**General Questions on Agile**

1. **What is Agile methodology?** Agile is an iterative and incremental approach to project management and software development that prioritizes flexibility, collaboration, and frequent delivery of working software. It encourages adapting to changes and customer feedback throughout the process.
2. **What is the Agile Manifesto?** The Agile Manifesto, published in 2001, outlines four core values:
   * Individuals and interactions over processes and tools.
   * Working software over comprehensive documentation.
   * Customer collaboration over contract negotiation.
   * Responding to change over following a plan.
3. **What is the difference between Agile and Waterfall?** Agile is iterative and flexible, delivering software incrementally. In contrast, Waterfall is a linear process with distinct phases, where testing occurs only after development is complete, making it harder to accommodate changes.

**Agile Roles**

1. **What does a Product Owner do?** The Product Owner represents the customer, prioritizes features in the product backlog, and ensures the team delivers value aligned with the client's needs.
2. **What is the role of a Scrum Master?** The Scrum Master facilitates the Agile process, removes obstacles for the team, and ensures adherence to Agile principles, but does not manage the team.
3. **What is the Delivery Team in Agile?** The Delivery Team consists of developers, testers, designers, and other specialists responsible for building and delivering the product.

**Processes and Concepts in Agile**

1. **What is a User Story in Agile?** A User Story describes a feature from the end user's perspective, typically written as: “As a [type of user], I want [functionality] so that [reason].”
2. **What is a Sprint?** A Sprint is a time-boxed period (commonly 1-4 weeks) during which the team works on delivering a set of user stories.
3. **What is a Product Backlog?** A Product Backlog is an ordered list of features or tasks that might be needed in the product. It evolves over time as priorities change.
4. **What is Velocity in Agile?** Velocity refers to the amount of work (in story points) a team can complete during a Sprint, helping measure progress and predict timelines.
5. **What is MVP (Minimum Viable Product)?** MVP is the simplest version of a product that can satisfy early customers and deliver value while allowing for feedback and further development.

**Agile Tools and Techniques**

1. **What is a Burn Down Chart?** A Burn Down Chart tracks progress over time, showing the amount of work completed versus the remaining work.
2. **What is Kanban?** Kanban visualizes work on a board with columns such as "To Do," "In Progress," and "Done," ensuring smooth workflow and minimizing bottlenecks.

**Agile Rituals**

1. **What is Sprint Planning?** Sprint Planning is a meeting where the team defines the scope of work for the next Sprint by selecting user stories from the backlog.
2. **What are Retrospectives in Agile?** Retrospectives are meetings held at the end of a Sprint to reflect on what went well, what could be improved, and how to adjust for future Sprints.

**General Questions on Spring Framework**

1. **Why was the Spring Framework developed?**
   * **Answer**: Spring was developed to simplify Java Enterprise application development by replacing complex frameworks like EJB with POJOs (Plain Old Java Objects). It introduced flexibility, modularity, and ease of testing.
2. **What is Inversion of Control (IOC)?**
   * **Answer**: IOC is a design principle where the control of object creation and dependency management is transferred to the framework (Spring). This promotes modularity and reduces coupling.
3. **What is Dependency Injection (DI)?**
   * **Answer**: DI is a technique to implement IOC, where dependencies are injected into objects by the framework rather than being created manually. It can be done via constructor, setter, or field injection.
4. **What are the advantages of using Spring Framework?**
   * **Answer**:
     + Simplifies development with POJOs.
     + Reduces boilerplate code with annotations.
     + Provides modularity and flexibility.
     + Supports integration with other frameworks.
     + Enables easy testing with dependency injection.

**Spring Dependency Injection**

1. **What is Constructor Injection in Spring?**
   * **Answer**: Constructor Injection is the default method of DI in Spring, where dependencies are passed as arguments to the constructor. It does not require the @Autowired annotation unless multiple constructors are present.
2. **What is Setter Injection in Spring?**
   * **Answer**: Setter Injection uses setter methods to inject dependencies. The @Autowired annotation is required for this type of injection.
3. **What is Field Injection in Spring? Why is it not recommended?**
   * **Answer**: Field Injection uses the @Autowired annotation directly on fields. It is not recommended because it makes dependencies less explicit and complicates testing.
4. **What happens if there are multiple beans of the same type?**
   * **Answer**: Spring throws a NoUniqueBeanDefinitionException if it cannot determine which bean to inject. This can be resolved using @Qualifier or specifying the bean name.

**Spring Configuration**

1. **What is the difference between BeanFactory and ApplicationContext?**
   * **Answer**:
     + BeanFactory is the basic IOC container that lazily initializes beans.
     + ApplicationContext is an advanced container that eagerly initializes beans and provides additional features like event handling and AOP.
2. **What is ClassPathXmlApplicationContext?**
   * **Answer**: It is a type of ApplicationContext that loads the Spring configuration file from the classpath.
3. **What is FileSystemXmlApplicationContext?**
   * **Answer**: It is a type of ApplicationContext that loads the Spring configuration file from the file system.
4. **What is AnnotationConfigApplicationContext?**
   * **Answer**: It is a type of ApplicationContext used when beans are configured using annotations like @Configuration and @Bean.

**Spring Beans**

1. **What is a Spring Bean?**
   * **Answer**: A Spring Bean is an object managed by the Spring IOC container. Beans are defined in configuration files or annotated with @Component, @Service, or @Repository.
2. **What are the different types of dependencies in Spring?**
   * **Answer**:
     + Primitive types (e.g., int, boolean).
     + Collection types (e.g., List, Set, Map).
     + Reference types (e.g., other beans).
3. **How does Spring reduce XML configuration overhead?**
   * **Answer**: Spring reduces XML configuration overhead by using annotations like @Autowired, @Component, @Configuration, and @Bean. These annotations simplify bean creation and dependency injection.
4. **What is Spring Boot?**
   * **Answer**: Spring Boot is a framework built on top of Spring that simplifies application development by providing auto-configuration, embedded servers, and minimal boilerplate code.
5. **What is the purpose of the** @SpringBootApplication **annotation?**
   * **Answer**: It is a combination of three annotations:
     + @EnableAutoConfiguration: Automatically configures the application based on the dependencies in the classpath.
     + @ComponentScan: Scans the package where the main class resides for Spring components like @Component, @Service, @Repository, and @Controller.
     + @Configuration: Indicates that the class contains Spring configuration methods.
6. **What is** SpringApplication.run() **used for?**
   * **Answer**: It starts the Spring Boot application, initializes the Spring IoC container, and returns an ApplicationContext object to access beans.
7. **How can you disable the Spring Boot banner?**
   * **Answer**: By creating an object of SpringApplication and calling setBannerMode(Banner.Mode.OFF) before running the application.

**Spring Boot Dependency Injection**

1. **What is** @Autowired **used for in Spring Boot?**
   * **Answer**: It is used for automatic dependency injection. Spring resolves the dependency by matching the type of the bean.
2. **What happens if there are multiple beans of the same type?**
   * **Answer**: Spring throws a NoUniqueBeanDefinitionException. This can be resolved using:
     + @Primary to mark one bean as the default.
     + @Qualifier to specify the exact bean to inject.
3. **What is the difference between** @Primary **and** @Qualifier**?**
   * **Answer**:
     + @Primary: Marks a bean as the default when multiple beans of the same type exist.
     + @Qualifier: Specifies the exact bean to inject by name.

**Spring Boot Configuration**

1. **What are the different types of configurations in Spring Boot?**
   * **Answer**:
     + **Java-based configuration**: Using @Configuration and @Bean annotations.
     + **XML-based configuration**: Using <bean> tags in XML files.
     + **Annotation-based configuration**: Using annotations like @Component, @Service, and @Repository.
2. **How do you create a bean in Java-based configuration?**
   * **Answer**:
     + Annotate a class with @Configuration.
     + Define a method with @Bean that returns an object of the desired type.
3. **What are the advantages of using annotations in Spring Boot?**
   * **Answer**:
     + Simplifies configuration by embedding it in code.
     + Reduces XML configuration overhead.
     + Provides compile-time error checks.
     + Enhances readability and maintainability.
4. **What does** @Component **do in Spring Boot?**
   * **Answer**: It marks a class as a Spring-managed bean, making it eligible for IoC container management.
5. **How does** @ComponentScan **work?**
   * **Answer**: It specifies the base packages to scan for annotated classes like @Component, @Service, @Repository, and @Controller.

**Dependency Injection and Bean Management**

1. **What is the use of** @Qualifier**?**
   * **Answer**: Resolves ambiguity by specifying the exact bean name to inject when there are multiple beans of the same type.
2. **What is the difference between** @Primary **and** @Qualifier**?**
   * **Answer**:
     + @Primary: Marks a bean as the default in case of ambiguity.
     + @Qualifier: Specifies a specific bean by name and overrides @Primary.
3. **What does** @Scope("prototype") **do?**
   * **Answer**: Specifies that a new instance of the bean is created every time it is accessed.
4. **How does Spring Boot handle dependency injection?**
   * **Answer**: Using annotations like @Autowired, Spring Boot automatically resolves and injects dependencies based on type and qualifiers.

**Configuration and Application Context**

1. **What is** @Configuration **used for?**
   * **Answer**: It designates a class as a configuration class and enables Java-based bean definitions using the @Bean annotation.
2. **How do you load a Java-based configuration class?**
   * **Answer**: By using AnnotationConfigApplicationContext:

java

ApplicationContext context = new AnnotationConfigApplicationContext(ApplicationConfig.class);

1. **What is the use of** @PropertySource**?**
   * **Answer**: It specifies a property file to use for externalized configurations. Values from the file can be injected using @Value.
2. **What is SpEL (Spring Expression Language)?**
   * **Answer**: SpEL allows dynamic expressions to manipulate or calculate values during injection, e.g., @Value("#{expression}").

**Spring Boot Beans and Lifecycle**

1. **What is** @Bean **in Spring Boot?**
   * **Answer**: It declares a method that returns a bean to be managed by the IoC container. The method name or custom name can be used as the bean name.
2. **What is** @PrePersist **and** @PreUpdate **used for?**
   * **Answer**:
     + @PrePersist: Runs before inserting an entity into the database, often used to populate fields.
     + @PreUpdate: Runs before updating an entity in the database, commonly used for validation and auditing.

**Error Handling and Miscellaneous**

1. **What happens if multiple beans of the same type exist?**
   * **Answer**: Spring Boot throws a NoUniqueBeanDefinitionException. This can be resolved using @Primary or @Qualifier annotations.
2. **How can you disable the Spring Boot banner?**
   * **Answer**: By creating an object of SpringApplication and calling setBannerMode(Banner.Mode.OFF).
3. **What is the purpose of the** <beans> **tag?**
   * **Answer**: It is the root element of the Spring XML configuration file, used to define beans and their dependencies.
4. **How is a bean defined in XML configuration?**
   * **Answer**: Beans are defined using the <bean> tag with attributes like id (bean name) and class (class name). Example: <bean id="myBean" class="com.example.MyClass"/>.
5. **What is the role of the** p-namespace**?**
   * **Answer**: The p-namespace provides a shorthand to set property values directly within the <bean> tag. Example: <bean p:name="value" />.

**Dependency Injection**

1. **How are values injected into a bean using XML?**
   * **Answer**: Use the <property> tag inside the <bean> definition. Example: <property name="propertyName" value="value"/>.
2. **How are collection types injected?**
   * **Answer**: Use tags like <list>, <set>, <map>, or <props> inside the <property> tag. Example:

xml

<property name="myList">

<list><value>Item1</value></list>

</property>

1. **How is a bean reference injected into another bean?**
   * **Answer**: Use the <ref> tag. Example: <property name="otherBean"><ref bean="otherBeanId"/></property>.

**Bean Lifecycle**

1. **How do you specify initialization and destruction methods?**
   * **Answer**: Use the init-method and destroy-method attributes in the <bean> tag. Example: <bean id="myBean" init-method="init" destroy-method="destroy"/>.
2. **What is the purpose of** registerShutdownHook()**?**
   * **Answer**: It ensures that destruction methods are called when the application context is closed.

**Scopes and Configurations**

1. **What are the different bean scopes in XML?**
   * **Answer**:
     + singleton: Single shared instance (default).
     + prototype: Creates a new instance for each request.
     + request, session, and globalSession: For web applications.
2. **What is the use of** @Required**?**
   * **Answer**: Ensures that a required property is set. If not, BeanInitializationException is thrown during initialization.

**Constructor Injection**

1. **How is constructor injection done in XML?**
   * **Answer**: Use <constructor-arg> with value or ref attributes. Example:

xml

<constructor-arg value="10"/>

<constructor-arg ref="otherBean"/>

1. **How is ambiguity resolved in constructor injection?**
   * **Answer**: By using the type, index, or name attributes in the <constructor-arg> tag.

**Properties and Placeholders**

1. **How do you read values from a properties file?**
   * **Answer**: Use <context:property-placeholder location="classpath:filename.properties"/> and inject values using placeholders like ${propertyName}.
2. **What happens if a property is missing in the properties file?**
   * **Answer**: Spring throws a BeanDefinitionStoreException.

**Miscellaneous**

1. **What is an inner bean?**
   * **Answer**: A bean defined within another bean's <property> tag. Example:

xml

<property name="innerBean">

<bean class="com.example.InnerBean"/>

</property>

1. **What is Aspect-Oriented Programming (AOP)?**
   * **Answer**: AOP is a programming paradigm that deals with cross-cutting concerns like logging, security, performance monitoring, and exception handling by centralizing them into reusable components called aspects.
2. **What are cross-cutting concerns in AOP?**
   * **Answer**: Cross-cutting concerns are functionalities that affect multiple classes, such as logging, security, or exception handling, but are not part of the business logic.
3. **What is an aspect in AOP?**
   * **Answer**: An aspect is a modular unit of cross-cutting concern, which contains methods called advices to implement specific functionality.

**Key Terms in AOP**

1. **What is advice in AOP?**
   * **Answer**: Advice is a method defined in an aspect class that contains the code to be executed at a specific join point in the application.
2. **What is a pointcut in AOP?**
   * **Answer**: Pointcut is an expression that specifies which methods or join points in the application should be advised by the aspect.
3. **What is a join point in AOP?**
   * **Answer**: Join point represents a specific point in the execution of a program, such as method calls, where advice can be applied.
4. **What is weaving in AOP?**
   * **Answer**: Weaving is the process of applying aspects (advices) to the target objects based on the defined pointcuts. It is handled by a weaver component.
5. **What is a proxy in AOP?**
   * **Answer**: A proxy is a dynamically generated class that combines business logic and advice logic, created during the weaving process.

**Frameworks and Implementations**

1. **What are some frameworks that support AOP?**
   * **Answer**:
     + **Spring AOP**: Used for runtime weaving and proxy-based AOP.
     + **AspectJ**: Provides compile-time and load-time weaving.
     + **JBoss AOP**: Used in JBoss applications.
2. **What is** @Aspect **in Spring AOP?**
   * **Answer**: @Aspect marks a class as an aspect and contains advices and pointcuts for cross-cutting concerns.

**Types of Advices**

1. **What does** @Before **do in AOP?**
   * **Answer**: Executes advice logic before the target method is invoked.
2. **What does** @After **do in AOP?**
   * **Answer**: Executes advice logic after the target method is invoked.
3. **What does** @AfterReturning **do in AOP?**
   * **Answer**: Executes advice logic after the target method completes successfully and returns a value.
4. **What does** @Around **do in AOP?**
   * **Answer**: Executes advice logic before and after the target method, and can control whether the target method is executed.
5. **What does** @AfterThrowing **do in AOP?**
   * **Answer**: Executes advice logic when the target method throws an exception.
6. **What is Spring Boot?**
   * **Answer**: Spring Boot is an extension of the Spring Framework that simplifies application development by offering convention over configuration, opinionated defaults, and minimal boilerplate code.
7. **What does “Convention over Configuration” mean in Spring Boot?**
   * **Answer**: It means Spring Boot follows pre-defined conventions and sensible defaults to minimize the need for manual configuration.
8. **What is meant by “Opinionated Defaults” in Spring Boot?**
   * **Answer**: Spring Boot automatically configures your application with best practices based on the jars included in the classpath.

**Annotations in Spring Boot**

1. **What is** @SpringBootApplication**?**
   * **Answer**: It is a combination of three annotations:
     + @EnableAutoConfiguration: Automatically configures beans based on classpath dependencies.
     + @ComponentScan: Scans for components like @Service and @Controller.
     + @Configuration: Marks the class as a configuration class.
2. **What is** @SpringBootTest **used for?**
   * **Answer**: It is used for writing integration tests, looking for the @SpringBootApplication class and running the entire application context.
3. **What is the difference between** @Service **and** @Repository**?**
   * **Answer**:
     + @Service: Marks a class as a business service component.
     + @Repository: Marks a class as a DAO component, and also provides exception translation.

**Configuration and Dependency Management**

1. **How does Spring Boot manage dependencies?**
   * **Answer**: Spring Boot uses starter dependencies (like spring-boot-starter-web), which include necessary libraries in a single POM entry.
2. **What is the role of** META-INF/spring.factories **in Spring Boot?**
   * **Answer**: It lists all autoconfiguration classes and condition checks used to enable auto-configuration.
3. **What are starter dependencies in Spring Boot?**
   * **Answer**: Starter dependencies are pre-configured POM entries that bundle multiple libraries needed for a specific feature, like spring-boot-starter-data-jpa for JPA support.

**Application Setup**

1. **What is the entry point of a Spring Boot application?**
   * **Answer**: The class containing the main() method and annotated with @SpringBootApplication.
2. **How does Spring Boot handle XML configuration?**
   * **Answer**: Spring Boot avoids XML configuration by relying on annotations, convention, and opinionated defaults. However, XML configuration can still be used if required.

**Testing and Lifecycle**

1. **What is the purpose of Spring Boot’s** application.properties**?**
   * **Answer**: It is used to configure application settings like port number, database URLs, and custom properties.
2. **What is actuator in Spring Boot?**
   * **Answer**: Actuator provides production-ready features like monitoring, metrics, and health checks of the application.

**Error Handling and Advanced Concepts**

1. **How does Spring Boot handle exceptions?**
   * **Answer**: It uses @ControllerAdvice for global exception handling and provides sensible error responses out of the box.
2. **What is the use of** spring-boot-starter-security**?**
   * **Answer**: It adds security features like authentication and authorization to the application using defaults, which can be customized as needed.
3. **What is a JDBC Template?**
   * **Answer**: It is a Spring utility class that simplifies interaction with the database by handling connection closing, preparing statements, and exception management. It uses the Template Design Pattern.
4. **What is DataSource in JDBC Template?**
   * **Answer**: A DataSource is an interface provided by javax.sql to manage database connections. It pools connections to optimize performance and avoid memory leaks.
5. **What is the purpose of a DriverManagerDataSource in Spring?**
   * **Answer**: It is a Spring implementation of the DataSource interface, used to create and manage database connections with driver class, URL, username, and password.

**Database Operations**

1. **What are the common methods of JDBC Template?**
   * **Answer**:
     + update(): Executes insert, update, and delete queries.
     + query(): Executes select queries and maps results to a list of objects.
     + queryForObject(): Executes select queries and maps a single row to an object.
2. **What is a RowMapper in JDBC Template?**
   * **Answer**: It is an interface used to map rows of data in a ResultSet to Java objects. The mapRow() method is overridden to define the mapping logic.
3. **What is the purpose of DTO and DAO in database applications?**
   * **Answer**:
     + DTO (Data Transfer Object): Carries data between layers.
     + DAO (Data Access Object): Provides an abstraction layer to interact with the database, using JDBC Template.

**SQL Queries**

1. **How does** queryForObject() **work in JDBC Template?**
   * **Answer**: It maps a single row or column to a Java object using SQL and a RowMapper. Example:

java

String sql = "SELECT name FROM employee WHERE id = ?";

String name = jdbcTemplate.queryForObject(sql, new Object[]{id}, String.class);

1. **How does** query() **differ from** queryForObject()**?**
   * **Answer**:
     + query(): Returns a list of mapped objects for multiple rows.
     + queryForObject(): Returns a single mapped object for one row.

**JDBC Technology**

1. **What are the main steps to use JDBC without Spring?**
   * **Answer**:
     + Import java.sql.\*.
     + Load and register the driver.
     + Create a connection.
     + Create and execute a statement.
     + Process the result.
     + Close the connection.
2. **What is the role of the** DriverManager **in JDBC?**
   * **Answer**: It manages a list of database drivers and provides connections using the getConnection() method.

**PreparedStatement vs Statement**

1. **How does PreparedStatement improve performance?**
   * **Answer**: PreparedStatement precompiles SQL queries, making execution faster compared to Statement, which parses and compiles queries every time.
2. **How does PreparedStatement prevent SQL Injection?**
   * **Answer**: It uses parameterized queries, avoiding direct concatenation of user input into SQL queries.

**JDBC Connection and Errors**

1. **What is a memory leak in database connections?**
   * **Answer**: A memory leak occurs when a database connection is opened but never closed, causing RAM to remain occupied indefinitely.
2. **What happens if no DataSource is used in JDBC Template?**
   * **Answer**: Connections must be created manually for every operation, leading to slower performance and risk of memory leaks.
3. **What are common exceptions in JDBC?**
   * **Answer**:
   * SQLException: General database access error.
   * DataIntegrityViolationException: Violation of database constraints.
   * DataAccessResourceFailureException: Failure in accessing the database resource.
4. **What is ORM and its purpose?**
   * **Answer**: ORM (Object-Relational Mapping) automatically maps Java objects to database tables and vice versa, eliminating the need for manual SQL queries and increasing developer productivity.
5. **What is JPA and how is it related to Hibernate?**
   * **Answer**: JPA (Java Persistence API) is a specification and API for ORM in Java. Hibernate is an implementation of JPA that provides additional features and customizations.

**Session Management**

1. **What are Session and SessionFactory in Hibernate?**
   * **Answer**:
     + Session: Represents a single unit of work with the database and provides methods to save, update, or query objects.
     + SessionFactory: A heavyweight object that creates and manages sessions; it is thread-safe and should be used as a singleton.
2. **What is the role of the** hibernate.cfg.xml **file?**
   * **Answer**: It is the Hibernate configuration file that contains database connection details, Hibernate properties, and mappings to entity classes.

**Entity Management**

1. **What is the difference between** @Entity **and** @Table**?**
   * **Answer**:
     + @Entity: Marks a class as a JPA entity.
     + @Table: Specifies the table name for the entity (if different from the class name).
2. **What is the use of the** @Transient **annotation?**
   * **Answer**: Fields marked with @Transient are not persisted in the database but still exist in the Java object.
3. **How is a many-to-many relationship implemented?**
   * **Answer**: Use @ManyToMany with @JoinTable to define the join table and its join columns. Example:

java

@ManyToMany

@JoinTable(name="student\_courses",

joinColumns=@JoinColumn(name="student\_id"),

inverseJoinColumns=@JoinColumn(name="course\_id"))

private List<Course> courses;

**Transactions**

1. **Why are transactions required in Hibernate?**
   * **Answer**: Transactions ensure that multiple operations are performed as a single unit of work. Without transactions, updates or inserts may fail, leaving the database in an inconsistent state.
2. **What is the use of** @Transactional **in Spring and Hibernate?**
   * **Answer**: It declares a method or class transactional, automatically managing commit or rollback based on the outcome of the transaction.

**Caching**

1. **What is Level 1 Cache in Hibernate?**
   * **Answer**: Level 1 Cache is the default caching mechanism in Hibernate, storing objects in the session’s memory. If an object is already present in the session, Hibernate fetches it from the cache instead of querying the database.

**Querying with Hibernate**

1. **What is HQL and how is it different from SQL?**
   * **Answer**: HQL (Hibernate Query Language) is an object-oriented query language specific to Hibernate, operating on entity objects rather than database tables.
2. **What are the methods to execute queries in Hibernate?**
   * **Answer**: Use Session.createQuery(HQL query) for HQL and Session.createNativeQuery(SQL query) for SQL.
3. **What is the difference between** Session.get() **and** Session.load()**?**
   * **Answer**:
     + get(): Fetches the object eagerly and returns null if the object does not exist.
     + load(): Fetches the object lazily and throws ObjectNotFoundException if the object does not exist.

**Hibernate Properties**

1. **What is the use of** hibernate.show\_sql **and** hibernate.format\_sql**?**
   * **Answer**:
     + hibernate.show\_sql: Enables logging of SQL queries executed by Hibernate.
     + hibernate.format\_sql: Formats SQL queries for readability.
2. **What is** hibernate.dialect**? Why is it required?**
   * **Answer**: It specifies the SQL dialect to be used based on the database (e.g., MySQLDialect for MySQL). Dialects handle database-specific query generation.
3. **What is Spring Data JPA?**
   * **Answer**: Spring Data JPA is a part of the Spring framework that simplifies database interactions using JPA. It provides an abstraction over JPA repositories with built-in methods for CRUD operations.
4. **What is JPQL? How is it different from SQL?**
   * **Answer**: JPQL (Java Persistence Query Language) is an object-oriented query language used in JPA that operates on entity classes and attributes rather than database tables and columns. SQL operates directly on database tables.

**Configuration**

1. **How do you configure Spring Data JPA for a MySQL database?**
   * **Answer**: Add configuration in application.properties:
   * spring.datasource.url=jdbc:mysql://localhost/db\_name
   * spring.datasource.username=user
   * spring.datasource.password=password
   * spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
   * spring.jpa.hibernate.ddl-auto=update
   * spring.jpa.show-sql=true
2. **What does** spring.jpa.hibernate.ddl-auto **do?**
   * **Answer**: It specifies the action to perform on the database schema, such as create, update, or none.

**Repository Methods**

1. **What does** Repo.save(Entity) **do?**
   * **Answer**: It saves the entity to the database. If the entity exists, it updates it; otherwise, it inserts a new record.
2. **What is the difference between** Repo.findById() **and** Repo.findAll()**?**
   * **Answer**:
     + findById(): Retrieves an entity by its ID and returns an Optional to handle null values.
     + findAll(): Retrieves all entities in the database.
3. **How does** Repo.saveAll(List) **work?**
   * **Answer**: It saves multiple entities in a batch to the database, reducing the number of queries.

**Custom Queries**

1. **How do you write a custom query using** @Query**?**
   * **Answer**: Use JPQL in the @Query annotation. Example:

java

@Query("select s from Student s where s.name=?1")

List<Student> findByName(String name);

1. **What is Domain-Specific Language (DSL) in JPA?**
   * **Answer**: DSL methods are automatically derived from method names following a pattern based on column names, such as findByName, findByAgeGreaterThan, etc.

**Optional Handling**

1. **Why does** findById() **return an** Optional**?**
   * **Answer**: To handle cases where the entity might not exist, preventing NullPointerException. Example:

java

Obj obj = repo.findById(1).orElse(new Obj());

1. **What is the difference between** Optional.orElse() **and** Optional.orElseGet()**?**
   * **Answer**:
     + orElse(): Creates a new default object even if the value is present.
     + orElseGet(): Only creates a default object if the value is absent.

**Spring Boot Starter**

1. **What is the role of** spring-boot-starter-data-jpa**?**
   * **Answer**: It bundles dependencies for Spring Data JPA and Hibernate, simplifying database integration.

**Transaction Management**

1. **How does Spring JPA handle transactions?**
   * **Answer**: Transactions are managed automatically by Spring, using the @Transactional annotation to ensure commit or rollback based on the outcome.

**Error Handling**

1. **What happens if you try to save a null entity with** Repo.save()**?**
   * **Answer**: A runtime exception, such as IllegalArgumentException, is thrown.

**Performance Optimization**

1. **How can batch operations improve performance in Spring JPA?**
   * **Answer**: Using methods like saveAll() or configuring Hibernate for batching reduces the number of queries, improving performance.

**SLF4J (Simple Logging Facade for Java)**

1. **What is SLF4J?**
   * SLF4J provides a simple and flexible logging abstraction for Java. It allows you to plug in various logging frameworks, such as Logback, Log4j, etc., without changing your code.
2. **How to use SLF4J?**
   * Add SLF4J dependency in your project.
   * Define a logger:

java

private static final Logger log = LoggerFactory.getLogger(ClassName.class);

* + Log messages:

java

log.info("This is an info message.");

log.error("This is an error message.");

**Lombok**

Lombok is a Java library that minimizes boilerplate code, such as getters, setters, constructors, and more, using annotations.

1. **Key Lombok Annotations**:
   * @Getter **and** @Setter: Automatically generate getter and setter methods for fields.

java

@Getter @Setter

private String name;

* + @ToString: Generates a toString() method.

java

@ToString

public class User {}

* + @EqualsAndHashCode: Automatically generate equals() and hashCode() methods.

java

@EqualsAndHashCode

public class User {}

* + @NoArgsConstructor **and** @AllArgsConstructor: Create constructors.

java

@NoArgsConstructor

@AllArgsConstructor

public class User {}

* + @RequiredArgsConstructor: Generates a constructor for fields marked as final or @NonNull.

java

@RequiredArgsConstructor

private final String name;

* + @Data: Shortcut for @Getter, @Setter, @ToString, @EqualsAndHashCode, and @RequiredArgsConstructor.

java

@Data

public class User {}

* + @Builder: Implements the Builder design pattern.

java

User user = User.builder().name("John").age(30).build();

* + @Value: Creates immutable classes.

java

@Value

public class User {

private final String name;

private final int age;

}

1. **Example with Autowiring**:
   * Using @Setter(onMethod\_=@Autowired) applies the @Setter annotation with @Autowired for dependency injection:

java

@Setter(onMethod\_ = @Autowired)

private MyService myService;

1. **What is Spring REST?**
   * **Answer**: Spring REST simplifies the development of web applications by using RESTful principles to map HTTP requests to methods in controller classes. It efficiently handles JSON and XML for communication.
2. **What does** @RestController **do?**
   * **Answer**: Combines @Controller and @ResponseBody, enabling classes to handle HTTP requests and return responses directly without rendering traditional views.
3. **What is the difference between** @RequestMapping **and** @GetMapping**?**
   * **Answer**:
     + @RequestMapping: Maps HTTP requests to a method and accepts all request types unless explicitly specified.
     + @GetMapping: Specifically maps HTTP GET requests.

**Annotations and HTTP Methods**

1. **What is** @PathVariable**? How is it used?**
   * **Answer**: It binds a method parameter to a URI variable. Example:

java

@GetMapping("/{id}")

public Event findEventById(@PathVariable("id") int id) {}

1. **What is** @RequestBody **used for?**
   * **Answer**: It deserializes incoming JSON or XML request data into Java objects.
2. **What is** @RequestParam**?**
   * **Answer**: It binds a method parameter to query parameters in the request URL. Example:

java

@GetMapping("/events")

public List<Event> getEvents(@RequestParam("type") String type) {}

1. **What are the main HTTP methods and their mapping annotations in Spring REST?**
   * **Answer**:
     + GET: @GetMapping
     + POST: @PostMapping
     + PUT: @PutMapping
     + DELETE: @DeleteMapping
     + PATCH: @PatchMapping

**Error Handling**

1. **How do you set custom HTTP response status using Spring REST?**
   * **Answer**: Use @ResponseStatus. Example:

java

@ResponseStatus(HttpStatus.NOT\_FOUND)

public class ResourceNotFoundException extends RuntimeException {}

**Serialization and Deserialization**

1. **What does** @JsonIgnoreProperties **do?**
   * **Answer**: Specifies fields to be ignored during serialization and deserialization of JSON, preventing unwanted fields from being included.
2. **How does Jackson integrate with Spring REST?**
   * **Answer**: Jackson is the default JSON processor in Spring REST. It converts Java objects to JSON and vice versa. Jackson Dataformat XML can be used to handle XML data.

**Relationships and DTOs**

1. **What issue occurs in bidirectional relationships during JSON serialization? How can it be resolved?**
   * **Answer**: An infinite loop can occur when one entity continuously references another. Solutions include:
     + @JsonIgnore to ignore certain fields.
     + @JsonManagedReference and @JsonBackReference to manage serialization.
2. **Why are DTOs used in Spring REST?**
   * **Answer**: Data Transfer Objects (DTOs) are used to encapsulate data, prevent exposing entity structure, and simplify communication between layers.

**Configuration and Entity Mapping**

1. **What does** @Column(unique=true) **do?**
   * **Answer**: Enforces uniqueness constraint on a column in the database.
2. **How does** @GeneratedValue(strategy=GenerationType.IDENTITY) **work?**
   * **Answer**: It auto-generates unique IDs for entities by delegating ID generation to the database.

**Custom Response and Content Types**

1. **How do you specify custom response types and input types in Spring REST?**
   * **Answer**: Use produces and consumes attributes in mapping annotations. Example:

java

@GetMapping(path = "/product", produces = "application/json")

public ResponseEntity<Product> getProduct() {}

@PostMapping(path = "/product", consumes = "application/xml")

public ResponseEntity<Product> createProduct(@RequestBody Product product) {}

**What is H2 Database?**

1. **In-Memory Database**: H2 is an in-memory relational database. Data exists only during the runtime of the application and is cleared upon shutdown. This makes it ideal for development and testing environments.
2. **Embedded Mode**: H2 can operate as an embedded database within your application, making it lightweight and easy to use.

**How to Configure H2 with Spring Boot**

1. **Dependencies**:
   * Add the spring-boot-starter-data-jpa dependency in your pom.xml. Spring Boot automatically includes H2 for testing purposes.
   * For explicit configuration of H2, add this dependency:

xml

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

1. **Basic Configuration in** application.properties:

properties

spring.datasource.url=jdbc:h2:mem:soh

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

1. **Understanding the Properties**:
   * spring.datasource.url: Defines the JDBC URL for the H2 database. The mem: part indicates an in-memory database.
   * spring.datasource.driver-class-name: Specifies the driver class for H2.
   * spring.datasource.username and password: Default credentials (sa and blank password).
   * spring.jpa.hibernate.ddl-auto=update: Automatically creates or updates database tables based on your entity classes.
   * spring.jpa.show-sql=true: Enables SQL query logging for debugging purposes.

**Accessing the H2 Database Console**

1. Enable the H2 web console in application.properties:

properties

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console

1. After starting your Spring Boot application, access the console via http://localhost:8080/h2-console.
2. Provide the JDBC URL (jdbc:h2:mem:soh) and the credentials (sa / blank) to connect to the database.

**Usage in Spring Boot**

1. **Entity Example**:

java

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

// Getters, setters, constructors

}

1. **Repository Example**:

java

@Repository

public interface UserRepository extends JpaRepository<User, Long> {}

1. **Saving Data**:

java

@RestController

public class UserController {

@Autowired

private UserRepository userRepository;

@PostMapping("/users")

public User createUser(@RequestBody User user) {

return userRepository.save(user);

}

}

**Benefits of H2 Database**

* Lightweight and fast.
* Ideal for development and testing.
* No need for external setup.
* Compatibility with various database dialects (via JDBC).