend service.

2: It usually has fixed permissions and does not represent a specific user.

User Token

1: Used by real people (users) when they log in.

Example: When a user logs in to your web app and gets a token to use APIs.

2: It carries the user's identity and permissions.

Where and Why?

- Use **Machine Token** for automated, system-to-system communication (e.g., scheduled jobs, microservices).

- Use **User Token** for user-driven actions (e.g., accessing their data, performing tasks as themselves).

Easy Way to Remember:

1: Machine Token = Robot/Service

2: User Token = Person/Human

Machine tokens are for machines, user tokens are for people.

JAVA BACKEND DEVELOPER

9.

Securing REST Endpoints

You have developed multiple REST APIs and need to secure them from unauthorized access.

Question:

How would you secure REST endpoints in a Spring Boot application? Explain the security mechanism and implementation approach.

 # MODEL ANSWER

To secure REST endpoints in a Spring Boot application, follow these simple steps:

1: **Add Spring Security Dependency**

Include spring-boot-starter-security in your project's pom.xml or build.gradle.

2: **Configure Security Rules**

- Create a class (e.g., SecurityConfig) and extend WebSecurityConfigurerAdapter.

- Override the configure(HttpSecurity http) method to specify which endpoints require authentication.

3: **Use Authentication**

By default, Spring Security uses basic authentication. You can also use JWT (JSON Web Token) or OAuth2 for more secure options.

4: **Set Up User Details**

Define users and roles in memory or connect to a database for user management.

Example Implementation:

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
            .antMatchers("/api/public").permitAll() // open endpoint
            .antMatchers("/api/secure").authenticated() // secured endpo
            .and()
            .httpBasic(); // enables basic authentication
    }
}
```

Summary:

1: Add Spring Security

2: Configure which endpoints are secure

3: Use authentication (Basic, JWT, OAuth2)

4: Manage users and roles

This way, only authorized users can access your REST APIs, keeping them safe from unauthorized access.

JAVA BACKEND DEVELOPER

10.

REST API Rate Limiting

You have developed a REST API endpoint, and you need to restrict it so that no user/client can call the endpoint more than 10 times per second. If someone tries to access the API more than the allowed limit, the request should be blocked.

Question:

How would you implement this rate-limiting mechanism in a Spring Boot REST API? Explain the approach and technologies/libraries you would use.

 # MODEL ANSWER

To implement rate limiting in a Spring Boot REST API so that no user/client can call the endpoint more than **10 times per second**, follow these simple steps:

1: **Use a Rate Limiting Library**

The easiest way is to use a library like **Bucket4j** or **resilience4j**. These libraries help you set limits without writing complex code.

2: **How It Works**

- For each user/client, you create a “bucket” that allows a maximum of 10 requests per second.

- Every time a user makes a request, the bucket checks if the limit is reached.

- If the bucket is empty (more than 10 requests in a second), the request is blocked.

3: **Implementation Steps (with Bucket4j example)**

- Add Bucket4j dependency in your pom.xml.

- Use a filter or interceptor to check the rate limit before processing the request.

- Identify the user/client (e.g., by IP address or API key).

- If the limit is exceeded, return HTTP 429 (Too Many Requests).

Sample Code:

```
// Add Bucket4j dependency
// In your filter/interceptor:
Bucket bucket = Bucket4j.builder()
    .addLimit(Bandwidth.simple(10, Duration.ofSeconds(1)))
    .build();

if (bucket.tryConsume(1)) {
    // Allow request
} else {
    // Block request, return 429 error
}
```

Summary:

1: Use Bucket4j or resilience4j for simple rate limiting.

2: Set 10 requests per second per user.

3: Block requests exceeding the limit with HTTP 429.

JAVA BACKEND DEVELOPER

11.

Caching & Load Handling

- Have you implemented caching in Spring Boot?

- How does caching help in handling load and improving performance?

MODEL ANSWER

Caching & Load Handling

1: **Have you implemented caching in Spring Boot?**

Yes, I have implemented caching in Spring Boot using the @Cacheable annotation. This allows methods to store their results in a cache so that repeated calls with the same parameters return the cached value instead of executing the method again.

Example:

```
@Cacheable("products")
public Product getProductById(Long id) {
    // Fetch product from database
    return productRepository.findById(id);
}
```

In this example, when getProductById is called with the same id, the result is retrieved from the cache instead of querying the database again.

2. **How does caching help in handling load and improving performance?**

Caching helps by:

- **Reducing database calls:** Frequently accessed data is served from the cache, not the database, which lowers the load on the database.

- **Faster response times:** Data is returned quickly from the cache, improving user experience.

- **Handling high traffic:** When many users request the same data, caching prevents repeated expensive operations, allowing the application to handle more requests efficiently.

Example:

If 100 users request the same product details, without caching, the database will be hit 100 times. With caching, only the first request fetches from the database; the next 99 get the data from the cache.

Summary:

Caching in Spring Boot is easy to implement and helps your application handle more users, respond faster, and reduce the strain on your database.

JAVA BACKEND DEVELOPER

12.

Handling High Load (E-commerce Scenario)

- In an e-commerce system, how would you handle situations where many users perform the same action simultaneously?

- How would you manage load on both server and database sides?

MODEL ANSWER

Handling High Load in E-commerce

When many users perform the same action at once, like checking out or searching for products, follow these steps:

1: **Use Load Balancers** - Spread user requests across multiple servers so no single server gets overloaded.

2: **Cache Frequently Used Data** - Store popular items or pages in memory (cache) so the database doesn't get hit every time.

3: **Database Optimization** - Use indexing and efficient queries, and consider read replicas to handle more users.

4: **Queue Requests** - For actions like placing orders, use queues to process them one by one, avoiding conflicts and overload.

5: **Auto-Scaling** - Automatically add more servers when traffic increases.

6: **Limit User Actions** - Set limits (rate limiting) so users can't send too many requests in a short time.

Summary:

Balance the load across servers, use caching, optimize the database, queue heavy actions, scale automatically, and limit requests. This keeps the system fast and reliable, even when many users are active.

JAVA BACKEND DEVELOPER

13.

Service-to-Service Communication

- In synchronous communication, how does one microservice call another?

MODEL ANSWER

In synchronous communication between microservices, one microservice sends a request (usually via HTTP or API call) to another microservice and waits for a response. The calling microservice cannot proceed until it receives the reply. This is similar to making a phone call and waiting for the other person to answer before continuing the conversation.

Example:

Microservice A needs user details, so it sends an HTTP request to Microservice B's API endpoint. Microservice B processes the request and sends back the user details as a response. Microservice A waits for this reply before continuing its work.

JAVA BACKEND DEVELOPER

14.

Microservices Communication

- Can you explain synchronous vs asynchronous communication between microservices?

- When should each approach be used, and how have you implemented them?

MODEL ANSWER

Synchronous vs Asynchronous Communication in Microservices

Synchronous Communication:

1: One service calls another and waits for a response (like a phone call).

2: Example: REST API calls.

3: Use when you need immediate feedback or real-time processing.

Asynchronous Communication:

1: One service sends a message and continues without waiting (like sending an email).

2: Example: Message queues (Kafka, RabbitMQ).

3: Use when you want better performance, reliability, or do not need instant response.

When to Use Each:

1: Use synchronous for quick, direct interactions (e.g., user login).

2: Use asynchronous for tasks that take time or can happen in the background (e.g., sending notifications).

Implementation Experience:

1: I have used synchronous REST APIs for real-time data exchange.

2: I have implemented asynchronous messaging for processing orders and sending emails, using Kafka and RabbitMQ.

Easy to Remember:

Phone call = synchronous (wait for reply)

Email = asynchronous (send and move on)

JAVA BACKEND DEVELOPER

15.

Observability & Monitoring

- Have you used any observability tools to monitor logs and application performance?

- If yes, how did you implement it?

MODEL ANSWER

Yes, I have used observability tools like Splunk, ELK Stack, and Prometheus to monitor logs and application performance.

Here’s how I implemented it, in simple steps:

1: **Installed the observability tool** (e.g., Splunk, ELK, Prometheus) on our servers.

2: **Configured the application** to send logs and performance data to the tool.

3: **Set up dashboards** to visualize real-time logs and metrics.

4: **Created alerts** for errors or performance issues, so we get notified quickly.

5: **Regularly reviewed the dashboards and alerts** to make sure everything was running smoothly.

This helped us quickly detect and fix problems, improve performance, and keep the application reliable.

JAVA BACKEND DEVELOPER

16.

Logging

- Which logging framework/mechanism are you currently using in your projects?

MODEL ANSWER

We are currently using Log4j as our main logging framework to record application events and errors. Along with Log4j, we also use Dynatrace for advanced monitoring and analysis. Log4j helps us capture logs, while Dynatrace gives us real-time insights into application performance, detects issues, and helps us troubleshoot faster. Together, these tools make our logging and monitoring process more effective and reliable.

JAVA BACKEND DEVELOPER

17.

Stream API Mapping

- In the Stream API, what is the difference between map() and mapToObj()?

- When should each be used?

MODEL ANSWER

Stream API Mapping: map() vs mapToObj()

1: map() is used to transform each element in a stream to another value. For example, you can convert a list of numbers to their squares.

2: mapToObj() is used when you are working with primitive streams (like IntStream, DoubleStream) and want to convert them to an object stream (Stream of objects).

When to use:

1: Use map() for regular streams (Stream) when you want to change the value but keep it as an object.

2: Use mapToObj() for primitive streams (like IntStream) when you want to turn primitives into objects.

Easy way to remember:

map() = change objects to other objects.

mapToObj() = change primitives to objects.

Example:

Stream<String>.map(String::length) # Stream

IntStream.mapToObj(i -> "Number: " + i) # Stream

Summary:

Use map() for object streams, mapToObj() for primitive streams to convert to objects.

JAVA BACKEND DEVELOPER

18.

Java 8 Features

- What Java 8 features or APIs have you frequently used in your projects?

MODEL ANSWER

Java 8 Features I Frequently Use

1: **Lambda Expressions:** Makes code shorter and easier to read by allowing you to write functions inline.

```
list.forEach(item -> System.out.println(item));
```

2: **Streams API:** Helps process collections (like lists) in a simple and efficient way, such as filtering, sorting, or mapping data.

```
List<String> filtered = list.stream().filter(s -> s.startsWith("A")).col
```

3: **Functional Interfaces:** Used with lambda expressions, like Predicate, Function, and Consumer.

```
Predicate<Integer> isEven = n -> n % 2 == 0;
```

4: **Default Methods in Interfaces:** Allows adding new methods to interfaces without breaking existing code.

```
interface MyInterface { default void show() { System.out.println("Hello"
```

5: **Optional Class:** Helps avoid NullPointerException by handling optional values safely.

```
Optional<String> name = Optional.ofNullable(getName());
```

6: **Date and Time API (java.time):** Makes working with dates and times much easier and less error-prone.

```
LocalDate today = LocalDate.now();
```

These features help write cleaner, faster, and safer code in Java.

JAVA BACKEND DEVELOPER

19.

Exception Handling

- How do you handle exceptions in your project?

- Can you explain Global Exception Handling in detail?

MODEL ANSWER

Exception Handling

In my project, I handle exceptions by using try-catch blocks. Whenever something goes wrong, like an error or unexpected situation, the code inside the try block is stopped and the catch block runs. This helps prevent the program from crashing and lets us show a user-friendly message or log the error.

Global Exception Handling

Global Exception Handling means catching all unhandled errors in one central place, usually at the top level of the application. This is useful because:

1: It makes sure no error goes unnoticed.

2: It allows us to log all errors and send proper responses to users.

3: In frameworks like Spring Boot (Java), we use @ControllerAdvice and @ExceptionHandler annotations to handle exceptions globally.

4: In .NET, we use middleware or filters for global exception handling.

In summary:

Global Exception Handling is like having a safety net for your application, so that every error is caught and managed, making your project more stable and reliable.

JAVA BACKEND DEVELOPER

20.

@Qualifier Annotation

- What is the purpose of the @Qualifier annotation in Spring?

MODEL ANSWER

The **@Qualifier** annotation in Spring is used to specify which bean should be injected when there are multiple beans of the same type. It helps Spring know exactly which one you want.

In simple words:

When you have more than one bean of the same kind, use @Qualifier to tell Spring which one to use.

Example:

If you have two beans called "apple" and "orange", and you want Spring to use "apple", you write:

```
@Autowired
@Qualifier("apple")
private Fruit fruit;
```

This way, Spring will inject the "apple" bean into your code.

JAVA BACKEND DEVELOPER

21.

Multiple Beans of Same Type

- In a configuration class, can you define multiple beans with the same return type (e.g., Employee)?

- If yes, how would you differentiate and use them?

MODEL ANSWER

Yes, you can define multiple beans of the same type (e.g., Employee) in a configuration class. To differentiate and use them, you assign each bean a unique name using the @Bean annotation or @Qualifier.

Example:

```
@Bean(name = "manager")
public Employee manager() {
    return new Employee("Manager");
}

@Bean(name = "developer")
public Employee developer() {
    return new Employee("Developer");
}
```

How to use them:

Inject the specific bean using @Qualifier:

```
@Autowired
@Qualifier("manager")
private Employee managerEmployee;

@Autowired
@Qualifier("developer")
private Employee developerEmployee;
```

Summary:

1: You can have multiple beans of the same type.

2: Use unique names and @Qualifier to select the one you need. This makes it easy to manage and use different beans of the same class in your application.

JAVA BACKEND DEVELOPER

22.

Configuration Classes

- What is the purpose of a configuration class in Spring Boot?

- Why is bean configuration required?

MODEL ANSWER

- The purpose of a configuration class in Spring Boot is to organize and manage the setup of your application. It tells Spring which components (beans) to create and how they should work together.

- Bean configuration is required because Spring needs to know how to build and connect different parts of your app. By configuring beans, you make sure that everything is ready and works correctly when the app starts.

In short:

1: Configuration classes help Spring know what to do.

2: Bean configuration makes sure all parts are set up and connected properly.

JAVA BACKEND DEVELOPER

23.

Injecting One Class into Another

- If you have two classes, how would you inject one class into another using Spring Boot features?

MODEL ANSWER

To inject one class into another in Spring Boot, follow these simple steps:

1: **Add @Component or @Service to the class you want to inject**

```
@Component
public class ClassA {
    // your code here
}
```

2: **Use @Autowired in the class where you want to inject it**

```
@Component
public class ClassB {
    @Autowired
    private ClassA classA; // ClassA is injected here
}
```

Summary:

1: Mark the class to be injected with @Component or @Service.

2: Use @Autowired in the other class to inject it automatically.

JAVA BACKEND DEVELOPER

24.

Dependency Injection

- What are the different ways to implement dependency injection in Spring Boot?

- Which approach is recommended and why?

MODEL ANSWER

Dependency Injection in Spring Boot

There are three main ways to implement dependency injection in Spring Boot:

1: **Constructor Injection**

Inject dependencies using the class constructor.

Example:

```
public MyService(MyRepository repo) { ... }
```

2: **Setter Injection**

Inject dependencies using setter methods.

Example:

```
public void setRepo(MyRepository repo) { ... }
```

3: **Field Injection**

Inject dependencies directly into fields using @Autowired.

Example:

```
@Autowired private MyRepository repo;
```

Recommended Approach:

Constructor injection is generally recommended because:

1: It makes dependencies clear and required.

2: It helps with testing and immutability.

3: It avoids issues related to Spring's internal processing.

Summary:

Use constructor injection for most cases in Spring Boot. It's clean, safe, and easy to test.

JAVA BACKEND DEVELOPER

25.

Entity vs DTO

- What is the difference between an Entity class and a DTO (Data Transfer Object)?

MODEL ANSWER

Entity:

An Entity class represents the actual data stored in the database. It usually includes all fields and relationships needed for persistence.

DTO (Data Transfer Object):

A DTO is used to transfer data between processes, layers, or systems. It contains only the data needed for a specific operation, and may exclude extra fields or logic.

Easy way to remember:

Entity = Database object (full data, persistent)

DTO = Data carrier (only what's needed, for transfer)

Entity Example:

```
public class User {
    private Long id;
    private String name;
    private String email;
    private String password;
    // getters and setters
}
```

DTO Example:

```
public class UserDTO {
    private String name;
    private String email;
    // getters and setters
}
```

Summary:

1: Entity has all fields, including database-related ones (like id, password).

2: DTO has only the fields needed for transfer (like name, email).

JAVA BACKEND DEVELOPER

26.

End-to-End Flow

- Can you explain the complete end-to-end request flow in a Spring Boot application?

- How does a request travel from the client to the database and back, including all intermediate components?

MODEL ANSWER

End-to-End Request Flow in Spring Boot

1: **Client Sends Request**

The client (browser, mobile app, etc.) sends an HTTP request to the server.

2: **Controller Receives Request**

Spring Boot's controller catches the request and decides what to do.

3: **Service Layer Processes Logic**

The controller calls the service layer, which handles business logic (calculations, rules, etc.).

4: **Repository Accesses Database**

The service calls the repository layer, which interacts with the database to fetch or save data.

5: **Database Returns Data**

The database sends the requested data back to the repository.

6: **Service Layer Handles Data**

The repository passes the data to the service layer, which may process it further.

7: **Controller Sends Response**

The service returns the final data to the controller, which sends an HTTP response back to the client.

In summary:

Client # Controller # Service # Repository # Database # Repository # Service # Controller # Client

JAVA FULLSTACK DEVELOPER (REACTJS)

27.

Write a Java program to find the Employee who contributed the maximum transaction amount in the last 30 days.

 # MODEL ANSWER

Below is a sample Java program that accomplishes this task. The program assumes you have a list of Transaction objects, each containing an employee ID, transaction amount, and transaction date. It finds the employee whose total transaction amount in the last 30 days is the highest.

```
import java.time.LocalDate;
import java.util.*;

class Transaction {
    String employeeId;
    double amount;
    LocalDate date;

    public Transaction(String employeeId, double amount,
                                        LocalDate date) {
        this.employeeId = employeeId;
        this.amount = amount;
        this.date = date;
    }
}

public class MaxTransactionEmployee {
    public static void main(String[] args) {
        // Sample transactions
        List<Transaction> transactions = Arrays.asList(
            new Transaction("E001", 500, LocalDate.now().minusDays(2)),
            new Transaction("E002", 700, LocalDate.now().minusDays(5)),
            new Transaction("E001", 300, LocalDate.now().minusDays(10)),
            new Transaction("E003", 1200, LocalDate.now().minusDays(20))
            new Transaction("E002", 400, LocalDate.now().minusDays(25)),
            new Transaction("E003", 600, LocalDate.now().minusDays(35))
                                                    // Outside 30 days
        );

        // Find employee with maximum transaction amount in last 30 days
        String maxEmployee = findMaxEmployee(transactions);
        System.out.println("Employee with maximum transaction
                                amount in last 30 days: " + maxEmployee);
    }

    public static String findMaxEmployee
                                        (List<Transaction> transactions) {
```

```
        LocalDate today = LocalDate.now();
        LocalDate thresholdDate = today.minusDays(30);

        // Map to store total amount per employee
        Map<String, Double> employeeTotals = new HashMap<>();

        for (Transaction tx : transactions) {
            if (!tx.date.isBefore(thresholdDate)) {
                employeeTotals.put(
                    tx.employeeId,
                    employeeTotals.getOrDefault(tx.employeeId, 0.0)
                                                            + tx.amount
                );
            }
        }

        // Find employee with maximum total
        String maxEmployee = null;
        double maxAmount = 0.0;
        for (Map.Entry<String, Double> entry :
                                    employeeTotals.entrySet()) {
            if (entry.getValue() > maxAmount) {
                maxAmount = entry.getValue();
                maxEmployee = entry.getKey();
            }
        }
        return maxEmployee;
    }
}
```

NOTE: The **getOrDefault()** method in Java's HashMap is used to retrieve the value associated with a specified key, or return a default value if the key does not exist in the map.

How It Works

1: Transaction Class: Represents a transaction with employee ID, amount, and date.

2: Main Method: Creates sample transactions and calls the method to find the employee with the highest total in the last 30 days.

3: findMaxEmployee:

- Filters transactions within the last 30 days.

- Sums amounts per employee.

- Finds the employee with the maximum total.

JAVA FULLSTACK DEVELOPER (REACTJS)

28.

How you are handling exceptions in your applications globally, not in method level or class level but in application level?

MODEL ANSWER

To handle exceptions globally in my application, I use a global exception handler. This is a central place in the application (not inside individual methods or classes) where all unhandled errors are caught. For example, in Java Spring Boot, I use **@ControllerAdvice** with **@ExceptionHandler**. This way, any error that happens anywhere in the application is handled in one place, allowing me to log the error, show a user-friendly message, and keep the application running smoothly. This makes exception management easier and more consistent across the whole application.

Example

In a Spring Boot application, I create a class with @ControllerAdvice and write a method with @ExceptionHandler to catch all exceptions globally:

```
@ControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler(Exception.class)
    public ResponseEntity<String> handleAllExceptions(Exception ex) {
        // Log the exception and return a user-friendly message
        return new ResponseEntity<>
              ("Something went wrong. Please try again later.",
                                    HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

This way, any unhandled exception in the whole application will be caught by this handler.

JAVA FULLSTACK DEVELOPER (REACTJS)

29.

What kind of interceptors you have used in Spring to intercept the request and response details of a controller endpoint?

MODEL ANSWER

In Spring, to intercept the request and response details of a controller endpoint, we commonly use two main types of interceptors:

1: **HandlerInterceptor**

- This is an interface provided by Spring MVC.

- We can create our own class by implementing `HandlerInterceptor` and override methods like:

  - `preHandle()` – runs before the controller method.
  - `postHandle()` – runs after the controller method but before the response is sent.
  - `afterCompletion()` – runs after the complete request is finished.

- This is useful for logging, authentication, or modifying requests/responses.

2: **Filter**

- This is a standard Servlet feature, but Spring Boot supports it easily.

- We can create a class that implements `javax.servlet.Filter`.

- The `doFilter()` method lets us intercept both the request and response before they reach the controller and after the response leaves the controller.

In summary:

1: Use `HandlerInterceptor` for Spring-specific interception.

2: Use `Filter` for more general or lower-level interception.

Both can be used to log, check, or modify request and response details in a Spring application.

JAVA FULLSTACK DEVELOPER (REACTJS)

30.

How can I intercept the request coming from frontend UI to certain endpoint and how can I intercept the response of that endpoint which our backend controller is sending to frontend?

MODEL ANSWER

To intercept requests and responses between the frontend (ReactJS) and backend (Spring Boot), follow these steps:

Intercepting Requests from ReactJS

1: **In ReactJS**, you can use tools like **Axios interceptors** or **Fetch wrappers**.

2: **How to do it with Axios -**

```
// Add a request interceptor
axios.interceptors.request.use(function (config) {
  // Do something before request is sent
  return config;
}, function (error) {
  return Promise.reject(error);
});
```

Intercepting Responses in ReactJS

With Axios -

```
// Add a response interceptor
axios.interceptors.response.use(function (response) {
  // Do something with response data
  return response;
}, function (error) {
  return Promise.reject(error);
});
```

Intercepting Requests and Responses in Spring Boot

1: **In Spring Boot**, use a **Filter** or **HandlerInterceptor**.

2: **Example using HandlerInterceptor -**

```
public class CustomInterceptor implements HandlerInterceptor {
    @Override
    public boolean preHandle(HttpServletRequest request,
                  HttpServletResponse response, Object handler) {
        // This runs before controller (intercept request)
        return true;
    }
    @Override
    public void postHandle(HttpServletRequest request,
           HttpServletResponse response, Object handler,
                                ModelAndView modelAndView) {
        // This runs after controller (intercept response)
    }
}
```

3: **Register the interceptor -**

```
@Configuration
public class WebConfig implements WebMvcConfigurer {
    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(new CustomInterceptor());
    }
}
```

Summary:

1: Use Axios interceptors in ReactJS to catch requests/responses on the frontend.

2: Use HandlerInterceptor (or Filter) in Spring Boot to intercept requests/responses on the backend.

This approach is simple and works for most SpringBoot-ReactJS applications!

JAVA FULLSTACK DEVELOPER (REACTJS)

31.

How can we avoid possibility of circular dependency using constructor injection approach?

 # MODEL ANSWER

To avoid circular dependency using constructor injection, make sure that two classes do not depend on each other directly through their constructors. If Class A needs Class B, and Class B also needs Class A, this creates a loop that cannot be resolved.

How to avoid it:

1: Refactor your code so only one class depends on the other via constructor.

2: Use interfaces or abstractions if needed.

3: Consider using setter injection or property injection for one of the classes, instead of constructor injection.

In short:

Never let two classes require each other in their constructors. Break the cycle by changing one dependency to a different injection method.

Problem: Circular Dependency with Constructor Injection

```
class A {
    private B b;
    public A(B b) { this.b = b; }
}

class B {
    private A a;
    public B(A a) { this.a = a; }
}
```

This code will cause a circular dependency because A needs B and B needs A via constructors.

Solution: Break the Cycle Using Setter Injection

```
class A {
    private B b;
    public A(B b) { this.b = b; }
}

class B {
    private A a;
    public B() { }
    public void setA(A a) { this.a = a; }
}
```

Now, only A depends on B via constructor, and B gets A using a setter method. This avoids the circular dependency problem.

Summary:

Use constructor injection for one class and setter (or property) injection for the other to break the circular dependency.

JAVA FULLSTACK DEVELOPER (REACTJS)

32.

What is circular dependency?

MODEL ANSWER

Circular dependency happens when two or more things depend on each other directly or indirectly, creating a loop that can cause problems.

Example:

Imagine there are two files in a program:

1: File A needs something from File B

2: File B also needs something from File A

Because both files are waiting for each other, it creates a circle, and the program might not work properly.

In short:

Circular dependency is like two friends who each refuse to start a task until the other does, so nothing ever gets done.

Real-Time Example: Student and Course

Suppose you are designing a school system where:

1: Each Student is enrolled in a Course.

2: Each Course keeps track of the Student as a representative.

If both classes directly reference each other in their fields, you create a circular dependency.

Student.java

```
public class Student {
    private String name;
    private Course course;
             // Student is enrolled in a Course

    public Student(String name, Course course) {
        this.name = name;
        this.course = course;
    }
}
```

Course.java

```
public class Course {
    private String title;
    private Student representative;
              // Course has a Student as representative

    public Course(String title, Student representative) {
        this.title = title;
        this.representative = representative;
    }
}
```

What happens?

1: When you try to create a Student, you need a Course.

2: When you try to create a Course, you need a Student.

3: This can make it difficult to initialize objects without running into issues, because each one requires the other to exist first.

How to avoid this?

1: Use setter methods to assign one property after initial creation.

2: Use interfaces or design patterns to decouple the classes.

JAVA FULLSTACK DEVELOPER (REACTJS)

33.

What is the difference between Serialization or Serializable and Externalization or Externalizable?

MODEL ANSWER

Serialization (Serializable):

1: Serialization is the process of converting an object into a byte stream so it can be saved to a file or sent over a network.

2: If a class implements Serializable, Java automatically handles the serialization of all its fields.

3: You don’t need to write any extra code, just implement the interface.

Example:

```
class Student implements Serializable {
    int id;
    String name;
}
```

Here, all fields (id, name) will be serialized automatically.

Externalization (Externalizable):

1: Externalization is also used to convert an object into a byte stream, but gives you full control over what and how to serialize.

2: You must implement the writeExternal() and readExternal() methods to manually specify which fields to serialize and how.

3: Useful when you want to exclude some fields or customize the process.

Example:

```
class Student implements Externalizable {
    int id;
    String name;

    public void writeExternal(ObjectOutput out)
                                        throws IOException {
        out.writeInt(id);
        out.writeObject(name);
    }

    public void readExternal(ObjectInput in)
                        throws IOException, ClassNotFoundException {
        id = in.readInt();
        name = (String) in.readObject();
    }
}
```

Here, you decide which fields to save and how.

Easy Way to Remember:

1: **Serializable** - Java does everything for you, just implement the interface.

2: **Externalizable** - You do everything yourself, write the methods to control serialization.

Summary Table:

Feature	Serializable	Externalizable
Control	Automatic	Manual
Methods		
Required	None	writeExternal/readExternal
Usage	Simple objects	Custom serialization

In Short:

Serializable = automatic, easy

Externalizable = manual, flexible

JAVA FULLSTACK DEVELOPER (REACTJS)

34.

How can you create a custom immutable class?

MODEL ANSWER

An immutable class is a class whose objects cannot be changed after creation. To make a class immutable in Java, follow these simple steps:

1: **Make the class final** so it cannot be subclassed.

2: **Make all fields private and final** so they cannot be changed after initialization.

3: **Do not provide any setter methods**.

4: **Initialize all fields via the constructor**.

5: **If any field is an object, return a copy in the getter method** (to prevent changes from outside).

Example:

```
public final class Person {
    private final String name;
    private final int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
}
```

Once you create a Person object, you cannot change its name or age, making it immutable.

Key points to remember:

1: Use final for class and fields.

2: Only getters, no setters.

3: All fields set in constructor.

JAVA FULLSTACK DEVELOPER (REACTJS)

35.

Can you explain what is immutable class?

MODEL ANSWER

An immutable class is a class whose objects cannot be changed after they are created.

Key Points:

1: Once you create an object of an immutable class, you cannot modify its values.

2: All fields (variables) in the class are usually marked as final and private.

3: There are no setter methods—only getter methods to access the values.

Example:

```
final class Student {
    private final String name;
    private final int age;

    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
}
```

Here, once a Student object is created, you cannot change its name or age.

In simple words:

Immutable means "unchangeable." Just like your birth date, once set, it cannot be changed!

JAVA FULLSTACK DEVELOPER (REACTJS)

36.

What are the major functions you have used in ReactJS? Can you explain those uses in detail?

MODEL ANSWER

The major functions I use in ReactJS are:

1: useState()

This function lets us add state to our components.

Example: We use it to store and update values like user input or a counter.

2: useEffect()

This function helps us run code when the component loads, updates, or unmounts.

Example: We use it to fetch data from an API or set up subscriptions.

3: useContext()

This function lets us use data from a context without passing props manually.

Example: We use it for things like user authentication or theme settings.

4: useRef()

This function gives us a way to access and store a value that doesn't cause re-renders when changed.

Example: We use it to get a reference to a DOM element or to keep a mutable variable.

5: useMemo()

This function helps us optimize performance by "remembering" the result of a calculation until its dependencies change.

Example: We use it to avoid recalculating expensive operations on every render.

6: useCallback()

This function returns a memoized version of a function, so it only changes if its dependencies change.

Example: We use it to prevent unnecessary re-renders of child components that use callback functions.

In summary:

- **useState** for storing values,

- **useEffect** for side effects,

- **useContext** for shared data,

- **useRef** for references,

- **useMemo** for memoizing values,

- **useCallback** for memoizing functions.

These functions make React apps interactive, efficient, and easy to manage!

JAVA FULLSTACK DEVELOPER (REACTJS)

37.

What are the different Hooks you have used in ReactJS?

MODEL ANSWER

1: **useState**

Used to create state variables in functional components.

```
const [count, setCount] = useState(0);
```

2: **useEffect**

Used to run side effects (like fetching data or updating the DOM).

```
useEffect(() => {
  console.log('Component mounted');
}, []);
```

3: **useContext**

Used to access values from React Context.

```
const user = useContext(UserContext);
```

4: **useRef**

Used to create a reference to a DOM element or a value that persists across renders.

```
const inputRef = useRef(null);
```

5: **useMemo**

Used to memoize expensive calculations.

```
const result = useMemo(() => expensiveCalculation(a, b), [a, b]);
```

6: **useCallback**

Used to memoize functions so they aren’t recreated every render.

```
const handleClick = useCallback(() => {
  alert('Clicked!');
}, []);
```

7: **useReducer**

Used for more complex state logic (like Redux, but simpler).

```
const [state, dispatch] = useReducer(reducer, initialState);
```

Tip:

1: Hooks always start with “use”.

2: They help manage state, side effects, and other React features in functional components.

You can remember the main hooks as: **useState, useEffect, useContext, useRef, useMemo, useCallback, useReducer**

JAVA FULLSTACK DEVELOPER (REACTJS)

38.

In ReactJS, why we used setState() function, why don’t we directly update the state like normal JavaScript?

MODEL ANSWER

In ReactJS, we use the `setState()` function to update the state because React needs to know when the state changes so it can re-render the UI. If we update the state directly (like normal JavaScript), React will not know about the change, and the UI will not update automatically.

Think of `setState()` as a way to tell React, "Hey, something changed! Please update the screen."

Directly changing the state is like changing something behind React's back, React won't notice, so the page might not show the latest data.

In short:

1: Use `setState()` so React can keep everything in sync and update the UI properly.

2: Don't update state directly, or the UI might not work as expected.

Example:

Suppose you have a counter in React:

```
// WRONG way: Directly updating state
this.state.count = this.state.count + 1; // UI will NOT update


// CORRECT way: Using setState()
this.setState({ count: this.state.count + 1 }); // UI WILL update
```

Explanation

1: If you change count directly, React won't know and won't update the UI.

2: If you use setState(), React knows the state changed and will update the UI to show the new value.

JAVA FULLSTACK DEVELOPER (REACTJS)

39.

What are the advantages of function based components as compared to class based components?

MODEL ANSWER

Advantages of Function-Based Components vs. Class-Based Components:

1: **Easier to write and read:** Function components are shorter and less complex than class components.

2: **No ‘this’ keyword confusion:** You don’t have to worry about using ‘this’, which can be tricky in class components.

3: **Hooks make things simpler:** Hooks allow you to use state and other features easily in function components.

4: **Better code reuse:** It’s easier to reuse logic and split code into smaller pieces with function components.

5: **Faster performance:** Function components can be faster and lighter because they don’t have extra overhead.

6: **Modern and recommended:** Function components are now the standard in React and most new features are built for them.

JAVA FULLSTACK DEVELOPER (REACTJS)

40.

Difference between function based components and class based components?

MODEL ANSWER

Function Based Components:

These are written using simple JavaScript functions. They are easier to read and write. With React Hooks, function components can do everything class components can, like manage state and lifecycle.

Class Based Components:

These use JavaScript classes. You need to use special methods like `render()`, and manage state with `this.state`. They were used for complex features before Hooks were introduced.

Easy way to remember:

1: Function components = simple functions, modern, and preferred.

2: Class components = classes, older style, more code.

Summary:

Use function components for most cases. Class components are now mostly used in old code.

JAVA FULLSTACK DEVELOPER (REACTJS)

41.

What are the features you have used in ReactJS?

MODEL ANSWER

Features I have used in ReactJS:

1: **Components:** Building blocks of UI, reusable and independent.

2: **JSX:** Writing HTML-like code inside JavaScript.

3: **Props:** Passing data from parent to child components.

4: **State:** Managing data inside a component.

5: **Hooks (like useState, useEffect):** Special functions to use state and lifecycle in functional components.

6: **Conditional Rendering:** Showing content based on certain conditions.

7: **Lists and Keys:** Displaying lists of data efficiently.

8: **Event Handling:** Handling user actions like clicks.

9: **Routing (React Router):** Navigating between pages.

10: **Context API:** Sharing data easily across components.

These features help to create interactive, dynamic, and maintainable web applications with ReactJS.

JAVA BACKEND DEVELOPER

42.

What is the scope of Bean Injection Problem in Spring?

MODEL ANSWER

Scope of Bean Injection Problem in Spring:

The scope of the Bean Injection Problem in Spring refers to issues that can happen when beans with different scopes are injected into each other. For example, if you inject a prototype-scoped bean into a singleton-scoped bean, the singleton will only get one instance of the prototype bean, not a new one every time. This can cause unexpected behavior.

In short:

Bean injection problems usually occur when beans with different lifecycles (scopes) are mixed together without proper handling. Always be careful about bean scopes when injecting beans in Spring.

JAVA BACKEND DEVELOPER

43.

What are the different Transaction Propagation Levels in Spring?

MODEL ANSWER

Transaction Propagation Levels in Spring (Simple Explanation):

1: **REQUIRED** – Join the current transaction if one exists; otherwise, start a new one.

(Most common, default option.)

2: **REQUIRES_NEW** – Always start a new transaction, suspending any existing one.

3: **SUPPORTS** – Join the current transaction if one exists; if not, run without a transaction.

4: **NOT_SUPPORTED** – Run without a transaction, suspending any existing one.

5: **MANDATORY** – Must run inside an existing transaction; throws an error if none exists.

6: **NEVER** – Must run without a transaction; throws an error if a transaction exists.

7: **NESTED** – Runs within a nested transaction if a current transaction exists; otherwise, acts like REQUIRED.

Easy way to remember:

- REQUIRED and SUPPORTS join existing transactions.

- REQUIRES_NEW and NESTED always create a new or nested transaction.

- MANDATORY and NEVER enforce presence or absence of a transaction.

- NOT_SUPPORTED ignores transactions completely.

Example:

```
import org.springframework.transaction.annotation.Transactional;
import org.springframework.stereotype.Service;

@Service
public class MyService {
  @Transactional(propagation = Propagation.REQUIRED)
                  // Change propagation type as needed
  public void doSomething() {
    // Business logic here
  }
}
```

JAVA BACKEND DEVELOPER

44.

What improvements are required for the Map in Java-8?

MODEL ANSWER

Improvements required for the Map in Java-8:

Java 8 introduced several improvements to the Map interface to make it easier and more powerful to use:

1: **Default Methods**: New helpful methods like forEach(), getOrDefault(), putIfAbsent(), remove(key, value), replace(), replaceAll(), and compute() methods were added.

2: **Lambda Support**: You can use lambda expressions with Map methods, making code shorter and clearer.

3: **Stream API Integration**: You can convert a Map to a stream for easy filtering, mapping, and processing.

4: **Easier Value Handling**: Methods like getOrDefault() help avoid null checks and make code safer.

5: **Bulk Operations**: Methods like forEach() and replaceAll() let you perform actions on all entries easily.

In short:

Java 8 made Map easier, safer, and more flexible by adding new methods and supporting functional programming.

More improvements that could be made to Map in Java

Immutable Maps: Make it easier to create unmodifiable or read-only maps.

Better Null Handling: Add options to handle null keys and values more safely.

Concurrent Modifications: Improve support for safe updates in multi-threaded environments.

Ordering: Provide built-in ways to keep entries sorted or ordered by insertion.

Memory Efficiency: Reduce memory usage for small maps or maps with few entries.

Serialization: Simplify the process of saving and loading maps to and from files.

Future improvements could focus on safety, performance, easier usage, and better support for modern programming needs.

JAVA BACKEND DEVELOPER

45.

What are the different scope of Beans in Spring?

MODEL ANSWER

Different Scopes of Beans in Spring:

1: **Singleton**

Only one instance of the bean is created for the entire Spring container. (Default scope)

```
@Scope("singleton")
//Only one bean instance for the entire Spring container.
```

2: **Prototype**

A new instance is created every time the bean is requested.

```
@Scope("prototype")
//A new bean instance each time it is requested.
```

3: **Request**

A new bean is created for each HTTP request. (Used in web applications)

```
@Scope("request")
//A new bean instance for every HTTP request (web apps).
```

4: **Session**

A new bean is created for each HTTP session. (Used in web applications)

```
@Scope("session")
//A new bean instance for every HTTP session (web apps).
```

5: **Application**

One bean is created for the entire web application.

```
@Scope("application")
//One bean instance for the entire web application.
```

6: **Websocket**

One bean is created for each WebSocket connection.

```
@Scope("websocket")
//A new bean instance for every WebSocket connection.
```

Example usage in code:

```
@Component
@Scope("singleton") // Use "prototype", "request", etc. as needed
public class MyBean { }
```

Tip to remember:

- Singleton = One for all

- Prototype = One per request

- Request/Session/Application/Websocket = Used in web apps, based on the lifecycle (request, session, app, or websocket)

JAVA BACKEND DEVELOPER

46.

What is composite key in JPA?

MODEL ANSWER

A **composite key** in JPA is a primary key made up of more than one column (field). Instead of using a single field to uniquely identify each row in a table, a combination of two or more fields is used together as the unique identifier. In JPA, you can create a composite key by using either `@Embeddable` and `@EmbeddedId` or by using `@IdClass`. This is useful when no single field is unique by itself, but the combination is unique.

Example:

Suppose you have an OrderItem entity where each order can have multiple items, and each item can appear in multiple orders. The combination of orderId and itemId uniquely identifies each record.

1: **Create the Composite Key Class**

```
import java.io.Serializable;
import javax.persistence.Embeddable;

@Embeddable
public class OrderItemId implements Serializable {
    private Long orderId;
    private Long itemId;

    // Constructors
    public OrderItemId() {}
    public OrderItemId(Long orderId, Long itemId) {
        this.orderId = orderId;
        this.itemId = itemId;
    }

    // Getters and Setters
    public Long getOrderId() { return orderId; }
    public void setOrderId(Long orderId) { this.orderId = orderId; }

    public Long getItemId() { return itemId; }
    public void setItemId(Long itemId) { this.itemId = itemId; }

    // equals() and hashCode() methods
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        OrderItemId that = (OrderItemId) o;
        return orderId.equals(that.orderId)
                            && itemId.equals(that.itemId);
    }

    @Override
    public int hashCode() {
        return java.util.Objects.hash(orderId, itemId);
    }
}
```

2: **Create the Entity Class Using Composite Key**

```
import javax.persistence.*;

@Entity
public class OrderItem {
    @EmbeddedId
    private OrderItemId id;

    private int quantity;

    // Constructors
    public OrderItem() {}
    public OrderItem(OrderItemId id, int quantity) {
        this.id = id;
        this.quantity = quantity;
    }

    // Getters and Setters
    public OrderItemId getId() { return id; }
    public void setId(OrderItemId id) { this.id = id; }

    public int getQuantity() { return quantity; }
    public void setQuantity(int quantity) { this.quantity = quantity; }
}
```

How it works

- The OrderItemId class is used as a composite key (primary key made of orderId and itemId).

- The OrderItem entity uses @EmbeddedId to include the composite key.

- Each OrderItem is uniquely identified by the combination of orderId and itemId.

Summary

Here, OrderItemId (with orderId and itemId) acts as the composite key for the OrderItem entity. This means each row is uniquely identified by the combination of orderId and itemId.

JAVA FULLSTACK DEVELOPER (REACTJS)

47.

What are the ORM frameworks which you have worked on till now?

MODEL ANSWER

I have worked with two ORM frameworks: **Hibernate** and **JPA**.

Hibernate:

Hibernate is a popular ORM framework for Java. It lets us map Java classes to database tables and manage data easily.

Entity Example

```
@Entity
public class Employee {
    @Id
    private int id;
    private String name;
}
```

Java Configuration Example

```
Configuration cfg = new Configuration();
cfg.configure(); // or skip if not using XML
cfg.addAnnotatedClass(Employee.class);
cfg.setProperty("hibernate.connection.url",
                        "jdbc:mysql://localhost:3306/testdb");
cfg.setProperty("hibernate.connection.username", "root");
cfg.setProperty("hibernate.connection.password", "password");
cfg.setProperty("hibernate.dialect",
                        "org.hibernate.dialect.MySQLDialect");

//SessionFactory configuration
SessionFactory sessionFactory =
                    cfg.buildSessionFactory();
```

JPA (Java Persistence API):

JPA is a standard specification for ORM in Java. Hibernate implements JPA. With JPA, we use annotations to define how Java objects relate to database tables.

Entity Example

```
@Entity
public class Employee {
    @Id
    private int id;
    private String name;
}
```

Java Configuration Example

```
Map<String, Object> props = new HashMap<>();
props.put("javax.persistence.jdbc.url",
                          "jdbc:mysql://localhost:3306/testdb");
props.put("javax.persistence.jdbc.user", "root");
props.put("javax.persistence.jdbc.password", "password");
props.put("javax.persistence.jdbc.driver",
                          "com.mysql.cj.jdbc.Driver");

//EntityManagerFactory configuration
EntityManagerFactory emf =
         Persistence.createEntityManagerFactory("myUnit", props);
```

Summary:

1: Hibernate and JPA help me work with databases using Java objects, making data operations simpler and avoiding complex SQL queries.

2: Hibernate and JPA allow me to map Java classes to database tables. I use Java-based configuration to set up connections and manage entities, making data access simple and efficient.

JAVA FULLSTACK DEVELOPER (REACTJS)

48.

What are the databases which you have worked on till now?

MODEL ANSWER

1: **MySQL**

MySQL is an open-source database that is widely used for websites and applications. It’s known for being fast, reliable, and easy to set up.

2: **Oracle**

Oracle Database is a powerful and secure database used by many big organizations. It supports large-scale applications and offers advanced features for managing data.

3: **SQL Server**

SQL Server is developed by Microsoft. It’s commonly used in business environments and integrates well with other Microsoft products.

4: **MongoDB**

MongoDB is a NoSQL database, which means it stores data in a flexible, document-based format (like JSON). It’s great for handling large volumes of data that don’t always fit into tables.

5: **PostgreSQL**

PostgreSQL is an open-source database known for its advanced features and flexibility. It supports complex queries and is highly reliable for both small and large projects.

In summary, MySQL, Oracle, SQL Server, and PostgreSQL are traditional databases that use tables (SQL), while MongoDB is a modern database that uses documents (NoSQL).

JAVA FULLSTACK DEVELOPER (REACTJS)

49.

What is package.json file in ReactJS application? What is the use of it exactly?

MODEL ANSWER

The package.json file in a ReactJS application is like the main information file for your project. It tells you (and the computer) important details such as:

1: The name and version of your app

2: Which libraries or packages your app needs to work (called dependencies)

3: Scripts to run tasks (like starting the app or running tests)

4: Other settings and metadata

In simple words:

package.json helps manage everything your React app needs. It makes it easy to install, update, and run your project. Without it, your app wouldn’t know what it needs or how to work properly.

JAVA FULLSTACK DEVELOPER (REACTJS)

50.

Have you ever worked on Authentication and Authorization phase of any application development? Can you explain these processes in detail?

MODEL ANSWER

Authentication is about confirming “Who are you?”

It checks if someone is really who they say they are.

Example: When you log in to an app with your username and password, the system checks your identity.

(Other ways: PIN, OTP, fingerprint, or face recognition.)

Authorization is about “What are you allowed to do?”

It decides what actions or information you can access after you are authenticated.

Example: After you log in, you might be able to view your profile, but only admins can change system settings.

In summary:

- Authentication = Proving your identity

- Authorization = Checking your permissions

Easy way to remember:

- First, the system asks “Who are you?” (Authentication)

- Then, it asks “What can you do?” (Authorization)

Both steps are important for keeping applications secure and ensuring users only access what they’re allowed to.

JAVA FULLSTACK DEVELOPER (REACTJS)

51.

What is the use of API gateways?

MODEL ANSWER

API gateways act like a single entry point for all client requests to backend services. They help manage, secure, and route requests efficiently. In simple words, an API gateway is like a smart receptionist that receives all calls (requests), checks them, and then connects them to the right department (service) inside a company. This makes things faster, safer, and easier to manage.

Example:

When you use a mobile banking app, your phone sends requests to check your balance, view transactions, or transfer money. Instead of sending these requests directly to different services, the app sends them to the API gateway. The gateway then forwards each request to the correct service, makes sure the request is secure, and sends the response back to your app. This keeps everything organized and safe.

JAVA FULLSTACK DEVELOPER (REACTJS)

52.

How does one microservice communicate with another microservice?

MODEL ANSWER

Microservices communicate with each other mainly through APIs, using simple messages over the network. The most common way is by sending HTTP requests (like how websites talk to each other). One microservice sends a request, and the other responds. Sometimes, they also use messaging systems (like a mailbox) to send messages back and forth. This way, each microservice can work independently but still share information when needed.

Example:

Imagine an online shopping app.

1: The "Order Service" needs to check if items are available before placing an order.

2: It sends an HTTP request to the "Inventory Service" asking, “Do you have item X in stock?”

3: The "Inventory Service" replies with “Yes” or “No.”

4: Based on the answer, the "Order Service" decides whether to complete the order.

This is how one microservice talks to another in real time!

JAVA FULLSTACK DEVELOPER (REACTJS)

53.

What is @Qualifier annotation in SpringBoot?

MODEL ANSWER

The @Qualifier annotation in Spring Boot is used to tell Spring exactly which bean to use when there are multiple beans of the same type.

Think of it like this:

If you have two beans of the same type (for example, two different implementations of an interface), Spring needs to know which one you want to use. By adding @Qualifier ("beanName") next to your dependency, you are specifying the exact bean you want.

Example:

```
@Autowired
@Qualifier("myBean")
private MyService myService;
```

This tells Spring to inject the bean named myBean into myService.

In short:

@Qualifier helps avoid confusion when there are multiple beans of the same type by specifying which one to use.

Here’s a complete and easy-to-understand example of using @Qualifier in Spring Boot

Suppose you have an interface called Animal and two implementations: Dog and Cat. You want to tell Spring which one to inject.

Step 1: Create the interface

```
public interface Animal {
  void makeSound();
}
```

Step 2: Create two implementations

```
import org.springframework.stereotype.Component;

@Component("dog")
public class Dog implements Animal {
  public void makeSound() {
    System.out.println("Woof!");
  }
}

@Component("cat")
public class Cat implements Animal {
  public void makeSound() {
    System.out.println("Meow!");
  }
}
```

Step 3: Use @Qualifier to specify which bean to inject

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;

@Service
public class AnimalService {

  @Autowired
  @Qualifier("dog") // Change to "cat" if you want the Cat bean
  private Animal animal;

  public void play() {
    animal.makeSound();
  }
}
```

Step 4: Run the service

```
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.beans.factory.annotation.Autowired;

@SpringBootApplication
public class DemoApplication implements CommandLineRunner {

  @Autowired
  private AnimalService animalService;

  public static void main(String[] args) {
    SpringApplication.run(DemoApplication.class, args);
  }

  @Override
  public void run(String... args) throws Exception {
    animalService.play(); // Output: Woof!
  }
}
```

Summary:

1: @Qualifier("dog") tells Spring to inject the Dog bean.

2: If you want the Cat bean, just change it to @Qualifier("cat").

This is how you use @Qualifier to choose between multiple beans of the same type!

JAVA FULLSTACK DEVELOPER (REACTJS)

54.

What are the benefits of using SpringBoot in your project?

MODEL ANSWER

Benefits of Using SpringBoot in Your Project:

1: **Easy to Start** - SpringBoot makes it simple to create new projects quickly.

2: **Less Code** - You write less code because SpringBoot handles a lot of setup for you.

3: **Automatic Setup** - It configures things automatically, so you don’t need to worry about complex settings.

4: **Built-in Tools** - Comes with useful tools for testing, monitoring, and deploying your application.

5: **Community Support** - Many people use SpringBoot, so it’s easy to find help and resources.

6: **Flexible** - Works well for small and large projects, and you can customize it as needed.

JAVA FULLSTACK DEVELOPER (REACTJS)

55.

What are the difference between ArrayList and HashSet?

MODEL ANSWER

Difference between ArrayList and HashSet:

1: **Duplicates**

ArrayList allows duplicate elements.

HashSet does NOT allow duplicate elements.

2: **Order**

ArrayList maintains the order in which you add elements (insertion order).

HashSet does NOT guarantee any order.

3: **Performance**

ArrayList is better if you need to access elements by their index (like getting the 3rd item).

HashSet is faster for checking if an element exists.

In short:

- Use ArrayList when you want an ordered list that can have duplicates.

- Use HashSet when you want a collection with unique items and don’t care about the order.

JAVA FULLSTACK DEVELOPER (REACTJS)

56.

How does HashMap internally works? Explain about this in detail.

MODEL ANSWER

A **HashMap** is a data structure that stores data in key-value pairs and allows fast access to values using their keys. Here's how it works in a simple way:

1: **Hashing the Key**

When you put a key-value pair into the HashMap, it uses a special function called a hash function to convert the key into a number (called a hash code).

2: **Index Calculation**

The hash code is used to find an index in an internal array (called a bucket array). This index tells HashMap where to store the value.

3: **Storing the Entry**

The value (along with the key) is stored at the calculated index in the array.

4: **Handling Collisions**

Sometimes, two different keys can get the same index (this is called a collision). When this happens, HashMap stores both entries at the same index using a linked list or a tree structure.

5: **Retrieving Values**

When you ask for a value using a key, HashMap hashes the key again, finds the correct index, and looks for the key in that spot. If it finds the key, it returns the value.

6: **Efficiency**

Because it uses the hash function and direct indexing, HashMap operations (like get and put) are usually very fast (close to constant time).

In summary:

1: HashMap uses a hash function to quickly find where to store or find data, making it very efficient for looking up values by keys.

2: Here's a step-by-step explanation of how HashMap works, and I can visualize this as a diagram for you.

Text Explanation

-> Key is given to HashMap.

-> HashMap calculates the hash code for the key.

-> Hash code is used to find the index in the internal array (bucket).

-> The key-value pair is stored at that index.

-> If two keys have the same index (collision), they are stored together using a linked list or tree.

-> When searching for a value, HashMap hashes the key, finds the index, and retrieves the value.

JAVA FULLSTACK DEVELOPER (REACTJS)

57.

What is the exception handeling in Java and what are the major exceptions related to Java?

MODEL ANSWER

Exception handling in Java is a way to manage errors or unexpected situations that happen during the execution of a program. It helps prevent the program from crashing and allows you to handle problems gracefully.

How it works:

1: Java uses try, catch, and finally blocks.

2: Code that might cause an error goes inside the try block.

3: If an error (exception) occurs, it is "caught" by the catch block, where you can write code to handle the problem.

4: The finally block (optional) runs code no matter what, even if an exception happened.

Major Exceptions in Java (Easy to Remember):

1: **NullPointerException** – When you try to use an object that is null.

2: **ArrayIndexOutOfBoundsException** – When you try to access an invalid index in an array.

3: **ArithmeticException** – When you do illegal math, like dividing by zero.

4: **ClassNotFoundException** – When Java can’t find a class you are trying to use.

5: **NumberFormatException** – When you try to convert a string to a number, but the string isn’t a valid number.

6: **FileNotFoundException** – When a file you are trying to read doesn’t exist.

In summary:

Exception handling in Java helps you deal with errors without stopping the whole program. Just remember: use try-catch blocks, and know the common exceptions so you can handle them easily!

JAVA FULLSTACK DEVELOPER (REACTJS)

58.

Can you explain about Java-8 features?

MODEL ANSWER

Java 8 Key Features (Easy to Remember):

1: **Lambda Expressions**

Lets you write code in a cleaner and shorter way, especially for passing behavior as a method argument.

```
Runnable r = () -> System.out.println("Hello from Lambda!");
```

2: **Functional Interfaces**

Interfaces with only one abstract method. Used mainly with lambda expressions.

```
@FunctionalInterface interface MyFunc { void doSomething(); }
```

3: **Stream API**

Helps process collections of objects easily (like filtering, sorting, and mapping data).

```
List<String> names = list.stream().filter(s -> s.startsWith("A"))
                                        .collect(Collectors.toList());
```

4: **Default & Static Methods in Interfaces**

You can now add methods with a body inside interfaces.

```
default void print() { System.out.println("Default method"); }
```

5: **Date and Time API**

Improved classes to handle date and time (like LocalDate, LocalTime, LocalDateTime).

```
LocalDate today = LocalDate.now();
```

6: **Optional Class**

Helps avoid NullPointerException by representing optional values.

```
Optional<String> name = Optional.ofNullable(getName());
```

7: **Nashorn JavaScript Engine**

Allows you to run JavaScript code on the Java platform.

```
ScriptEngine engine = new ScriptEngineManager()
                                    .getEngineByName("nashorn");
```

In short:

Java 8 made Java more powerful and easier to use with features like lambdas, streams, better date/time handling, and more!

JAVA FULLSTACK DEVELOPER (REACTJS)

59.

What are props in ReactJS?

MODEL ANSWER

Props in ReactJS are like special containers used to pass data from one component to another.

1: Think of props as "arguments" you send to a function, but here you send them to a component.

2: They help make components reusable by allowing you to give different data to each one.

3: Props are read-only, which means a component can use them, but cannot change them.

In short:

Props are a way to send information from parent to child components in React.

Example 1: Passing a Name as a Prop

```
function Welcome(props) {
 return <h1>Hello, {props.name}!</h1>;
}

// Using the component and passing "John" as a prop
<Welcome name="John" />
```

Example 2: Passing Multiple Props

```
function UserInfo(props) {
 return <p>{props.name} is {props.age} years old.</p>;
}

// Using the component and passing two props
<UserInfo name="Alice" age={25} />
```

Example 3: Passing a Function as a Prop

```
function Button(props) {
 return <button onClick={props.onClick}>Click me</button>;
}

function handleClick() {
 alert("Button was clicked!");
}

// Using the component and passing a function as a prop
<Button onClick={handleClick} />
```

Summary:

Props help you send data (like text, numbers, or even functions) from one component to another, making your components flexible and reusable.

JAVA FULLSTACK DEVELOPER (REACTJS)

60.

What is the difference between function and class in ReactJS?

MODEL ANSWER

The difference between Function and Class in ReactJS:

1: **Function Components:**

- Are just JavaScript functions.

- Easier and shorter to write.

- Use React Hooks (like `useState`, `useEffect`) to manage state and side effects.

- Preferred in modern React.

2: **Class Components:**

- Use ES6 classes.

- More code and a bit complex.

- Manage state using `this.state` and lifecycle methods like `componentDidMount`.

- Were used before Hooks were introduced.

Function Component Example

```
function Welcome() {
  return <h1>Hello from Function Component!</h1>;
}
```

With State using Hook:

```
import React, { useState } from 'react';
function Counter() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increase</button>
    </div>;
}
```

Class Component Example

```
import React, { Component } from 'react';
class Welcome extends Component {
  render() {
    return <h1>Hello from Class Component!</h1>;
  }
}
```

With State:

```
import React, { Component } from 'react';
class Counter extends Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }
  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={() =>
                  this.setState({ count: this.state.count + 1 })}>
          Increase
        </button>
      </div>;
  }
}
```

In summary:

1: Function components are simpler and more modern, while class components are older and more complex. Most new React code uses function components.

2: Function components use simple functions and Hooks for state.

3: Class components use classes, this.state, and lifecycle methods.

JAVA FULLSTACK DEVELOPER (REACTJS)

61.

What is the difference between undefined and NULL?

MODEL ANSWER

Difference between undefined and NULL:

1: **undefined** means a variable has been declared, but no value has been given to it yet.

Think: "I don’t know what this is yet."

2: **NULL** means a variable has been given a value, and that value is "nothing" or "empty."

Think: "I know this is nothing."

Summary:

1: undefined = not assigned any value

2: NULL = assigned to be empty on purpose

JAVA FULLSTACK DEVELOPER (REACTJS)

62.

Can you explain the difference between undefined and undeclared variables in JavaScript, when it comes undefined and when it comes undeclared?

MODEL ANSWER

Undefined vs. Undeclared Variables in JavaScript (Easy Explanation):

Undefined:

A variable is undefined when you declare it, but don’t give it a value.

Example:

```
let x;
console.log(x); // Output: undefined
```

Think: "I made a box, but didn’t put anything inside."

Undeclared:

A variable is undeclared when you try to use it, but never declared it at all.

Example:

```
console.log(y); // Error: y is not defined
```

Think: "I’m looking for a box that doesn’t exist."

Summary:

1: Undefined = declared, but not assigned a value.

2: Undeclared = never declared, using it causes an error.

Tip to Remember:

1: Undefined: Box exists, but empty.

2: Undeclared: Box doesn’t exist.

JAVA FULLSTACK DEVELOPER (REACTJS)

63.

What is the callback function in JavaScript? Where did we use this callback function actually? Can we use this callback function from on class to another class in the JavaScript, here class is nothing but different AJAX calls, means can we use callback function for certain AJAX call and the after getting it's response, can we again use the same callback function for the another AJAX call which we are doing from the success response of the first AJAX call?

MODEL ANSWER

A callback function in JavaScript is a function that is passed as an argument to another function and is executed after some operation is completed.

Where do we use callback functions?

Callback functions are commonly used in asynchronous operations like AJAX calls, setTimeout, event listeners, or when you want to run some code only after another function finishes its task.

Can we use the same callback function across different AJAX calls or classes?

Yes, you can use the same callback function for multiple AJAX calls. For example, you can call an AJAX request, and in its success handler (callback), you can make another AJAX call and reuse the same callback function again. In JavaScript, functions are treated as first-class citizens, so you can pass them around and reuse them wherever needed.

Example:

```
function handleResponse(data) {
    console.log("Received data:", data);
    // You can process the data or trigger another action here
}


// First AJAX call
$.ajax({
    url: 'api/first',
    success: function(response) {
        handleResponse(response);
        // Second AJAX call inside the success of the first
        $.ajax({
            url: 'api/second',
            success: handleResponse // Reusing the same callback
        });
    }
});
```

Summary:

1: Callback = function passed to another function to run later.

2: Used in AJAX, timers, and event handling.

3: You can reuse the same callback for multiple AJAX calls, even across different parts of your code.

JAVA FULLSTACK DEVELOPER (REACTJS)

64.

Can you tell me some built-in methods in JavaScript?

MODEL ANSWER

Some Common Built-in Methods in JavaScript (Easy to Remember):

1: alert() – Shows a popup message.

2: console.log() – Prints messages to the browser console.

3: toUpperCase() – Converts text to uppercase.

4: toLowerCase() – Converts text to lowercase.

5: push() – Adds an item to the end of an array.

6: pop() – Removes the last item from an array.

7: slice() – Returns a part of a string or array.

8: length – Gives the number of items in a string or array.

9: Math.random() – Gives a random number between 0 and 1.

10: parseInt() – Converts a string to an integer number.

These methods help you work with text, numbers, and arrays easily in JavaScript!

JAVA FULLSTACK DEVELOPER (REACTJS)

65.

What are the different class loaders in JVM?

MODEL ANSWER

The different class loaders in JVM are:

1: **Bootstrap Class Loader**

- Loads core Java classes (like java.lang.*) from the JDK.

- It is the parent of all other class loaders.

2: **Extension (Platform) Class Loader**

- Loads classes from the ext (extension) directory of the JDK.

- It is a child of the Bootstrap Class Loader.

3: **Application (System) Class Loader**

- Loads classes from the application's classpath (the location where your code and libraries are).

- It is a child of the Extension Class Loader.

Tip to remember:

1: Think of class loaders as a chain:

Bootstrap # Extension # Application

2: Each one loads classes from a specific place, starting from the core Java classes up to your own application code.

JAVA FULLSTACK DEVELOPER (REACTJS)

66.

What is clustering in Java?

MODEL ANSWER

Clustering in Java is a technique where multiple Java servers (or instances) work together as a group, called a "cluster", to handle requests. This helps improve performance, reliability, and allows the application to keep running even if one server fails. In simple words, clustering lets many servers share the work and back each other up, making Java applications faster and more reliable.

Examples of clustering in Java:

1: **Web servers:** Multiple Tomcat servers running together in a cluster to handle more website visitors and share the load.

2: **Application servers:** Java EE servers like JBoss or WebLogic use clustering to provide high availability and failover support for enterprise applications.

3: **Distributed caching:** Tools like Hazelcast or Apache Ignite use clustering to store and share data across many Java servers, so data is always available even if one server goes down.

4: **Microservices:** In a microservices architecture, Java-based services are often clustered to ensure continuous service and better performance.

These examples show how clustering helps Java applications handle more users, avoid downtime, and work more efficiently.

JAVA FULLSTACK DEVELOPER (REACTJS)

67.

Java is a single threading programming language or multi threading programming language?

MODEL ANSWER

Java is a multi-threading programming language.

This means Java can run several tasks at the same time using threads. So, Java is not limited to doing just one thing at a time (single-threading); it can do many things together (multi-threading). This makes Java powerful and efficient for programs that need to handle multiple actions at once, like games or servers.

Easy to remember:

Java = Multi-threading = Many tasks at the same time

JAVA FULLSTACK DEVELOPER (REACTJS)

68.

Explain the difference between JDK, JRE and JVM?

MODEL ANSWER

1: **JVM (Java Virtual Machine)**

It is like a translator that runs Java programs. It takes the compiled Java code (bytecode) and converts it into machine code so your computer can understand and execute it.

2: **JRE (Java Runtime Environment)**

It is a package that gives you everything you need to run Java programs, including the JVM and some libraries, but it does not have tools for developing Java programs.

3: **JDK (Java Development Kit)**

It is a complete package for Java developers. It includes the JRE (so you can run programs), the JVM, and also tools to write, compile, and debug Java code.

In short:

JVM = Runs Java programs

JRE = Runs Java programs + libraries

JDK = Develops and runs Java programs (JRE + development tools)

JAVA FULLSTACK DEVELOPER (REACTJS)

69.

What is a JIT compiler in Java?

MODEL ANSWER

1: A JIT (Just-In-Time) compiler in Java is a part of the Java Virtual Machine (JVM) that helps make Java programs run faster. When you run a Java program, the JVM first converts the code into an intermediate form called bytecode. The JIT compiler then takes this bytecode and translates it into machine code (native code) just before it is needed, while the program is running. This makes the program execute more quickly because the machine code runs directly on your computer's processor. In short, the JIT compiler improves the speed and performance of Java applications by converting bytecode to machine code at runtime.

2: The JIT compiler is not strictly mandatory, but it is highly beneficial. Java programs can run without the JIT compiler, but they will be slower because the JVM would interpret the bytecode line by line instead of converting it to faster machine code. The JIT compiler is used to boost performance, so most Java environments include it by default.

In summary, while Java can work without JIT, having it makes programs run much faster and more efficiently.

JAVA FULLSTACK DEVELOPER (REACTJS)

70.

What are the most typical memory leak issues in Java and how do you handle it?

MODEL ANSWER

Most Typical Memory Leak Issues in Java & How to Handle Them

1: **Holding Unused Objects:**

Keeping references to objects you no longer need prevents garbage collection.

Solution: Set unused object references to null or remove them from collections.

Example:

```
// Release reference when done
myObject = null;
```

2: **Static Fields:**

Objects stored in static fields live as long as the application, causing leaks if not cleared.

Solution: Clear static fields when they are no longer needed.

Example:

```
// Clear static collection when not needed
MyClass.staticList.clear();
```

3: **Listeners and Callbacks:**

Forgetting to remove listeners or callbacks keeps objects alive.

Solution: Always unregister listeners and callbacks when done.

Example:

```
// Unregister listener when done
button.removeActionListener(myListener);
```

4: **Large Collections:**

Adding lots of data to collections (like Lists, Maps) without removing old entries.

Solution: Remove unused items from collections regularly.

Example:

```
// Remove unused entry from collection
myMap.remove(oldKey);
```

5: **Inner Classes:**

Non-static inner classes hold references to their outer class, which can cause leaks.

Solution: Use static inner classes or make sure to clean up references.

Example:

```
// Use static inner class to avoid holding outer reference
static class MyInnerClass { }
```

How to Handle Memory Leaks:

1: Use profiling tools (like VisualVM or Eclipse Memory Analyzer) to spot leaks.

2: Follow good coding practices: clear references, unregister listeners, clean up collections.

In summary:

Avoid keeping unused objects, clear static fields, unregister listeners, clean up collections, and use profiling tools to detect leaks.

JAVA FULLSTACK DEVELOPER (REACTJS)

71.

Do we have a return type for Abstract class and for Interface?

MODEL ANSWER

No, abstract classes and interfaces themselves do not have a "return type."

1: **Abstract Class:** The methods inside an abstract class can have return types, but the abstract class itself does not.

2: **Interface:** Similarly, the methods inside an interface can have return types, but the interface itself does not.

Remember:

Return type is for methods/functions, not for classes or interfaces themselves.

JAVA FULLSTACK DEVELOPER (REACTJS)

72.

What is the difference between Abstract class and Interface class in Java?

MODEL ANSWER

Difference between Abstract Class and Interface in Java:

Abstract Class:

1: Can have both abstract (without body) and concrete (with body) methods.

2: Can have variables with any access modifier (private, protected, etc.).

3: Can provide some code (implementation) that subclasses can reuse.

4: A class can extend only one abstract class (single inheritance).

Interface:

1: All methods are abstract by default (until Java 8, after which default and static methods are allowed).

2: All variables are public, static, and final (constants) by default.

3: Cannot provide code for methods (except default/static methods).

4: A class can implement multiple interfaces (multiple inheritance).

In short:

1: Use an abstract class when you want to share some code among related classes.

2: Use an interface when you want to define a contract that many classes can follow, even if they are not related.

JAVA FULLSTACK DEVELOPER (REACTJS)

73.

How do we configure caching in SpringBoot applications? What are the annotations used for caching?

MODEL ANSWER

How to Configure Caching in Spring Boot Applications

1: **Enable Caching:**

Add @EnableCaching annotation in your main application class.

2: **Add Cache Dependency:**

Include a cache provider in your pom.xml (like EhCache, Redis, or use default).

3: **Use Caching Annotations in Your Code:**

A: @Cacheable: Marks a method whose result should be cached.

B: @CachePut: Updates the cache with the new value.

C: @CacheEvict: Removes data from the cache.

```
@EnableCaching
@SpringBootApplication
public class MyApp { ... }

@Cacheable("users")
public User getUserById(Long id) { ... }

@CachePut("users")
public User updateUser(User user) { ... }

@CacheEvict("users")
public void deleteUser(Long id) { ... }
```

Summary:

1: Use @EnableCaching to turn on caching.

2: Use @Cacheable, @CachePut, and @CacheEvict to manage cache in methods.

JAVA FULLSTACK DEVELOPER (REACTJS)

74.

How are you handling authentication and authorization in SpringBoot applications?

MODEL ANSWER

In Spring Boot applications, authentication and authorization are usually handled using Spring Security.

1: **Authentication** means checking who the user is (like asking for username and password).

2: **Authorization** means checking what the user is allowed to do (like checking their roles or permissions).

How it works:

1: When a user tries to log in, Spring Security checks their credentials (authentication).

2: If the credentials are correct, Spring Security gives them access.

3: For each request, Spring Security checks if the user has permission to access that resource (authorization).

Example:

1: You can use in-memory users, a database, or external providers (like OAuth2) for authentication.

2: You can set up rules to allow or block access to certain URLs based on user roles.

In short:

Spring Boot uses Spring Security to make sure only the right people can log in and only see or do what they are allowed to.

JAVA FULLSTACK DEVELOPER (REACTJS)

75.

How can we enable the CrossOrigin resource sharing in SpringBoot? Which builtin method overrides the CrossOrigin resource sharing feature and which annotation mostly we used in this case?

MODEL ANSWER

To enable CORS in Spring Boot, follow these simple steps:

1: Use the @CrossOrigin Annotation:

The easiest way to enable CORS for a specific controller or method is to add the @CrossOrigin annotation.

Example:

```
@CrossOrigin
@GetMapping("/example")
public String example() {
  return "Hello";
}
```

2: Override the addCorsMappings Method:

For global CORS configuration, override the addCorsMappings method in a configuration class that implements WebMvcConfigurer.

Example:

```
@Configuration
public class WebConfig implements WebMvcConfigurer {
  @Override
  public void addCorsMappings(CorsRegistry registry) {
    registry.addMapping("/**").allowedOrigins("*");
  }
}
```

Summary (Easy to Remember):

1: Use @CrossOrigin annotation for quick enablement.

2: Override addCorsMappings for global settings.

3: The most used annotation is @CrossOrigin.

Tip:

Remember: @CrossOrigin = Enable CORS easily in Spring Boot!

JAVA FULLSTACK DEVELOPER (REACTJS)

76.

Can you explain the difference between @Component, @Service and @Repository annotations in SpringBoot?

MODEL ANSWER

@Component, @Service, and @Repository are all used in Spring Boot to let Spring manage classes for us.

1: @Component:

For any general class you want Spring to manage.

2: @Service:

For classes that handle business logic (like calculations or processing).

3: @Repository:

For classes that deal with database operations (like saving or fetching data).

Summary:

1: @Component = any class

2: @Service = business logic

3: @Repository = database access