TCS Interview on 19-11-24

1.different == .equals operation in java

Ans:- ==: Checks if two references point to the same object.

.equals(): Checks if two objects are logically equal (based on their content if overridden properly).

2. Different types of Loops in Java

Ans:- **for loop**: Used when the number of iterations is known.

for (initialization; condition; update) { // code block to be executed }

for (int i = 0; i < 5; i++) { System.out.println(i); }

**while loop**: Used when the condition is checked before entering the loop.

while (condition) { // code block to be executed }

int i = 0;

while (i < 5) {

System.out.println(i);

i++;

}

**do-while loop**: Used when the loop body should execute at least once.

do { // code block to be executed } while (condition);

int i = 0;

do {

System.out.println(i);

i++;

} while (i < 5);

**Enhanced for loop**: Used for iterating over arrays or collections more easily.

for (type element : array) { // code block to be executed }

int[] numbers = {1, 2, 3, 4, 5};

for (int number : numbers) {

System.out.println(number);

}

5.**Explain Base principals of Object oriented programming**

encapsulation :- In Java, encapsulation is the process of bundling data and methods into a single unit,for providing security and hiding the data from other classes

Abstraction :- Hiding the complex implementation details and showing only the necessary features or functionalities.

Benefit: Reduces complexity for users by focusing on essential aspects.

We can Achieve this by Two Ways By Abstract Class and Interface

A class which is declared by abstract keyword is known as abstract class

Abstract class Is restricted class that cannot to be used to create object (to access it it must be inherited from another class)

Abstract method : can only be used in abstract class and It doesn’t have body the body is provided by Sub Class (Inherited from)

Interface is used to group related methods with empty bodies to access the interface it must be implemented by another class with implement keyword

6.what is the tool use used in Java -- Gitlab, Tortise Svn

7.deployment what u will use –

**Apache Tomcat**: Java web application as a **WAR** (Web Archive) file and deploy it to the Tomcat server by copying the WAR file into the webapps directory.

Jenkins (CI/CD Automation Tool): Jenkins is a continuous integration/continuous deployment (CI/CD) tool that automates building, testing, and deploying Java projects.

**Docker**: Deploy applications in containers.

**Kubernetes**: Orchestrates containerised Java applications.

8. explain Features of Java 8 –

1. What is Lambda Expression

The Expression through which we can represent an anonymous function Anonymous means unknown Nameless a method who doesn’t have any name or modifier Syntax () -> {}

Lambda Expression can be applicable only to Functional interface the method which is written inside the Functional interface for that method only we can write the lambda expression

2. What is the Functional Interface

The interface that contains only one abstract method but can have multiple default and static methods is called a Functional interface

EG – Runnable -> run() Comparator -> compare() Callable -> call() Comparable -> compareTo()

**Consumer Functional interface**   
The be consumer <T> is an in built functional interface Introduced in Java 8

consumer can be used in all contexts where an object need to be consumed it taken input and some operation is to be performed on the object without returning any result

performs this operation on the given argument

@param I the input argument roud accept (It)

**Predicate functional interface**

This functional interface used for conditional check Where you think We can use this True False returning function in day to day programming We choose predicates

@param t to the input argument

Void accept (T t )- Type generis the argument can be any types it can be the object int string

Return type Boolean we can filter the date also

**Supplier Functioned interface**

Supplier can be stat in all contexts wh

ere there is no input but output expected

Get a result Briton a result I get()

What is Stream Api in java

Stream Api is used to process collections of objects

A stream is a sequence of objects that supports various Methods that can be pipelined to produce the desired result

A stream is not a data structure instead It takes input from the collections. Arrays

Stream doesn't change the original data structure they only provide the result as per the pipelined Methods

Why He and stream Functional programming If I have functional Interface then I can represent in Lambda expression code Reduce If you are going to write he Lombda expression then code is going to minize

Bulk operator - If I have Bulk data performing some operation then go for stream Api

Methods Filter -for conditional check forEach :- for iteration

What is Map and FlatMap

Map is Used for data Transformation

It does only mapping

It takes Stram <T> as Input and Return Stream its mapper function produces a single value for each input value hence it is one to one mapping

FlatMap is used for data Transformation and Data Flattering

It does mapping as well as flattering

Its mapper function produces multiple values for each input value is called one-to-many mapping

transformation - [[1,2],[3,4] [5,6] ] = > [1,2,3,4,5,6] converting streams of Stream into Single Stream is called Flattering.

Converting lowercase to Uppercase is called mapping

9. What is an Optional class in Java 8

Ans : The Optional class is a part of the java. util package & is commonly used in streams & other functional programming patterns introduced in Java 8. With the Optional class, you can explicitly express the possibility of a null value, making your code more expressive & readable.

10.what is Microservices how u will you create

Ans: Microservices are a software development approach that breaks down an application into a collection of independent services that communicate with each other:

11. what are the starters u imported while creating the Spring Boot Project

Ans :- spring-boot-starter-web : To create web applications, including RESTful APIs

spring-boot-starter-data-jpa : For database integration using Spring Data JPA with Hibernate as the default JPA implementation.

spring-boot-devtools : To improve the development experience with features like hot swapping and auto-restart.

spring-boot-starter-security : Adds security features, including authentication and authorization.

spring-boot-starter-test: Provides dependencies for testing Spring Boot applications.

spring-boot-starter-thymeleaf : For rendering HTML views using Thymeleaf templating engine

spring-boot-starter-jdbc : To support traditional JDBC for database access.

spring-boot-starter-mail

12. why we use Spring

Ans :-

Spring is widely considered to be a secure, low-cost and flexible framework that improves coding efficiency and reduces overall application development time through efficient use of system resources. Spring removes tedious configuration work so that developers can focus on writing business logic.

Speed up development.Reduce boilerplate code,Embedded servers,Simplified dependency management

13. what is Role of @Springbootapplication Annotation

Ans :- The @SpringBootApplication annotation is a central part of Spring Boot applications. It is a convenience annotation that combines several other important annotations to enable auto-configuration and component scanning in a Spring Boot project. It is typically placed on the main class of a Spring Boot application to mark it as the entry point.

The @SpringBootApplication annotation is essentially a combination of the following three annotations:

@EnableAutoConfiguration: This annotation tells Spring Boot to automatically configure the application based on the dependencies present in the classpath. It enables auto-configuration

@ComponentScan: his annotation enables component scanning. It tells Spring to scan the current package and its sub-packages for Spring components (like @Component, @Service, @Repository, and @Controller) and automatically register them as beans in the Spring application context.

@Configuration : This marks the class as a configuration class, which can contain @Bean definitions.

14.how u will connect database to Spring Boot Application

ans : In a Spring Boot application, connecting to a database involves a series of steps that include adding the necessary dependencies, configuring the database connection, creating entity classes, and interacting with the database using Spring Data JPA (or JDBC). Spring Boot simplifies this process with auto-configuration,

15.what is Dependency injection

Ans :- In Spring, Dependency Injection is achieved via the Spring container (also called the IoC container). The container is responsible for instantiating the beans (objects), managing their lifecycle, and injecting their dependencies.

For example, if you have a Car class that depends on an Engine class, Spring can inject an Engine object into the Car class.

which injects dependencies into beans automatically based on annotations like @Autowired

1. how to read Excel file which dependency need to use

To read an Excel file in a Spring Boot or Java application, the most commonly used dependency is **Apache POI**. Apache POI provides libraries for handling Microsoft Office files, including Excel (.xls and .xlsx formats).

2. what is datatype u used to store excel file

**BLOB (Binary Large Object)**: If you're storing the file directly in the database, the file is usually stored as a BLOB (Binary Large Object).

**byte[] (Byte Array)**: When handling file data in Java, you can use a byte[] array to represent the binary contents of the file. You can read and write Excel files as a byte array before saving or retrieving them from the database.

**Database**: Use BLOB. **Java Code**: Use byte[] to handle the file content. **File Manipulation**: Use libraries like **Apache POI** or **JExcelAPI**.

3.What is JPA ?

**JPA** (Java Persistence API) is a specification in Java that defines how to manage relational data in applications using object-relational mapping (ORM). It provides a way to map Java objects (entities) to database tables and simplifies the process of interacting with relational databases in Java applications.

**.save()** is for saving or updating a single entity **.saveAll()** is for saving or updating multiple entities at once.

**@Entity**: Marks a class as an entity (a table in the database).

**@Table**: Specifies the table name for an entity.

**@Id**: Marks the primary key of the entity.

**@GeneratedValue**: Specifies the generation strategy for primary keys.

**@OneToMany**, **@ManyToOne**: Defines relationships between entities.

4.Difference Between @Controller @RestController

**Controller** :-Used for traditional web applications, a controller maps HTTP requests to view names. It's best used in UI-based applications that return a view, such as an HTML page.

**RestController :-** Used for building RESTful web services, a rest controller returns data, such as JSON or XML, rather than a view. It's best used when you want to build APIs that serve data to clients, mobile apps, or other applications.

**Combination of Annotations**: @RestController is a convenience annotation that combines @Controller and @ResponseBody.

Spring Boot Microservices Questions By Sagar Baranwal

1. what are the types of BeanScope and explain

Ans:- In the Spring Framework, there are different types of bean **scopes** available. The scope of a bean mainly defines the lifecycle and visibility of a given bean within the context it is being used.

1. **Singleton** - One single bean instance per Spring IoC container.
2. **Prototype** - A new bean instance each time when requested.
3. **Request** - A single bean per HTTP/HTTPS(Web) request.\*
4. **Session** - A single bean instance per HTTP/HTTPS session.\*
5. **Global Session** - A single bean instance per HTTP/HTTPS global session.\*
6. **Application** - A single bean per lifecycle of a ServletContext.\*

2. what is the use of @Qualifier and @Primary

Ans:- In Spring, both @Qualifier and @Primary annotations help resolve the issue of **ambiguity** when multiple beans of the same type are available in the Spring context, and Spring doesn’t know which one to inject.

* **@Qualifier**: Use when you want to **specify** exactly which bean to inject by name.
* **@Primary**: Use when you want a bean to be the **default** choice when no specific qualifier is provided.

Both annotations are helpful in managing dependencies when your application has multiple beans of the same type, but you need to decide which one to use in different scenarios.

3. how to implement exception handling in SpringBoot

Ans:- 1.**Using @ControllerAdvice (Global Exception Handling)**

The @ControllerAdvice annotation allows you to handle exceptions **globally** across multiple controllers in a centralized way. It is used to create a **global error handling component**.

**How to use:**

* Create a class annotated with @ControllerAdvice.
* Define methods annotated with @ExceptionHandler inside this class to handle specific exceptions globally.



2.using Custom Exception Classes :- To make your exception handling meaningful, you can create custom exception classes. These exceptions can carry custom messages and details relevant to your business logic.

3. **Using @ExceptionHandler (Local Exception Handling)**

You can use the @ExceptionHandler annotation inside a controller to handle specific exceptions for that controller. This method is **local** to the controller where the exception occurs.

**4**. Using ResponseEntityExceptionHandler for Global Exception Handling

Spring provides a built-in ResponseEntityExceptionHandler class that you can extend to handle exceptions in a structured way for RESTful applications. This class provides default handlers for Spring exceptions

5. What are microservices and how do they interact with each other?

Ans:- **What Are Microservices?**

**Microservices** is an architectural style in software development where a large application is broken down into smaller, independent, loosely coupled services. Each microservice is designed to handle a specific business function or process and can be developed, deployed, and maintained independently by different teams. These services communicate with each other through well-defined APIs, typically over network protocols like HTTP, gRPC, or messaging systems.

1.**Loose Coupling**: Microservices are loosely coupled, meaning changes in one service typically do not require changes in other services.

2. **Small and Focused**: Each service focuses on a specific business capability, such as user management, inventory, billing, etc.

3.**Technology Diversity**: Each microservice can use different technologies, programming languages, databases, and frameworks, based on the specific requirements of that service.

4.**Decentralized Data Management**: Each microservice manages its own database, preventing direct data sharing.

**1. Synchronous Communication :** Synchronous communication means that when one service calls another, it waits for a response. This is the traditional request-response model.

**2. Asynchronous Communication** : Asynchronous communication occurs when services communicate without waiting for an immediate response, allowing the calling service to continue its work.

**Example of How Microservices Interact:**

Let’s imagine an **e-commerce** application that uses microservices. Here are some typical microservices:

* **User Service**: Manages user accounts.
* **Order Service**: Handles order placement and tracking.
* **Inventory Service**: Manages product stock.
* **Payment Service**: Processes payments.

Interaction Workflow:

* A user places an order through a **client application**.
* The client sends a request to the **API Gateway**, which forwards the request to the **Order Service**.
* The **Order Service** communicates with the **Inventory Service** to check stock levels and the **Payment Service** to process the payment.
* Once payment is confirmed, the **Order Service** updates the order status.
* The **Notification Service** may send an email to the user confirming the order.

6.Message queue based interaction between microservices.

Ans:- In microservices architecture, one of the most common ways to enable communication between different services is through **message queues**. This method is especially useful for **asynchronous** communication, where services don't need to wait for immediate responses and can continue their work independently after sending or receiving messages.

7.How do microservices transfer data between each other? (Using Rest)

Ans:- In a microservices architecture, REST is a popular method for services to exchange data synchronously. By using HTTP methods, microservices can request, update, and delete resources from other services, allowing them to interact and coordinate their functionalities. Despite some drawbacks like latency and network dependency, REST remains a powerful and widely-used approach for service-to-service communication.

 **GET**: Retrieve data from the server (e.g., getting user details).

 **POST**: Send data to the server (e.g., creating a new order).

 **PUT**: Update existing data on the server (e.g., updating user information).

 **DELETE**: Remove data from the server (e.g., deleting an item).

8. How does microservices interact in cloud based systems.

Ans:- In cloud-based systems, microservices interact through a combination of **synchronous (HTTP/REST, gRPC)** and **asynchronous (message queues, event streams)** communication methods. The cloud provides essential services like **API gateways**, **service discovery**, **load balancing**, and **security mechanisms** to facilitate reliable and scalable communication between services. Additionally, tools like **service meshes** and **serverless** functions help manage complex microservice interactions, ensuring a flexible and resilient architecture.

9. Diff between @bean and @autowired

Ans:- The @Bean annotation is used at the method level to explicitly define a bean in a Spring configuration class (usually annotated with @Configuration). Use @Bean when you want to create and configure beans manually**, typically for third-party library classes or complex beans that require custom initialization.**

The @Autowired annotation is used for **automatic dependency injection**. It tells Spring to automatically resolve and inject the dependent bean into the marked field, constructor, or method. Use @Autowired to automatically wire beans by type without manually fetching them from the Spring context.

10. Why we need to use hibernate

Ans:- Hibernate is a popular **Object-Relational Mapping (ORM)** framework for Java that simplifies database interaction by mapping Java objects to database tables.

**Simplifies Database Interaction**

* Without Hibernate, developers need to write a lot of **boilerplate JDBC code** to handle SQL queries, manage database connections, transactions, and exception handling. Hibernate abstracts these tasks, allowing you to work with Java objects instead of writing complex SQL queries.

**Eliminates Boilerplate Code**

* Hibernate removes the need for **manual SQL** in many cases. For instance, you don’t need to write SQL queries for basic CRUD (Create, Read, Update, Delete) operations. Hibernate provides simple APIs like save(), update(), delete(), find(), etc., making your code cleaner and more maintainable.

**Database Independence**

* Hibernate is **database-agnostic**, meaning you can switch databases (e.g., from MySQL to PostgreSQL) without changing your code, since Hibernate generates database-specific SQL based on dialects. You just need to configure the correct database dialect in your Hibernate configuration.

**Automatic Table Schema Generation**

* Hibernate can automatically generate and manage the schema of your database using annotations in your Java classes. It can create, update, or validate the schema based on your domain models.
* <property name="hibernate.hbm2ddl.auto" value="update"/>

Hibernate provides a built-in **caching mechanism** to improve performance

Hibernate integrates seamlessly with **transaction management** frameworks (like JTA or Spring’s transaction management),

**HQL (Hibernate Query Language)**

* Hibernate offers its own object-oriented query language called **HQL** (Hibernate Query Language), which is similar to SQL but works with Java objects instead of tables. This makes it easier to work with database queries in an object-oriented fashion.

String hql = "FROM Employee WHERE department = :dept";

Query query = session.createQuery(hql);

query.setParameter("dept", "Sales");

List<Employee> results = query.list();

11.Default scope in hibernate

Ans :- In Hibernate, the default scope for most beans (like SessionFactory) is **singleton**. However, when working with a Session, it’s not tied to the singleton scope and is typically short-lived and created per request or transaction.

12. How to resolve cyclic depency injection

Ans:- Cyclic dependency injection occurs when two or more beans depend on each other in such a way that it creates a circular reference, preventing the Spring container from resolving the dependencies. For example, if **BeanA** depends on **BeanB**, and **BeanB** depends on **BeanA**, Spring cannot instantiate either bean because they require each other to be fully created first.

Use @Lazy on one of the dependencies to defer the initialization of that bean until it's needed. 1.Setter Injection 2.Field Injection 3.Constructor Injection 4.with @Lazy Annotation Spring’s 5.ApplicationContextAware Interface

13. Have you worked on rest api

Ans:- Yes, I have worked extensively on REST APIs in various projects, particularly in developing and consuming them using **Spring Boot**.

**Developing RESTful APIs**:

I have designed and implemented RESTful services using **Spring Boot** with proper HTTP methods (GET, POST, PUT, DELETE) adhering to REST principles.

For example, GET for retrieving resources, POST for creating new resources, PUT for updating existing resources, and DELETE for deleting resources.

**Handling Request and Response**:

I've worked with **@RequestMapping**, **@GetMapping**, **@PostMapping**, and other annotations to map HTTP requests to controller methods.

Used **@RequestBody** to handle incoming JSON requests and **@ResponseBody** to return JSON responses.

**Exception Handling**:

Implemented centralized error handling for REST APIs using **@ControllerAdvice** and **@ExceptionHandler** annotations to return meaningful error responses (such as 400 Bad Request or 404 Not Found) to clients.

**Security**:

Integrated **Spring Security** and **OAuth2** for securing REST APIs. I have worked on authentication and authorization using JWT tokens and role-based access control.

**Swagger and API Documentation**:

I have integrated **Swagger** (using the springdoc-openapi library) to generate interactive API documentation for easier API testing and collaboration with front-end developers.

**REST API Consumption**:

Used **RestTemplate** and **WebClient** for consuming external REST APIs, particularly in microservice architectures where services communicate with each other via REST.

14. How to implement security in rest api

Ans:- Implementing security in a REST API is crucial to ensure that only authorized users or systems can access sensitive data or perform specific actions. In a Spring Boot REST AP

**Basic Authentication** is a simple way of securing APIs where the client sends the username and password in the HTTP header with each request. This method is not very secure on its own because credentials are base64-encoded (not encrypted) and sent with every request.

Secures all endpoints except /public/\*\*

Requires a username and password for authentication.

**JWT (JSON Web Token)** is a more secure and stateless method where the client authenticates by sending a token (generated after a successful login) with each request. JWT tokens contain user identity and claims, allowing the server to verify the user without keeping session state.

Add the necessary dependency in pom.xml: jjwt

Create a utility class for generating and validating JWT tokens:

Create a filter to intercept requests and validate the JWT:

Update SecurityConfig to include JWT validation:

Create an authentication endpoint to issue JWT tokens:

For more complex needs, **OAuth2** allows you to implement role-based and delegated access.

15. How to maintain version in rest api

Ans:- In many scenarios, **URI Path Versioning** is the most common and easily understandable approach for both developers and consumers. However, for more complex API ecosystems, **Custom Header Versioning** or **Content Negotiation** might provide more flexibility.

16. Diff between post and put

Ans:- POST is generally used when you want to create a new resource on the server. The server generates a new resource ID.

Each POST request can result in the creation of a new resource, even if the data sent in the requests is the same.

Returns a 201 Created status and often includes the newly created resource's URL in the response.

PUT is used when you want to update an existing resource, or create it if it does not already exist.

It is idempotent, meaning making the same PUT request multiple times will not create multiple resources.

Returns a 200 OK status with the updated resource, or 201 Created if a new resource was created.

17. Why spring boot is preferable over spring mvc

Ans:- **Spring Boot** provides **auto-configuration** by default, which means it automatically configures the application based on the dependencies in the project.

**Spring MVC**, on the other hand, requires you to configure most components manually, which can lead to more complex and error-prone configurations.

 With **Spring Boot**, you can run your application with an embedded server, so you don't need to manually install and configure a web server.

 **Spring MVC** requires external server configuration (e.g., Tomcat or Jetty), or you need to package the application as a WAR file to be deployed to an external server.

 **Spring Boot** offers an easy-to-use **Spring Initializr** (<https://start.spring.io/>), which helps you generate a complete Spring Boot project with dependencies and configuration, allowing you to start coding right away.

 **Spring MVC** requires more manual setup, which could be a longer process when you just want to start developing.

18. What is the default server in spring boot and how to change it

Ans:-  **Default Server**: Spring Boot uses **Tomcat** by default.

 **To change the server**, exclude the default Tomcat dependency from spring-boot-starter-web and add the desired server dependency (e.g., **Jetty** or **Undertow**).

 **Server Configuration**: You can also configure the port and other settings via application.properties or application.yml.

19. What spring data jpa ,give some example

Ans:- **Spring Data JPA** is a part of the **Spring Data** project that simplifies the implementation of **JPA (Java Persistence API)** based data access layers. It provides a set of abstractions to reduce boilerplate code and make working with databases easier. it supports features like **automatic query generation**, **pagination**, **sorting**, and more.

20. What spring data jpa ,give some example

**Spring Data JPA** is a part of the **Spring Data** project that simplifies the implementation of **JPA (Java Persistence API)** based data access layers. It provides a set of abstractions to reduce boilerplate code and make working with databases easier. it supports features like **automatic query generation**, **pagination**, **sorting**, and more.

Add Dependencies in pom.xml (for Maven): To use Spring Data JPA, you need to include the data-jpa

Configure the Database Connection (application.properties): In your application.properties or application.yml, configure the connection to your database.

 **Entity**: Employee represents a table with fields like id, name, department, and salary.

 **Repository**: EmployeeRepository extends JpaRepository, which allows you to perform CRUD operations without writing custom SQL queries.

 **Service**: EmployeeService contains business logic, which uses EmployeeRepository to fetch and save data.

 **Controller**: EmployeeController exposes REST endpoints to interact with the service.

21. What is transaction in spring boot and how to implement it

Ans:- A transaction ensures that all the database operations are executed as a single unit of work. If any operation in a transaction fails, the entire transaction is rolled back, ensuring that the data remains consistent. Spring Boot supports both programmatic and declarative transaction management

The @Transactional annotation marks the methods that should be executed within a transaction. If the method executes successfully, the transaction will be committed. If an exception occurs, the transaction will be rolled back automatically.