1. Hibernate and Hibernate Architecture:

Hibernate is an open-source Object-Relational Mapping (ORM) framework for Java. It simplifies the process of interacting with a database by providing a high-level abstraction layer over the database. With Hibernate, developers can work with plain Java objects instead of SQL statements, which reduces the amount of boilerplate code and makes the application more maintainable.

Hibernate Architecture: Hibernate architecture is based on the Model-View-Controller (MVC) design pattern. It consists of four main layers:

Hibernate Configuration: This layer is responsible for configuring Hibernate and setting up the session factory, which is used to create sessions.

Object-Relational Mapping (ORM) Layer: This layer maps Java objects to relational database tables. It includes the Hibernate mapping file or annotations that describe how Java classes are mapped to database tables.

Data Access Layer: This layer is responsible for executing database operations. It includes Hibernate's session API, which provides methods for creating, updating, deleting, and querying data.

Database Layer: This layer is the physical database where the data is stored.

Overall, Hibernate architecture provides a clean separation of concerns between application logic and database access. This makes the application more modular and easier to maintain over time.

Session tracking and how to do it in Java:

Session tracking is the process of maintaining the state of a user's interaction with a web application across multiple requests. This is important because HTTP, the protocol used for communication between a web browser and a web server, is stateless, meaning that it does not maintain any information about previous requests or user sessions. To implement session tracking in a Java web application, we can use one of the following methods:

Cookies: Cookies are small text files that are stored on the user's computer by the web browser. They can be used to store information about the user's session, such as a session ID, which can be used to identify the user across multiple requests.

URL Rewriting: In this method, the session ID is appended to the URL of each request. This allows the server to maintain state information across requests without relying on cookies.

Hidden Form Fields: In this method, the session ID is stored as a hidden field in an HTML form. When the form is submitted, the session ID is sent to the server along with the form data.

HttpSession: The HttpSession is a server-side object that provides a way to store session-specific information. It allows developers to store and retrieve data associated with a user's session.

Difference between Redirect and RequestDispatcher scenario:

RequestDispatcher: RequestDispatcher is used to forward the request from one servlet or JSP to another servlet or JSP within the same web application. The forward is done on the server-side and the client is not aware of it. The original request and response objects are forwarded to the next resource, allowing it to access

and modify the request and response objects. The RequestDispatcher is typically used in scenarios where the processing of a request needs to be split into multiple components.

Redirect: Redirect, on the other hand, is used to redirect the request to a different URL, either within the same web application or to a different web application. It is done on the client-side, which means that a new request and response objects are created and sent to the new URL. It is typically used in scenarios where we want to send the user to a different URL after performing a specific action.

Servlet Life Cycle:

Loading: When the server starts up or when a request is made for the servlet for the first time, the servlet container loads the servlet class into memory.

Instantiation: After the servlet class is loaded, the servlet container creates an instance of the servlet class by calling its no-argument constructor.

Initialization: Once the servlet is instantiated, the servlet container calls its init() method. The init() method is used to perform any necessary initialization of the servlet, such as loading configuration data or establishing database connections.

Handling Requests: Once the servlet has been initialized, it can begin handling requests. When a request is made to the servlet, the servlet container calls the servlet's service() method to process the request.

Destroying: When the servlet container determines that the servlet is no longer needed (e.g., when the server is shutting down), it calls the servlet's destroy() method. This method is used to clean up any resources used by the servlet, such as closing database connections or releasing memory.

JSP Life Cycle:

Translation: When a JSP is first accessed, the JSP container translates the JSP file into a Java servlet source file.

Compilation: The JSP container compiles the generated Java servlet source file into a Java class.

Instantiation: The JSP container creates an instance of the servlet class by calling its no-argument constructor.

Initialization: Once the servlet is instantiated, the JSP container calls its init() method. The init() method is used to perform any necessary initialization of the JSP, such as loading configuration data or establishing database connections.

Handling Requests: Once the JSP has been initialized, it can begin handling requests. When a request is made to the JSP, the JSP container calls the servlet's service() method to process the request.

Destroying: When the JSP container determines that the JSP is no longer needed (e.g., when the server is shutting down), it calls the servlet's destroy() method. This method is used to clean up any resources used by the JSP, such as closing database connections or releasing memory.

Hibernate Entity Lifecycle:

Transient: In the transient state, the entity is not associated with a session or persistence context.

Persistent: In the persistent state, the entity is associated with a session or persistence context, and any changes made to the entity will be saved to the database when the session is flushed.

Detached: In the detached state, the entity is no longer associated with a session or persistence context, but any changes made to the entity will be saved to the database when it is re-attached to a session.

Removed: In the removed state, the entity has been marked for deletion, and its changes will be saved to the database when the session is flushed.

Difference between get() and load() of Hibernate:

get() and load() are two methods provided by Hibernate to retrieve an object from the database based on its identifier. Both methods can be used to fetch an object, but they differ in their behavior and the scenarios in which they are used.

get(): The get() method is used to fetch an object from the database by its identifier. If the object with the specified identifier exists in the database, the get() method returns the object, otherwise, it returns null. The get() method always hits the database and retrieves the object immediately. If the object is not found, it returns null.

load(): The load() method is used to retrieve an object from the database based on its identifier. Unlike the get() method, the load() method does not immediately hit the database to retrieve the object. Instead, it returns a proxy object that represents the object with the given identifier. If the object is not found in the database, the load() method throws an ObjectNotFoundException. The load() method is useful in scenarios where we want to retrieve an object lazily or fetch it only when it is needed.

The main differences between the get() and load() methods are:

Return type: The get() method returns the actual object, while the load() method returns a proxy object that represents the object with the given identifier.

Database hit: The get() method always hits the database immediately, while the load() method does not hit the database immediately but returns a proxy object.

Exception handling: The get() method returns null if the object is not found, while the load() method throws an ObjectNotFoundException if the object is not found.

How to call a stored procedure in Hibernate:

Hibernate provides the storedprocedureQuery•\ API to call a stored procedure. Here's an example of calling a stored procedure that takes two input parameters and returns a single result:

```
StoredProcedureQuery query = session.createStoredProcedureQuery("procedure_name");
query.registerStoredProcedureParameter(1, String.class, ParameterMode.IN);
query.registerStoredProcedureParameter(2, Integer.class, ParameterMode.IN);
query.registerStoredProcedureParameter(3, Integer.class, ParameterMode.OUT);

query.setParameter(1, "parameter1_value");
query.setParameter(2, parameter2_value);

query.execute();

Integer result = (Integer) query.getOutputParameterValue(3);
```

What is IOC and Dependency Injection:

Inversion of Control (IoC) is a design pattern where the control of a program is inverted, meaning that the framework or container manages the lifecycle of objects and controls the flow of the application. Dependency Injection (DI) is a sub-pattern of IoC that involves injecting dependencies into a class instead of creating the dependencies within the class.

What is auto-wiring and its types:

Auto-wiring is a feature in Spring that allows dependencies to be automatically injected into a bean. There are four types of auto-wiring:

ByName: Spring matches the name of the bean with the name of the dependency.

ByType: Spring matches the type of the bean with the type of the dependency.

Constructor: Spring uses the constructor to autowire the bean.

Autodetect: Spring first attempts to autowire by constructor, then falls back to byType.

If a prototype bean is auto-wired into a singleton bean, a new instance of the prototype bean will be created every time the singleton bean is used, which may result in unexpected behavior and performance issues.

Spring Bean Lifecycle and Scopes:

Spring Bean Lifecycle involves several phases: Bean Definition Bean Initialization Bean Post-Processing Bean Destruction

Spring Bean Scopes define the lifecycle of a bean within an application context. There are five scopes:

Singleton: Only one instance of the bean is created and shared across the application context.

Prototype: A new instance of the bean is created each time it is requested from the container.

Request: A new instance of the bean is created for each HTTP request.

Session: A new instance of the bean is created for each HTTP session.

Global Session: A new instance of the bean is created for each global HTTP session.

What is use of @Transactinal? Why it should be used on service layer?

The @Transactional annotation is used in Spring to indicate that a method should be executed within a transactional context. In other words, it ensures that a set of operations are executed as an atomic unit of work, which either succeeds as a whole or fails as a whole.

When used on the service layer, @Transactional ensures that a single unit of work involving multiple database operations is executed as a single transaction. For example, if a service method involves reading data from one or more tables, updating some data in one of the tables, and writing data to another table, @Transactional ensures that either all of these operations are committed to the database or none of them are.

The benefit of using @Transactional on the service layer is that it helps to ensure data consistency and integrity. If one of the database operations fails, @Transactional ensures that all of the other operations are rolled back, so that the database is left in a consistent state.

In short, @Transactional is an important annotation that helps to ensure data consistency and integrity in a Spring application.

Explain Spring MVC life cycle

In simple terms, the Spring MVC lifecycle is the sequence of events that occur when a request is made to a Spring MVC-based web application.

Here are the high-level steps involved in the Spring MVC lifecycle:

1)The client sends a request to the web server for a particular URL.

2)The DispatcherServlet of the Spring MVC framework receives the request.

3)The DispatcherServlet consults the HandlerMapping to determine which controller will handle the request.

4)Once the appropriate controller is determined, the DispatcherServlet passes the request to the controller.

5)The controller executes the request and returns a ModelAndView object, which contains the view name and the model data.

6)The DispatcherServlet uses the ViewResolver to determine the view that will be used to render the response. Finally, the DispatcherServlet passes the model data to the view for rendering, and the response is sent back to the client.

What is the difference between SOAP and REST? What is the significance of RestController?

SOAP (Simple Object Access Protocol) and REST (Representational State Transfer) are two different web service architectures.

SOAP is a protocol for exchanging structured information using XML while REST is an architectural style that uses a simple URL to access resources.

RestController is a special type of Controller in Spring Framework that is used to handle RESTful web services. It combines the functionality of the @Controller and @ResponseBody annotations to simplify the development of RESTful web services.

What is Spring Boot? What do you mean by opinionated defaults? How does auto-configuration work?

Spring Boot is a framework that is used to simplify the creation of Spring-based applications. It provides a set of opinionated defaults that SOAP (Simple Object Access Protocol) is a messaging protocol that uses XML for encoding messages. It relies on a set of well-defined rules for building and exchanging messages between systems, typically over HTTP or other application layer protocols. SOAP is often used in enterprise applications where a high degree of reliability and security is required.

REST (Representational State Transfer) is an architectural style for building web services that uses a more lightweight and flexible approach. REST is based on the concept of resources, which are identified by URIs (Uniform Resource Identifiers), and can be accessed using standard HTTP methods such as GET, POST, PUT, and DELETE. REST is often used in web and mobile applications, where performance and scalability are more important than strict adherence to messaging protocols.

As for the significance of RestController, it is a specific class in Spring MVC, a popular Java web framework, that is used for building RESTful web services. It is a specialized version of the Controller class that is designed to handle RESTful requests and responses. It provides a convenient way to map HTTP requests to specific methods in the controller, and to return the results as JSON or XML data. It is a key component in building modern web and mobile applications that rely on RESTful APIs for communication between the client and server.

What is the significance of @CrossOrigin and how does it work?

@CrossOrigin is an annotation in Spring Framework that is used to allow cross-origin requests from a web browser to a Spring MVC controller. It can be used to configure the CORS (Cross-Origin Resource Sharing) policy for a specific controller or for the entire application.

CORS is a security feature in web browsers that prevents web pages from making requests to a different domain than the one that served the page. The @CrossOrigin annotation bypasses this security feature by allowing requests from other domains to be processed by the controller. It works by adding the necessary headers to the HTTP response to allow the browser to make the request.