Unit - 1 (Foundation of Enterprise Programming)

MCQ Questions

1. What does JDBC stand for?

   A. Java Data Connection
   B. Java Database Connectivity
   C. Java Direct Connect
   D. Java Data Communication

2. Which of the following is NOT a component of JDBC?

   A. JDBC Driver
   B. JDBC API
   C. JDBC Connection Pool
   D. JDBC Database

3. Which type of JDBC driver is also known as the Native-API driver?

   A. Type 1: JDBC-ODBC Bridge driver
   B. Type 2: Native-API driver
   C. Type 3: Network Protocol driver
   D. Type 4: Thin driver

4. Which JDBC driver type is most suitable for web applications due to its platform independence and performance?

   A. Type 1: JDBC-ODBC Bridge driver
   B. Type 2: Native-API driver
   C. Type 3: Network Protocol driver
   D. Type 4: Thin driver

5. What is the correct way to load the Oracle JDBC driver in a Java program?

   A. Class.forName("oracle.jdbc.OracleDriver");
   B. DriverManager.loadDriver("oracle.jdbc.driver.OracleDriver");
   C. Connection conn = new oracle.jdbc.OracleDriver().connect();
   D. DriverManager.registerDriverfnew oracle.jdbc.OracleDriver());

6. Which method is used to establish a connection to an Oracle database?

   A. DriverManager.getConnection(String url)
   B. DriverManager.getConnection(String url, Properties info)
   C. DriverManager.getConnection(String url, String user, String password)
   D. All of the above

7. What is the default port number used by Oracle for database connections?

   A. 1521
   B. 1433
   C. 3306
   D. 8080

8. How do you create a Statement object in JDBC?

   A. Statement stmt = new Statement(conn);
   B. Statement stmt = conn.createStatement();
   C. Statement stmt = conn.newStatement();
   D. Statement stmt = conn.getStatement();

9. Which Eclipse plugin is commonly used to integrate Maven with Eclipse?

A. M2Eclipse
B. Maven2IDE
C. EclipseMaven
D. MavenEclipse

10. What is the purpose of the pom.xml file in a Maven project?

A. To configure the IDE settings
B. To manage project dependencies and build configuration
C. To specify the project's Java version
D. To list all Java classes in the project

11. How can you convert an existing Eclipse project to a Maven project?

A. Right-click on the project, select Convert to Maven Project
B. Right-click on the project, select Configure > Convert to Maven Project
C. Right-click on the project, select Add Maven Nature
D. Right-click on the project, select Build Path > Convert to Maven Project

12. Which Maven command in Eclipse is used to update the project's dependencies and configurations from the pom.xml file?

A. Maven > Update Project
B. Maven > Refresh Dependencies
C. Maven > Synchronize
D. Maven > Rebuild

Unit - 1 (Foundation of Enterprise Programming)

Fill in the blanks

1) **XML** is a markup language used to encode documents in a format that is both human-readable and machine-readable.
2) In an XML document, **Opening and Closing** tags are used to define the data elements.
3) **JDBC** is an API in Java used for connecting and executing queries with databases
4) In JDBC architecture, the **Driver Manager** acts as a bridge between the application and the database
5) The **Type-4** driver type is known as the Thin driver in JDBC.
6) To establish a connection with an Oracle database using JDBC, you typically use the **Oracle Thin** driver.
7) In JDBC, the **executeQueryQ** method is used to execute a SQL query and return a ResultSet.
8) The **createStatementQ** method in the Connection interface is used to create a Statement object for sending SQL statements to the database.
9) In JDBC, the **PreparedStatementQ** object is used to execute precompiled SQL queries.
10) **MySql** is a popular open-source relational database management system used with JDBC.
11) In MySQL, the default port number for connecting to the database is **3306**
12) **Maven** is a build automation tool used primarily for Java projects
13) In Maven, the **POM.XML** file is used to configure project dependencies and other settings.
14) To integrate Maven with Eclipse, you typically install the **M2Eclipse** plugin.
15) In a POM.xml file, the **<dependencies>** element is used to specify dependencies for the project.
16) The **mvn compile** command in Maven is used to compile the source code of the project.
17) In Maven, the **<groupId>** element specifies the unique identifier for a project.
18) To add a MySQL JDBC driver dependency in a Maven project, you include it in the **<dependencies>** section of the POM.xml.
19) In a Maven POM.xml file, the **<main>** element is used to specify the main class to be executed.
20) In JDBC, a **CallableStatement** is used to call stored procedures in a database.

Unit - 1 (Foundation of Enterprise Programming)

Short Questions

1. What is JDBC?

JDBC (Java Database Connectivity) is an API that allows Java programs to interact with a wide range of databases.

2. Name the main components of JDBC.

The main components of JDBC are: JDBC API, JDBC Driver Manager, JDBC Test Suite, JDBC-ODBC Bridge.

3. What is the purpose of the JDBC Driver Manager?

The JDBC Driver Manager is used to manage a list of database drivers. It handles the establishment of connections between the Java application and the database.

4. List the types of JDBC drivers.

Type 1: JDBC-ODBC Bridge Driver, Type 2: Native-API Driver, Type 3: Network Protocol Driver, Type 4: Thin Driver.

5. What is the main advantage of the Type 4 JDBC driver?

The main advantage of the Type 4 driver is that it is platform-independent, as it is written entirely in Java.

6. Which JDBC driver type is known as the 'Pure Java' driver?

The Type 4 driver is known as the 'Pure Java' driver.

7. What are the core interfaces provided by JDBC?

The core interfaces are: Driver, Connection, Statement, PreparedStatement, CallableStatement, ResultSet, ResultSetMetaData, DatabaseMetaData.

8. Explain the Connection interface in JDBC.

The Connection interface provides methods for establishing a connection to a database, as well as for managing transactions and closing the connection.

9. How do you establish a connection to a database in JDBC?

You establish a connection by using the DriverManager.getConnection method with the appropriate database URL, username, and password.

10. Provide an example of a JDBC URL for connecting to a MySQL database.

jdbc:mysql://localhost:3306/mydatabase

11. What are CRUD operations?

CRUD operations refer to Create, Read, Update, and Delete operations performed on a database.

12. Write a sample code snippet for a SELECT query using JDBC.

```
Connection conn = DriverManager.getConnectionfurl, username, password);

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQueryf'SELECT * FROM users");

while (rs.next()) {

        System, out. printings.getString("username"));

}

conn.close));
```

13. What is the RowSet interface in JDBC?

The RowSet interface is a part of the JDBC API that encapsulates the retrieval of data in a tabular format and provides additional features like scrolling and updating rows.

14. Name different types of RowSet implementations.

JdbcRowSet,

CachedRowSet,

WebRowSet,

JoinRowSet,

Filtered RowSet.

15. What is a JdbcRowSet?

A JdbcRowSet is a connected RowSet object, meaning it maintains a connection to the database while processing data.

16. Explain CachedRowSet.

A CachedRowSet is a disconnected RowSet object, meaning it can operate without maintaining a connection to the database. It caches data in memory.

17. What is a WebRowSet?

A WebRowSet is an extension of CachedRowSet that can read and write XML data.

18. Describe JoinRowSet.

A JoinRowSet allows two or more RowSet objects to be combined based on a common column.

19. What is a FilteredRowSet?

A FilteredRowSet provides the capability to apply filtering criteria to rows in a RowSet.

20. How do you configure Apache Tomcat server in Eclipse IDE?

Download and install Apache Tomcat.

Open Eclipse and go to Window > Preferences > Server > Runtime Environments.

Click Add and select Apache Tomcat.

Specify the Tomcat installation directory and click Finish.

Unit - 1 (Foundation of Enterprise Programming)

Long Questions

1. What is XML (extensible Markup Language)? How is it different from HTML?

XML is a markup language designed to store and transport data. Unlike HTML, which defines the presentation of data, XML focuses on describing the structure and meaning of data. XML allows for custom tags and is designed to be self-descriptive, making it ideal for data interchange between different systems and platforms.

2. Explain XML Schema (XSD) and its importance in XML documents.

XML Schema (XSD) is a language for describing the structure and constraints of XML documents. It defines the elements, attributes, data types, and rules for valid XML documents in a particular namespace. XSD ensures that XML documents conform to a specific structure and format, providing validation and ensuring interoperability between systems that exchange XML data.

3. What is JDBC (Java Database Connectivity)? Describe its architecture and the role of JDBC drivers.

JDBC is an API that allows Java applications to interact with databases. Its architecture consists of a JDBC API, Driver Manager, Drivers, and Database. The JDBC Driver Manager manages a list of database drivers. Drivers translate Java calls into database-specific calls.

JDBC Drivers are categorized into four types:

JDBC-ODBC Bridge,

Native-API Driver,

Network Protocol Driver, and

Thin Driver.

JDBC facilitates connection establishment, statement execution, result set retrieval, and transaction management with databases.

4. How do you establish a connection to a database using JDBC?

Provide a step-by-step explanation.

To connect to a database using JDBC:

1. Load the database driver using Class.forName() or let JDBC 4.0+ auto-load.
2. Create a connection URL specifying the database location, username, and password.
3. Establish a connection using DriverManager.getConnection(url, username, password).
4. Optionally, set auto-commit or transaction isolation levels.
5. Execute SQL queries/statements using Statement, PreparedStatement, or CallableStatement. 6. Handle results and exceptions appropriately.

5.What are PreparedStatement objects in JDBC? How are they different from Statement objects?

PreparedStatement in JDBC is a precompiled SQL statement that can accept input parameters at runtime. It enhances performance by precompiling and caching the SQL statement, making it efficient for repeated executions with different parameters. Unlike Statement, which is used for static SQL queries, PreparedStatement is suitable for dyna

mic queries and prevents SQL injection attacks through parameterized queries.

6. Explain the ResultSet interface in JDBC. How do you navigate through a ResultSet and retrieve data?

ResultSet in JDBC represents the result set of a SQL query. It provides methods to navigate through rows and retrieve data from columns. Steps to navigate and retrieve data: 1. Use next() to move to the next row. 2. Use getXxx() methods (getString(), getInt(), etc.) to retrieve column values by index or column name. 3. Iterate until next() returns false. 4. Close the ResultSet and Statement objects after use to release resources.

7. What are transactions in JDBC? How do you manage transactions using JDBC?

Transactions in JDBC are sequences of operations that are treated as a single unit of work, ensuring data consistency and integrity.

Manage transactions using:

1. Begin a transaction with connection.setAutoCommit(false) or connection.setTransactionIsolation().

2. Execute SQL statements.

3. Commit the transaction using connection. commit() or rollback using connection, ro 11 back() in case of errors.

4. Set auto-commit to true or handle exceptions in a try-catch-finally block. JDBC supports transaction management through Connection methods and SQL commands (COMMIT, ROLLBACK)

8. How does JDBC batch processing improve performance? Explain with an example.

JDBC batch processing allows executing multiple SQL statements in a batch, reducing communication overhead with the database and improving performance.

Example:

1. Create a Statement or PreparedStatement.

2. Add SQL queries using addBatch().

3. Execute batch using executeBatch().

4. Process results or handle exceptions. Batch processing is useful for bulk inserts, updates, or deletes where multiple similar queries are executed together.

9. How do you connect to an Oracle database using JDBC? Describe Oracle-specific considerations.

To connect to Oracle using JDBC:

1. Include Oracle JDBC driver in the classpath (ojdbc.jar).

2. Use Class.forName() to load the driver.

3.Construct connection URL (jdbc:oracle:thin:@hostname:port:SID).

4. Obtain connection using DriverManager.getConnection(url, username, password).

5. Handle exceptions and close resources. Considerations: Oracle specific JDBC URLformat, driver version compatibility, and configuration of Oracle JDBC properties like connection pooling and performance tuning.

10. Explain how to connect to a MySQL database using JDBC. Discuss MySQL-specific configurations.

Connect to MySQL using JDBC:

1. Include MySQL JDBC driver (mysql-connector-java.jar).

2. Load driver using Class.forName() or allow auto-loading.

3. Create connection URL (jdbc:mysql://hostname:port/database).

4. Establish connection with DriverManager.getConnection(url, username, password).

5. Handle exceptions and close resources. MySQL considerations: JDBC URL format, driver version compatibility, and configuring connection parameters like SSL, timezone, and character encoding.

11. What is Maven? Describe its features and benefits in software development.

Maven is a build automation tool that manages project dependencies, builds, and documentation. Features include dependency management, project lifecycle management, build automation, and plugins for various tasks. Benefits include standardized project structure, dependency resolution, continuous integration support, and ease of project management with centralized configuration (pom.xml).

12. Explain the typical structure of a Maven project. What are the key files and directories in a Maven project?

Maven project structure:

1. src/main/java: Java source files.

2. src/main/resources: Non-Java resources.

3. src/test/java: Test source files.

4. src/test/resources: Test resources.

5. pom.xml: Project Object Model (POM) configuration.

6. target: Compiled output and generated artifacts. Key files: pom.xml (configuration), source directories (java, resources), and target (output directory).

13. How does Maven handle dependencies? Describe the process of adding and managing dependencies in a Maven project.

Maven manages dependencies through pom.xml. Steps to add dependencies: 1. Add <dependency> tag with artifact coordinates (groupId, artifactId, version). 2. Maven downloads dependencies and their transitive dependencies from repositories. 3. Dependencies are managed centrally, ensuring version compatibility and reducing conflicts. Use mvn clean install to download dependencies and build project. Maven resolves dependencies based on <dependencyManagement> and <dependencies> sections in pom.xml.

14. What are Maven plugins? How do you configure and use plugins in a Maven project?

Maven plugins provide additional functionality like compiling code, running tests, packaging applications, etc. Configure plugins in pom.xml using <plugin> tags with <groupId>, <artifactId>, and <version>. Plugins can be bound to phases of Maven lifecycle (compile, test, package) or executed standalone (mvn plugin:goal). Use mvn clean install to execute plugins configured in pom.xml and achieve project-specific tasks like code generation or static analysis.

15. How do you integrate Maven with Eclipse IDE? Describe the steps to create and manage Maven projects in Eclipse.

Integrate Maven with Eclipse:

1. Install Maven plugin (m2eclipse) in Eclipse.

2. Create a new Maven project (File > New > Maven Project).

3. Select archetype (project template) and configure pom.xml.

4. Import existing Maven projects (File > Import > Existing Maven Projects).

5. Manage dependencies, build lifecycle, and run Maven goals (clean, install) from Eclipse using Run As and Maven options. Eclipse provides integration with Maven for seamless project management and development.

16. What is pom.xml in Maven? Describe its structure and the essential elements it contains.

pom.xml is the Project Object Model configuration file in Maven. Structure:

1. <project>: Root element.

2. <modelVersion>: Maven version.

3. <groupId>, <artifactId>, <version>: Project coordinates.

4. <dependencies>: Project dependencies.

5. <build>: Build configuration (plugins, resources, testResources).

6. <repositories>: Dependency repositories, pom.xml defines project metadata, dependencies, plugins, build configurations, and repository locations for Maven projects.

17. Explain the Maven build lifecycle phases and their sequence. How can you customize build lifecycle in Maven?
Maven build lifecycle consists of phases (validate, compile, test, package, install, deploy) executed sequentially. Customize lifecycle using plugins (<plugin>), goals (<execution>), and lifecycle bindings (<phase>). Define custom phases and execute specific goals (mvn phase) or skip phases (-DskipTests). Maven plugins and configurations in pom.xml control build process, dependencies resolution, testing, packaging, and deployment lifecycle phases.

18. How do you integrate JDBC drivers with Maven? Provide an example configuration in pom.xml for MySQL and Oracle JDBC drivers.

Integrate JDBC drivers in Maven pom.xml:

1. Add <dependency> for MySQL (mysql-connector-java) or Oracle (ojdbc) JDBC driver.

2. Specify <groupId>, <artifactId>, and <version> for driver dependency.

Example configurations:

<dependency> for MySQL:

<groupId>mysql</groupId>,

<artifactId>mysql-connector-java</artifactId>,

<version>8.0.26</version>.

<dependency>

for Oracle:

<groupId>com. oracle. database.jdbc</groupId>,

<artifactId>ojdbc8</artifactId>,

<version>19.8.0.0</version>.

19. Describe the JDBC architecture with a detailed explanation of each component.

JDBC architecture:

1. JDBC API: Interfaces and classes for database access (Connection, Statement, ResultSet).

2. JDBC Driver Manager: Manages JDBC drivers and establishes connections (DriverManager).

3. JDBC Drivers: Implementations translating JDBC calls to database-specific protocols (JDBC-ODBC, Native-API, Network Protocol, Thin). 4. Database: Target data repository (Oracle, MySQL). JDBC API and drivers facilitate database connection, SQL execution, and result processing in Java applications.

20. How can Maven be used for deployment? Discuss strategies for deploying Maven-built artifacts to servers or repositories.

Deploy Maven artifacts:

1. Build artifact (mvn clean install) generates target files (JAR, WAR).

2. Deploy to local repository (~/.m2/repository) using mvn deploy.

3. Publish to remote repository (Nexus, Artifactory) for shared access. Deployment strategies: Automated CI/CD pipelines integrate Maven (Jenkins, GitLab CI) for continuous integration, testing, and deployment. Maven plugins (maven-deploy-plugin) configure artifact publishing and version management for centralized artifact storage and distribution.

21. What is RowSet interface and explain its types?

The RowSet interface in Java is part of the JDBC (Java Database Connectivity) API, which provides a higher-level abstraction over the traditional ResultSet interface. It was introduced to simplify the process of working with database data in Java applications.The RowSet interface in Java is part of the JDBC (Java Database Connectivity) API, which

provides a higher-level abstraction over the traditional ResultSet interface. It was introduced to simplify the process of working with database data in Java applications.

Common Characteristics of RowSet Interfaces:

Disconnected Operation: Except for JdbcRowSet, all other types are disconnected from the database after fetching data, which improves performance and reduces resource usage.

Serializable: All types can be serialized and used in distributed applications or stored in session state.

Cursor Movement: They all support cursor movement similar to ResultSet, allowing navigation through rows.

Update Capabilities: With the exception of JdbcRowSet, all others can be updated while disconnected and later synchronized with the database.

Types of RowSet Interfaces:

1. JdbcRowSet:

   Description: This is a connected rowset, meaning it maintains a connection to the database throughout its lifecycle.

2. CachedRowSet:

   Description: This rowset is disconnected, meaning it fetches data from the database and then closes the connection, allowing it to operate independently of the database.

3. WebRowSet:

   Description: Extends CachedRowSet and adds additional features for use in web applications.

4. FilteredRowSet:

   Description: Implements the FilteredRowSet interface, allowing rows to be filtered programmatically based on specific criteria.

Unit - 2 (Servlet Programming)

MCQ Questions

1. What is the first method called when a servlet is initialized?

   A. service))
   B. init()
   C. doGet))
   D. doPost))

2. Which method is called by the servlet container to handle a client request?

   A. init()
   B. service))
   C. doGet))
   D. destroy))

3. Which method is called when a servlet is taken out of service?

   A. init()
   B. service))
   C. destroy))
   D. doGet))

4. Which method is called when a servlet is taken out of service?

   A. init()
   B. service))
   C. destroy))
   D. doGet))

5. What is the purpose of the doGet)) and doPost)) methods in a servlet?

   A. To initialize the servlet
   B. To destroy the servlet
   C. To handle HTTP GET and POST requests, respectively
   D. To configure the servlet

6. When is the init() method of a servlet called?

   A. Every time a request is made to the servlet
   B. When the servlet is first loaded into memory
   C. When the servlet is destroyed
   D. When the servlet is reloaded

7. Which class must a Java class extend to become an HTTP Servlet?

   A. java.servlet.HttpServlet
   B. javax.servlet.HttpServlet
   C. javax.servlet.GenericServlet
   D. java.servlet.GenericServlet

8. Which of the following methods is used to handle HTTP POST requests in an HTTP Servlet?

   A. doGet))
   B. doPost))
   C. service))
   D. init()

9. Which of the following methods in an HTTP Servlet handles HTTP DELETE requests?

   A. doDelete))

B. doPost()
C. doGet()
D. doRemove()

10. How do you set the content type of the response to "text/html" in an HTTP Servlet?

A. response.setHeader("Content-Type", "text/html");
B. response.setContentType("text/html");
C. response.setContentC'text/html");
D. response.setType("text/html");

11. What method is used to retrieve a parameter sent in an HTTP request in a servlet?

A. request.getParameter(String name)
B. request.getAttribute(String name)
C. request.getProperty(String name)
D. request.getValue(String name)

12. Which tag is used to map a URL pattern to a servlet in the web.xml file?

A. <url-mapping>
B. <servlet-mapping>
C. <mapping>
D. <url-pattern>

13. Where do you specify initialization parameters for a servlet in the web.xml file?

A. Inside the <servlet> tag using <param>
B. Inside the <servlet-mapping> tag using <param>
C. Inside the <servlet> tag using <init-param>
D. Inside the <servlet-mapping> tag using <init-param>

14. How do you retrieve an initialization parameter in a servlet?

A. config.getlnitParameterf'paramName")
B. context.getlnitParameter("paramName")
C. request.getParameterf'paramName")
D. response.getlnitParameter("paramName")

15. What is the purpose of the <load-on-startup> element in the web.xml file?

A. To specify the order in which servlets should be loaded
B. To delay the loading of servlets until they are first requested
C. To load the servlet immediately after the server starts
D. To configure the servlet's URL pattern

16. Which of the following methods is used to track sessions using URL rewriting?

A. response.encodeURL(String url)
B. response.rewriteURL(String url)
C. response.trackURL(String url)
D. response.modifyURL(String url)

17. What is the default timeout period for an HTTP session in minutes, if not configured?

A. 15 minutes
B. 30 minutes
C. 60 minutes
D. 120 minutes

18. Which method is used to invalidate a session?

A. session.destroy))
B. session.terminate()
C. session.invalidate))
D. session.end()

19. How can you set an attribute in an HTTP session?

A. session.putAttribute(String name, Object value)
B. session.addAttributefString name, Object value)
C. session.setAttribute(String name, Object value)
D. session.storeAttribute(String name, Object value)

20. Which of the following is NOT a method for session tracking in servlets?

A. Cookies
B. URL Rewriting
C. Hidden Form Fields
D. Page Refreshing

Unit - 2 (Servlet Programming)

Fill in the blanks

1) The **InitQ** method is called by the servlet container to initialize a servlet.
2) The **serviced** method is called by the servlet container to handle a client request.
3) The **destroyQ** method is called by the servlet container to remove a servlet from service.
4) The **HttpServlet** class provides methods to handle HTTP-specific services.
5) The doGet() method of HttpServlet is used to handle HTTP GET requests.
6) The **doPostQ** method of HttpServlet is used to handle HTTP POST requests.
7) A **ServletContextQ** object provides information about the servlet's environment.
8) The **<servlet>** element in web.xml is used to configure a servlet.
9) A **ServletConfig** object contains initialization parameters for a servlet.
10) A **RequestDispather** object can be used to forward a request from one servlet to another.
11) The **getParameterQ** method of HttpServletRequest retrieves the value of a request parameter.
12) A **HttpSession** is used to track sessions in servlets.
13) The **setAttribute()** method of HttpSession stores an attribute in the session.
14) The **removeAttributeQ** method of HttpSession removes an attribute from the session.
15) The **sendRedirectQ** method of HttpServletResponse sends a redirect response to the client.
16) In a CRUD operation, the POSTQ method is typically used to create a new resource.
17) In a CRUD operation, the GETQ method is typically used to read a resource.
18) In a CRUD operation, the **PUTQ** method is typically used to update a resource.
19) In a CRUD operation, the **DELETEQ** method is typically used to delete a resource.
20) The **getCookiesQ** method of HttpServletRequest returns an array of cookies sent by the client.

Unit - 2 (Servlet Programming)

Short Questions

1. What is HTTP and what are its common methods used in web development?

HTTP (Hypertext Transfer Protocol) is the protocol used for transmitting web pages over the internet. Common HTTP methods include:

- GET: Requests data from a server (e.g., fetching a webpage).

- POST: Submits data to be processed to a server (e.g., form submission).

- PUT: Updates existing resources on the server.

- DELETE: Deletes resources from the server.

- HEAD: Similar to GET, but it retrieves only the headers, not the body of the response.

- OPTIONS: Describes the communication options for the target resource.

2. What is a web server, and how does it work?

A web server is software that serves web pages to clients over the HTTP protocol. It processes incoming requests from clients, retrieves the requested resources (such as HTML pages, images, or data), and sends them back as HTTP responses. Common web servers include Apache HTTP Server, Nginx, and Microsoft IIS.

3. Explain the difference between a web server and an application server.

A web server handles HTTP requests and serves static content (HTML, CSS, images). An application server, on the other hand, provides business logic to application programs through various protocols, including HTTP. Application servers can serve dynamic content, manage transactions, and provide middleware services. Examples include Apache Tomcat (which can also serve as a web server) and JBoss.

4. Describe the servlet lifecycle in detail.

The servlet lifecycle includes the following stages:

1. Loading and Instantiation: The servlet container loads the servlet class and creates an instance.

2. Initialization ('init()' method): Called once when the servlet is first loaded. It is used for one-time setup like resource allocation.

3. Request Handling ('service))' method): Called for each client request. It determines the request type (GET, POST, etc.) and dispatches it to the appropriate method ('doGet()', 'doPost()', etc.).

4. Destruction ('destroyO' method): Called once when the servlet is being taken out of service. It is used for cleanup activities like releasing resources.

5. What is the purpose of the 'init()' method in a servlet?

The 'init()' method is called by the servlet container to initialize the servlet. It is executed only once when the servlet is first loaded into memory. This method is used for one-time initialization tasks such as reading configuration parameters, setting up database connections, or preparing resources needed by the servlet during its lifecycle.

6. What is the difference between init()' and 'serviceQ' methods in a servlet?

A: The ' init()' method is called once during the servlet's lifecycle, when the servlet is first loaded and initialized by the servlet container. It is used for one-time setup and initialization tasks, such as loading configuration parameters or establishing database connections.

The 'service))' method is called for each client request to the servlet. It determines the type of request (GET, POST, etc.) and dispatches it to the appropriate method ('doGet()', 'doPostQ', etc.) for handling.

Servlets API

7. What are the main classes and interfaces in the Servlet API, and what are their purposes?

Key classes and interfaces in the Servlet API include:

- 'Servlet' interface: Defines the basic methods a servlet must implement ('init', 'service', 'destroy', 'getServletConfig', 'getServletInfo').

- 'GenericServlet' class: An abstract class that implements the 'Servlet' interface and is protocol-independent.

- 'HttpServlet' class: Extends 'GenericServlet' to provide methods specific to handling HTTP requests ('doGet', 'doPost', etc.).

- 'ServletConfig' interface: Used to pass configuration information to a servlet.

- 'ServletContext' interface: Provides methods for interacting with the servlet container, sharing information between servlets, and accessing resources.

- 'HttpServletRequest' and 'HttpServletResponse' interfaces: Provide methods to access request data and construct responses, respectively.

8. Explain the 'HttpServlet' class and its role in servlet development.

The 'HttpServlet' class is an abstract class provided by the Servlet API that extends 'GenericServlet' and implements the 'Servlet' interface. It provides methods to handle HTTP-specific services. Subclasses of 'HttpServlet' override methods like 'doGet()', 'doPost()', 'doPut()', and 'doDelete()' to handle HTTP GET, POST, PUT, and DELETE requests, respectively. This class simplifies the development of HTTP-based web applications by providing a framework for handling common HTTP request types.

HTTP Servlets

9. What is the 'doGet()' method in an HttpServlet', and when should it be used?

The 'doGet()' method handles HTTP GET requests. It is used when the client requests data from the server, typically to retrieve a web page or query parameters. GET requests should be idempotent and safe, meaning they do not modify the server state and can be repeated without side effects, protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { response.setContentType("text/html"); Printwriter out = response.getWriter(); out.println("<htmlxbody>"); out.println("<hl>Hello, World!</hl>"); out.println("</bodyx/html>");}

10. Describe the 'doPostQ' method in an 'HttpServlet' and its use cases.

The 'doPost()' method handles HTTP POST requests, which are used to send data to the server, typically from a form submission. POST requests can change the server state and are not idempotent. They are used for operations like creating or updating resources, processing forms, and uploading files, protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { String username = request.getParameter("username"); String password = request.getParameter("password"); // Process the data response.setContentType("text/html"); Printwriter out = response.getWriter(); out.println("<htmlxbody>"); out.println("<hl>Welcome," + username + "!</hl>"); out.println("</bodyx/html>");}

11. How can servlets be configured using the web.xml' file? Provide an example.

Servlets can be configured using the 'web.xml' deployment descriptor file by defining servlets and their mappings. Here is an example: <web-app> <servlet> <servlet-name>ExampleServlet</servlet-name> <servlet-class>com.example.ExampleServlet</servlet-class> </servlet>

<servlet-mapping> <servlet-name>ExampleServlet</servlet-name> <url-pattern>/example</url-pattern> </servlet-mapping> </web-app>

This configuration maps the 'ExampleServlet' class to the URL pattern '/example'.

12. What are servlet initialization parameters, and how are they defined in 'web.xml'?

A: Servlet initialization parameters are configuration parameters passed to a servlet during initialization. They are defined in the 'web.xml' file within the '<init-param>' tags. Here is an example: <servlet> <servlet-name>ExampleServlet</servlet-name> <servlet-class>com. example. ExampleServlet</servlet-class> <init-param> <param-name>paraml</param-name> <param-value>valuel</param-value> </init-param> </servlet>

The servlet can retrieve these parameters using the 'getlnitParameter' method: public void init(ServletConfig config) throws ServletException {super.init(config); String paraml = config.getlnitParameter("paraml");}

Servlets Context

13. What is the 'ServletContext' interface, and how is it used? Provide examples.

A: The 'ServletContext' interface provides methods for servlets to interact with the web application environment. It allows servlets to:

- Retrieve application-wide parameters: String paramValue = getServletContext().getlnitParameter("globalParam");

- Set and get attributes for sharing data between servlets: getServletContext().setAttribute("sharedData", data); Object data = getServletContext().getAttribute("sharedData");

Access resources like files or request dispatchers: Inputstream input = getServletContext().getResourceAsStream("/WEB-INF/config. properties"); RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/anotherServlet"); dispatcher.forward(request, response);

14. How can servlets log messages using ServletContext ?

A: Servlets can log messages using the 'log()' method of the 'ServletContext' interface. This method writes log messages to the servlet container's log file, which helps in debugging and monitoring. Here are examples: getServletContext().log("This is an informational message."); getServletContext().log("This is an error message.", new Exceptionf'Sample Exception"));

The first method logs a simple message, while the second logs a message with an exception.

15. What is servlet collaboration, and how is it achieved?

A: Servlet collaboration refers to the mechanism by which servlets communicate and share data with each other. It can be achieved using:

- Request Dispatcher: Forwards a request from one servlet to another resource within the same application. RequestDispatcher dispatcher = request.getRequestDispatcher("/anotherServlet"); dispatcher.forward(request, response);

- Shared Attributes: Stores and retrieves shared data using request, session, or context attributes.

request.setAttribute("key", "value"); Object value = request. getAttributefkey");

- Redirects: Sends a client-side redirect to another servlet or resource.

response.sendRedirect("anotherServlet");

Servlets Collaboration (continued)

16. Explain the RequestDispatcher' interface and its methods.

RequestDispatcher dispatcher = request.getRequestDispatcherf'/header.jsp");

dispatcher.include(request, response);

This method includes the content of 'header.jsp' in the current response.

17. What is session tracking, and why is it important in web applications?

Session tracking is a mechanism used to maintain state and data for a user across multiple requests in a web application. It is important because HTTP is a stateless protocol, meaning it does not retain any information about previous requests. Session tracking allows web applications to recognize users, store user preferences, manage login information, and maintain the state of a user's interaction with the application.

18. Explain the different techniques used for session tracking.

Techniques for session tracking include:

- Cookies:

Small pieces of data stored on the client-side.

They are sent with every request to the server.

Cookie userCookie = new Cookie("username", "john");

userCookie.setMaxAge(60*60*24); // 1 day response.addCookie(userCookie);

- URL Rewriting:

Appends session information to the URL.

String url = response.encodeURL("nextPage.jsp");

- Hidden Form Fields:

Includes session data in hidden fields of HTML forms. <input type="hidden" name="sessionID" value="12345">

- HttpSession API: Provides a more robust and easier way to manage sessions. HttpSession session = request.getSession(); session.setAttribute("username", "john");

19. Describe how the HttpSession' interface is used to manage sessions in servlets.

The 'HttpSession' interface provides methods to create and manage sessions. It allows servlets to store and retrieve user-specific data across multiple requests.

- Creating/Retrieving a Session: HttpSession session = request.getSession(); //true by default

- Storing Data in a Session: session.setAttribute("username", "john");

- Retrieving Data from a Session: String username = (String) session.getAttribute("username");

- Invalidating a Session:

session.invalidate();

20. How can you handle exceptions in servlets?

A: Exceptions in servlets can be handled using:

- 'try-catch' blocks: Surrounding code that might throw exceptions with 'try-catch' blocks.

- 'error-page' configuration in 'web.xml': Configuring custom error pages for specific exceptions or status codes.

<error-page> <error-code>404</error-code> <location>/error404.html</location> </error-page> <error-page> <exception-type>java.lang.Throwable</exception-type> <location>/error.jsp</location> </error-page>

- Logging: Using logging mechanisms to log the details of exceptions.

- 'HttpServletResponse.sendError': Sending an HTTP error response.

response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR, "An internal error

Unit - 2 (Servlet Programming)

Long Questions

1. What is Servlet ? Explain advantages of servlet ?

- Servlets is a java-based web technology from Sun Microsystems. Servlet technology is J2EE technology.
- A servlet is a user-defined public java Class that implements javax.servlet.Servlet interface and it is managed by the servlet container (is also called Servlet Engine).
- Properties of servlet are:
    - o A servlet is a Dynamic Web Resource that enhances the functionality of the webserver,
    - o A servlet is a Web Server-Side Piece of Code (server-side program).
    - o A servlet is a Web Component.
- Servlets use the classes within the java packages javax.servlet and javax.servlet.http.
- For each client request servlet container will generate a separate thread on the respective servlet object. If we increase the number of requests even containers will create a number of threads instead of processes so it increase the performance of the server-side application.

Advantages of Servlet.

1. Portability: Servlets are highly portable across operating systems and server implementations because servers are written in java and follow well known standardized APIs so they are.
2. Powerful: It is possible to do several things with the servlets which were difficult or even impossible to do with CGI.
3. Efficiency: As compared to CGI the invocation of the servlet is highly efficient. When the servlet gets loaded in the server, it remains in the server's memory as a single object instance.
4. Safety: As servlets are written in java, servlets inherit the strong type safety of java language. Servlets are generally safe from memory management problems because of Java's automatic garbage collection and a lack of pointers.
5. Integration: Servlets are tightly integrated with the server. Servlet basically uses the server to translate the file paths, perform logging, check authorization, and MIME type mapping, etc.
6. Extensibility: The servlet API is designed in such a way that it can be easily extensible. As it stands today, the servlet API supports HTTP Servlets, but later it can be extended for another type of servlets.

2. Explain the life cycle of servlet with an example.

The Java Servlet Life cycle includes three stages right from its start to the end until the Garbage Collector clears it. These three stages are described below.

1. init()

2. service()

3. destroy()

1. init()

The init() is the germinating stage of any Java Servlet. When a URL specific to a particular servlet is triggered, the init() method is invoked.

Another scenario when the init() method gets invoked is when the servers are fired up. With every server starting up, the corresponding servlets also get started, and so does the init() method.

One important specialty of the init() method is the init() method only gets invoked once in the entire life cycle of the Servlet, and the init() method will not respond to any of the user's commands.

The init() method Syntax:

```
public void init() throws ServletException {

//init() method initializing }
```

2. service()

The service!) method is the heart of the life cycle of a Java Servlet. Right after the Servlet's initialization, it encounters the service requests from the client end.

The client may request various services like:

• GET

• PUT

• UPDATE

• DELETE

The service)) method takes responsibility to check the type of request received from the client and respond accordingly by generating a new thread or a set of threads per the requirement and implementing the operation through the following methods.

• doGet() for GET

• doPut() for PUT

• doUpdate() for UPDATE

• doDeleteQ for DELETE

The service!) method Syntax:

```
public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException {

}
```

3. destroy!)

Like the init() method, the destroy!) method is also called only once in the Java Servlet's entire life cycle. When the destroy!) method is called, the Servlet performs the cleanup activities like, Halting the current or background threads, Making a recovery list of any related data like cookies to Disk.

After that, the Servlet is badged, ready for the Garbage collector to have it cleared.

The destroy!) method Syntax:

```
public void destroy!) {
    //destroy!) method finalizing
}
```

3. What is a deployment descriptor? Give an example to map the servlet

• A web application's deployment descriptor maps the http request with the servlets.

• When the web server receives a request for the application, it uses the deployment descriptor to map the URL of the request to the code that ought to handle the request.

• The deployment descriptor should be named as web.xml. It resides in the application's WAR file under the WEB-INF/ directory.

• Root element of web.xml should be <web-app>. <servlet> element map a URL to a servlet using <servlet-mapping> element. To map a URL to a servlet, you declare the servlet with the <servlet> element, then define a mapping from a URL path to a servlet declaration with the <servlet-mapping> element.

• The <servlet> element declares the servlet class and a logical name used to refer to the servlet by other elements in the file.

• You can declare multiple servlets using the same class but name for each servlet must be unique across the deployment descriptor. The <servlet-mapping> element specifies a URL pattern and the name of a declared servlet to

use for requests whose URL matches the pattern. The URL pattern can use an asterisk (*) at the beginning or end of the pattern to indicate zero or more of any character.

• The standard does not support wildcards in the middle of a string, and does not allow multiple wildcards in one pattern. The pattern matches the full path of the URL, starting with and including the forward slash (/) following the domain name. The URL path cannot start with a period (.).

Structure of deployment descriptor file - web.xml

<servlet>

<servlet-name>Practicel</servlet-name>

<servlet-class>

com. parul.controller.se rvlet

</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Practicel</servlet-name>

<url-pattern>/ </url-pattern>

</servlet-mapping>

4. What is the difference between GET and POST method?

1. GET is a safe method (idempotent) where POST is non-idempotent method.
2. We can send limited data with GET method and it's sent in the header request URL whereas we can send large amount of data with POST because it's part of the body.
3. GET method is not secure because data is exposed in the URL and we can easily bookmark it and send similar request again, POST is secure because data is sent in request body and we can't bookmark it.
4. GET is the default HTTP method whereas we need to specify method as POST to send request with POST method.
5. Hyperlinks in a page uses GET method.

5. What is HttpServlet and how it is different from GenericServlet?

GenericServlet

GenericServlet is an abstract class that provides a generic, protocol-independent servlet. It is intended to handle any type of protocol such as HTTP, FTP, SMTP, etc., although it is commonly used with HTTP. Here are some key points about GenericServlet:

Protocol Independence: It is designed to be protocol-independent, meaning it can handle any type of protocol. This is achieved by not assuming any specific details about the request or response.

Methods: The key methods provided by GenericServlet are:

init(ServletConfig config): Initializes the servlet with configuration data.

service(ServletRequest req, ServletResponse res): Handles the client request.

destroy)): Cleans up resources before the servlet instance is removed.

Implementation Requirements: Subclasses of GenericServlet must override the service method to provide specific handling for the protocol they intend to support.

HttpServlet

HttpServlet is an abstract subclass of GenericServlet that specifically handles HTTP requests and responses. It extends the capabilities of GenericServlet to provide functionality tailored for web applications that communicate over HTTP. Here's how HttpServlet differs:

HTTP-Specific Functionality: HttpServlet is designed to handle HTTP-specific details such as HTTP methods (GET, POST, PUT, DELETE, etc.), HTTP headers, cookies, session management, etc.

Methods: In addition to the methods inherited from GenericServlet, HttpServlet adds:

doGet(HttpServletRequest req, HttpServletResponse res): Handles GET requests.

doPost(HttpServletRequest req, HttpServletResponse res): Handles POST requests.

doPut(HttpServletRequest req, HttpServletResponse res): Handles PUT requests.

doDelete(HttpServletRequest req, HttpServletResponse res): Handles DELETE requests.

doHead(HttpServletRequest req, HttpServletResponse res): Handles HEAD requests.

doOptions(HttpServletRequest req, HttpServletResponse res): Handles OPTIONS requests.

doTrace(HttpServletRequest req, HttpServletResponse res): Handles TRACE requests.

These methods are specific to HTTP and provide a way to process requests based on the HTTP method used.

Usage: HttpServlet is typically used for building web applications that handle HTTP requests and generate HTTP responses. It encapsulates common web application logic, such as handling form submissions, managing user sessions, and interacting with HTTP-specific headers and parameters.

Key Differences

Protocol Support: GenericServlet is protocol-independent, whereas HttpServlet is specifically tailored for HTTP.

Methods: HttpServlet adds HTTP-specific methods (doGet, doPost, etc.) on top of the methods provided by GenericServlet.

Purpose: GenericServlet is more generic and can be used for any protocol, whereas HttpServlet is specialized for handling HTTP requests and responses in web applications.

In summary, HttpServlet extends GenericServlet to provide specialized support for handling HTTP requests and responses, making it the preferred choice for developing servlets in web applications that operate over the HTTP protocol.

6. What are the advantages of Servlet over CGI?

Servlets provide better performance in terms of processing requests, better memory usage by using the advantage of multithreading (a new thread is created for each request, which is faster than allocating memory for a new object for each request, as in CGI).

Servlets, platform and system are independent. Thus, a web application written using servlets can be run in any servlet container that implements the standard and in any operating system.

Servlet usage improves program reliability, because the servlet container itself takes care of the servlet life cycle (and therefore memory leaks), security, and garbage collection.

Servlets are relatively easy to learn and maintain, so the developer only needs to care about the business logic of the application, and not the internal implementation of web technologies.

7. What is the process to implement doGet and doPost methods?

doGet():

doGet() methods are the service methods that is included in the servlets. This allow the servlet to handle the GET request that is being processed by the client. This overrides the method to support the GET request that automatically supports the HTTP head request that is given in the no body in the response and included in the header fields. It is being implemented as:

protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, java.io.lOException

It is used to read the request data and provide the response of it by putting the solution in the header. It uses the response's writer or the object stream object that provides the response to the client. This is safe to use and can be safely repeated in case of any other request..

doPost():

doPostQ is used to handle a POST request that is also given at the time of filling up the form or any other action that is related to the user submission. This method allows the client to send the data of any length to the web server and that is also at single time. To read the request data the response headers are included that takes the response writer class to write that uses the output stream object. It uses the function of Printwriter object that returns the response to set the content type of accessing the object. It is given as:

doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, java.io.lOException"

What is a web application and what is it's directory structure in Servlet? "Web Applications are modules that requires to run on server to handle the request and return the response. Java provides web application support through Servlets and JSPs that can run in a servlet container and provide dynamic content to client browser.

Java Web Applications are packaged as Web Archive (WAR) and it has following folders in it.

WEB-INF : contains the lib directory(application dependencies), classes(contains java servlet files) and web.xml(Deployment Descriptor)

META-INF: contains MANIFEST.MF

webapp: contains all the static pages like .html, .js etc"

8. What is Servlet API? Explain the Servlet API packages

The Java Servlet API is a class library for Servlets. Servlets are Java programs that run on the server and send response to the client. Two packages are available: javax.servlet and javax.servlet.http.

The first one contains basic classes and interfaces that we can use to write Servlets from the scratch. The second package offers more advanced classes and interfaces that extend classes and interfaces from the first package. It is much more convenient to program using the second package.

The Servlet API packages are listed below.

Packages

javax.servlet

The javax.servlet package contains a number of classes and interfaces that describe and define the contracts between a servlet class and the runtime environment provided for an instance of such a class by a conforming servlet container,

javax.servlet.http

The javax.servlet.http package contains a number of classes and interfaces that describe and define the contracts between a servlet class running under the HTTP protocol and the runtime environment provided for an instance of such a class by a conforming servlet container.

9. What is the ServletConfig object? Explain with an example

• ServletConfig is an object, it will store all the configuration details of a specific servlet, where the configuration details include the logical name of the servlet, initialization parameters, reference of ServletContext object, and so on.

• ServletConfig is an object, it will provide the entire view of a specific servlet. In the web application, the container will prepare ServletConfig objects individually to every and each servlet.

• In web application execution, the container will prepare ServletCofig object immediately after servlet instantiation and just before calling init() method in servlet initialization.

• The container will destroy the ServletConfig object just before servlet de-instantiation.

• If we declare any data in the ServletConfig object then that data is going to be shared up to the respective servlet. Due to the above reasons, the scope of the ServletConfig object is up to a particular servlet.

```
<web-app>
 <servlet>
   <servlet-name>firstservlet</servlet-name>
   <servlet-class>com.teknobizz.controller</servlet-class>
   <init-param>
     <param-name> username </param-name>
     <param-value> jones@aol.com </param-value>
   </init-param>
   <init-param>
     <param-name> fullname </param-name>
     <param-value> Mark Jones </param-value>
   </init-param>
 </servlet>
 <servlet-mapping>
   <servlet-name>firstservlet</servlet-name>
   <url-pattern>/</url-pattern>
 </servlet-mapping>
 <welcome-file-list>
   <welcome-file>index.html</welcome-file>
 </welcome-file-list>
</web-app>
protected void doPost(HttpServletRequest req,HttpServletResponse res) throws ServletExceptionJOException {
        Printwriter pw=res.getWriter();
      res.setContentType(""text/html"");

      ServletConfig conf = getServletConfig();
      String username = conf.getInitParameter(""username"");
```

```
        String fullname = conf.getInitParameter(""fullanme"");
        pw.println(""Username + username + "<br/>Fullname + fullname);
        pw.close();
}
```

10. What is Servlet Context ? Explain with an example

• It will manage all the context details of a particular web application, where the context details include the logical name of the web application and context parameters, and so on.

• It will provide a complete view of a particular web application. ServletContext object will be prepared by the container the moment when we start the server i.e. the time when we deploy the web application.

• It can be created and removed only by the server, not by the programmer.

• ServletContext object is created at the time of deploying the project.

• ServletContext object will be removed by the server when we un-deploy the project.

• ServletContext object will be destroyed by the container when we shut down the server i.e. the time when we un-deploy the web application. Due to the above reasons, the life of the ServletContext object is almost all the life of the respective web application.

• If we declare any data in the ServletContext object then that data will be shared with all the number of resources that are available in the present web application. Due to the above reason, ServletContext will provide more shareability.

• In web applications, the container will prepare ServletContext object irrespective of getting requests from the client.

• In the web application, ServletContext will allow both parameters and attributes data. Due to the above reason, ServletContext will allow both Static Inclusion and Dynamic Inclusion of data.

Web.xml

```
<web-app>
   <context-param>
      <param-name> numl </param-name>
      <param-value> 100 </param-value>
   </context-param>
   <context-param>
      <param-name> num2 </param-name>
      <param-value> 200 </param-value>
   </context-param>
 <servlet>
    <servlet-name>ctxservlet</servlet-name>
    <servlet-class>com.teknobizz.controller</servlet-class>
 </servlet>
 <servlet-mapping>
    <servlet-name> ctxservlet </servlet-name>
    <url-pattern>/</url-pattern>
```

```
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
protected void doPost(HttpServletRequest req,HttpServletResponse res)throws ServletExceptionJOException
  {
     Printwriter pw=res.getWriter();
     res.setContentType(""text/htmr");
     ServletContext context = getServletContext();
     String numl = context.getlnitParameter(""numl"");
     String num2 = context.getlnitParameter(""num2"");
     pw.println(""numl value is + numl+ "" and num2 is + num2);
    pw.closeQ;
  }
```

11. What is the difference between Printwriter and ServletOutputStream?

Printwriter is a class for working with a character stream, and ServletOutputStream is a class for working as a byte stream. Printwriter is used to write character-based information, such as an array of characters or a string in the response, while ServletOutputStream is used to write a byte array to the response.

To get an instance of ServletOutputStream, use the ServletResponse getOutputStream () method , and for Printwriter , use the ServletResponse getWriter () method

12. What is Servlet container ? What tasks does a servlet container perform usually?

Servlet containers are part of a web server that offers network services. They depend on the MIME-based requests and responses. A servlet container handles servlets.

A servlet container performs the following tasks:

It facilitates communication between the servlets, JSPs and the web client. You don't have to build a server socket to receive requests, parse them and generate responses because of the container. The container takes care of these tasks, allowing you to focus on the business logic.

The servlet container handles the life cycle of servlets. It loads the servlets into memory, initialises them, invokes the necessary methods and destroys them. Servlet containers also simplify resource management by offering utilities such asJNDI.

Servlet containers create new threads for every request and give servlets request and response objects. This way, you don't have to initialise the servlets for every request, saving a lot of memory and time."

13. Differences between ServletConfig and ServletContext

ServletConfig	ServletContext
Defined in javax.servlet package.	Defined in the same javax.servlet package.
Values are placed in web.xml file.	Values are placed in web.xml file.
getInitParameter() defined ServletConfig is used to read values.	Same method getInitParameter() is also present in ServletContext interface also to read the values
When the values of <param-value> are changed, the Servlet need not be compiled again and thereby maintenance of code is easier.	The same case also with <context-param>.
No limit on the existence of <init-param> tags in <servlet>.	the web.xml fiel can have any number of <context-param> tags.
Servlet is initialized with the initialization data provided in <init-param>. This data is specific for one Servlet only.	The data provided in <context-param> tag is meant for the usage of all Servlets under execution.
Data in included within <servlet> tag.	Data is included within <context-param> tag.
It is local data for a particular Servlet.	It is global data sharable by all servlets.
Each Servlet comes with a separate ServletConfig object.	There will be only one ServletContext object available accessed by all Servlets.

14. What is RequestDispatcher ? explain forward() and include method() with an example

The RequestDispatcher is an Interface that comes under package javax.servlet. Using this interface we get an object in servlet after receiving the request. Using the RequestDispatcher object we send a request to other resources which include (servlet, HTML file, or JSP file). A RequestDispatcher object can be used to forward a request to the resource or to include the resource in a response. The resource can be dynamic or static.

There are two methods defined in the RequestDispatcher interface.

public void forward(ServletRequest request,ServletResponse response) throws ServletException,java.io.IOException

Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.

public void include(ServletRequest request,ServletResponse response) throws ServletException,java.io.IOException

Includes the content of a resource (servlet, JSP page, or HTML file) in the response.

The forwardf) method is used to transfer the client request to another resource (HTML file, servlet, jsp etc). When this method is called, the control is transferred to the next resource called. On the other hand, the include() method is used to include the content of the calling file into the called file. After calling this method, the control remains with the calling resource, but the processed output is included into the called resource.

Example

```
RequestDispatcher rd = request.getRequestDispatcher(""RequestDispacherExl_l"");

    rd.forwa rd (request, response);

    RequestDispatcher rd = request.getRequestDispatcher(""RequestDispacherEx2_l"");

    rd.include(request, response);"
```

15. What is the difference between sendRedirect() and Forward)) in a Servlet?

The difference between sendRedirect)) and Forward() can be explained as follows:

sendRedirect)):

sendRedirect() method is declared in HttpServletResponse Interface.

The syntax for the function is as follows:

1

void sendRedirect(String URL)

This method redirects the client request for further processing, the new location is available on a different server or different context. The web container handles this and transfers the request using the browser, this request is visible in the browser in the form of a new request. It is also called a client-side redirect.

Forward)):

Forward)) method is declared in the RequestDispatcher Interface.

The syntax for the function is as follows:

1

forward(ServletRequest request, ServletResponse response)

This passes the request to another resource for processing within the same server, another resource could be any servlet, JSP page. Web container handles the Forward)) method. When we call Forward)) method, a request is sent to another resource without informing the client, about the resource that will handle the request. It will be mentioned on the requestDispatcher object which we can be got in two ways. Either using ServletContext or Request."

17. What is session tracking ? Explain the various techniques with an example

Session Tracking is a way to maintain state (data) of an user. It is also known as session management in servlet.

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.There are four techniques used in Session tracking:

1. Cookies
2. Hidden Form Field
3. URL Rewriting
4. HttpSession

Cookie

A cookie is a small piece of information that is persisted between the multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.

There are 2 types of cookies in servlets.

Non-persistent cookie

Persistent cookie

Non-persistent cookie : It is valid for single session only. It is removed each time when user closes the browser.

Persistent cookie : It is valid for multiple session . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

Hidden Field

The Server embeds new hidden Fields in every dynamically generated From page for the client, when the client submits the form to the server the hidden fields identify the client.

Hidden field is an invisible text box of the form page, hidden field value goes to the server as a request parameter when the form is submitted.

It always work whether cookie is disabled or not.

URL rewriting

URL-Rewriting Session Tracking Mechanism is almost all same as HttpSession Tracking Mechanism, in URL-Rewriting Session Tracking Mechanism we will not depending on a Cookie to maintain Session-Id value, we will manage SessionId value as an appended to URL in the next generated form.

In URL-Rewriting Session Tracking Mechanism, every time we need to rewrite the URL with Session-Id value in the next generated form. So that this mechanism is called a URL-Rewriting Session Tracking Mechanism.

HTTP Session

HttpSession interface enables a servlet to read and write the state information that is associated with an HTTP session.

A session contains information, specific to a particular user across the whole application. When a user enters into a website for the first time.

Session Management facility creates a unique session ID and typically sends it back to the browser as a cookie or store in request parameter.

Each subsequent request from this user passes the cookie containing the session ID, and the Session Management facility uses this ID to find the user's existing HttpSession object.

HttpSession is obtained via request.getSession(). The default timeout value is 30 minutes.

19. What are the different methods involved in generic servlet?

Generic servlet is a base class of servlet. It involves the following methods:

init() method: This method is called once by the servlet container in its lifecycle . It takes a ServletConfig object that contains the initialization parameters and configuration of the servlet.

Service)) method : This method is defined as a public void service as ServletRequest "req" OR ServletResponse "res" which gives Servlet Exception, as IOException .If once the servlet starts getting requests, the service)) method is called by the servlet container to respond to the requests.

Getservlet config)): The method comprises of the initialization and the startup of the servlet. It is also responsible for returning the Server Config object.

Getservlet info)): This method is defined as public String getServletInfo)) .It is responsible for returning a string which is in the form of plain text and not any kind of markup.

destroy)): This method is defined as public destroy().this method is called when want to close the servlet."

20. What is the difference between request attributes, session attributes, and ServletContext attributes?

A ServletContext attribute is an object bound into a context through ServletContext.setAttribute)) method and which is available to ALL servlets (thus JSP) in that context, or to other contexts via the getContext() method. By definition a context attribute exists locally in the VM where they were defined. So, they're unavailable on distributed applications.

Session attributes are bound to a session, as a mean to provide state to a set of related HTTP requests. Session attributes are available ONLY to those servlets which join the session. They're also unavailable to different JVMs in distributed scenarios. Objects can be notified when they're bound/unbound to the session implementing the HttpSessionBindingListener interface.

Request attributes are bound to a specific request object, and they last as far as the request is resolved or while it keep dispatched from servlet to servlet. They're used more as comunication channel between Servlets via the

RequestDispatcher Interface (since you can't add Parameters...) and by the container. Request attributes are very useful in web apps when you must provide setup information between information providers and the information presentation layer (a JSP) that is bound to a specific request and need not be available any longer, which usually happens with sessions without a rigorous control strategy.

Thus we can say that context attributes are meant for infra-structure such as shared connection pools, session attributes to contextual information such as user identification, and request attributes are meant to specific request info such as query results.

Unit - 3 (JSP Programming)

MCQ Questions

1. Which JSP scripting element is used to declare variables and methods?
   - **A.** <%! %>
   - B. <% °/o>
   - C. <%= %>
   - D. <%@ %>
2. Which JSP scripting element is used to embed Java expressions that are evaluated and inserted into the output
   - A. <%!%>
   - B. <%%>
   - **C. <%=%>**
   - D. <%@%>
3. Which JSP scripting element is used to include Java code that is executed when the JSP is requested?
   - A. <%!%>
   - **B. <%%>**
   - C. <%=%>
   - D. <%@%>
4. What is the purpose of the <%@ %> directive in JSP?
   - A. To declare variables and methods
   - B. To embed Java expressions
   - C. To include Java code
   - **D. To provide instructions to the JSP engine**
5. Which JSP directive is used to include another file in a JSP at page translation time?
   - **A. <%@ include file="filename" %>**
   - B. <% include file="filename" %>
   - C. <%= include file="filename" %>
   - D. <%! include file="filename" %>
6. Which JSP directive is used to specify page-related attributes, like content type and error page?
   - A. <%@ taglib %>
   - B. <%@ include %>
   - **C. <%@ page %>**
   - D. <%@ directive %>
7. Which directive is used to define the scripting language in a JSP page?
   - A. <%@ include %>
   - B. <%@ taglib %>
   - **C. <%@ page %>**
   - D. <%@ language %>
8. Which directive is used to include a file during the translation phase of the JSP?
   - **A. <%@ include file="filename" %>**
   - B. <% include file="filename"%>
   - C. <%@ file include-'filename" %>
   - D. <% file include="filename" %> 1

9. Which directive declares the content type of a JSP page?

   A. <%@ page contentType="type" %>
   B. <%@ page type="content" %>
   C. <%@ include contentType="type" %>
   D. <%@ taglib contentType="type" %>

10. How do you specify an error page in a JSP using directives?

   A. <%@ page errorPage="error.jsp" %>
   B. <%@ page error="error.jsp" %>
   C. <%@ include errorPage="error.jsp" %>
   D. <%@ taglib errorPage-'error.jsp" %>

11. Which directive is used to import classes, interfaces, or packages in a JSP page?

   A. <%@ import="java.util.*" %>
   B. <%@ page import="java.util.*" %>
   C. <%@ include import="java.util.*" %>
   D. <%@ taglib import-'java.util.*" %>

12. Which directive is used to specify that the current JSP page is an error page?

   A. <%@ page errorPage="true" %>
   B. <%@ page isErrorPage="true" %>
   C. <%@ include isErrorPage="true" %>
   D. <%@ taglib isErrorPage="true" %>

13. What is the correct syntax for the taglib directive in JSP?

   A. <%@ taglib uri="uri" prefix="prefix" %>
   B. <%@ taglib uri="prefix" prefix="uri" %>
   C. <%@ taglib prefix="uri" uri="prefix" %>
   D. <%@ taglib prefix="prefix" uri="uri" %>

14. Which page directive attribute is used to define a buffer size for the JSP page?

   A. buffer
   B. autoFlush
   C. contentType
   D. session

Unit - 3 (JSP Programming)

Fill in the Blanks

1. JSP stands for **Java Server Page**
2. Scripting elements in JSP allow embedding **Java Code** within HTML pages.
3. Directive elements in JSP provide instructions to the container about JSP **page compilation and execution**
4. CRUD operations typically involve **Creating, Reading, Updating, Deleting**
5. The <%@ page import-'..." %> directive is used in JSP for **Importing Java classes**
6. In JSP, <%!... %> denotes **declaration**
7. The <%= ... %> syntax in JSP is used for **Outputting data**
8. JSP pages are compiled into **Java bytecode**
9. The action element <jsp:useBean> in JSP is used for **Instantiating a JavaBean**
10. The <jsp:forward> action in JSP is used for **Redirecting to another resource**
11. The <jsp:include> action in JSP is used for **Including the output of another resource or file**
12. In JSP, the <%@ taglib %> directive is used for **Importing a tag library**
13. The request.getParameter("paramName") method in JSP is used to **Extract data from an HTTP request**
14. The method used to delete data from a database in JSP is **doDelete()**
15. JSP technology allows **Static HTML pages** to be generated dynamically.
16. The <%@ include file="..." %> directive in JSP is used for **Including the content of another file**
17. A JSP page is compiled into a(n) **Servlet**
18. The <c:forEach> tag in JSP is used for **Iterating over a collection**
19. The <form> element in HTML is used to **Collect user input** in JSP.
20. In JSP, the <jsp:scriptlet> tag is used for **Embedding Java code**

Unit - 3 (JSP Programming)

Short Questions

1. What is JSP?

JSP (JavaServer Pages) is a technology that allows developers to create dynamically generated web pages based on HTML, XML, or other document types.

2. How is a JSP page processed?

A JSP page is first translated into a servlet, compiled into bytecode, loaded and instantiated, and then executed by the servlet container.

3. What is the life cycle of a JSP page?

The life cycle includes translation, compilation, instantiation, initialization, execution, and destruction of the JSP page.

4. How do you create and execute a JSP page?

To create a JSP page, create a .jsp file containing HTML and JSP elements. Execute by deploying on a servlet container like Tomcat.

5. Discuss the directory structure of a JSP page.

Typical structure includes .jsp files in the webapp directory, with WEB-INF containing web.xml and other configuration files.

6. What is the JSP API?

The JSP API provides classes and interfaces to support the development of web applications based on JSP technology. Discuss the packages of the JSP API. Packages include javax.servlet.jsp for JSP-specific classes and javax.servlet.jsp.tagext for tag library support.

7. What is a scripting element in JSP?

A scripting element in JSP allows embedding Java code directly into the JSP page.

8. Discuss the types of scripting elements in JSP.

Types are:

include declaration (<%! %>) for variable and method declarations,

scriptlet (<% %>) for executable Java code, and

expression (<%= %>) for outputting data.

9. Provide an example of a declaration in JSP.

jsp
<%I int count = 0; %>

10. What are implicit objects in JSP?

Implicit objects are predefined objects accessible without explicit declaration in JSP pages.

11. Discuss the types of implicit objects in JSP.

Types include request, response, session, application, out, config, pageContext, exception, page, pageScope, param, header, cookie.

12. Explain the usage of the request implicit object in JSP with an example.

The request object provides access to client request information. Example: String name (String)request.getParameter("username");

13. Explain request and response methods in JSP with examples.

Request methods (getParameter(), getAttributef)) retrieve data sent by the client.

Response methods (setContentType(), getWriterQ) manage server responses.

14. Discuss JSP configuration

(<jsp-config>), application (<jsp:useBean>), session (<jsp:session>), out (<jsp:out>), context (<jsp:context>), and exception handling (<%@ page errorPage="error.jsp" %>) in JSP. These elements configure JSP behavior, manage application data, session attributes, output content, application-wide context, and handle exceptions.

15. What are JSP action tags?

JSP action tags are XML-based tags that provide server-side processing logic in JSP pages.

16. Provide an example of a JSP action tag.

<jsp:include page="header.jsp" />

17. What is expression language (EL) in JSP?

Expression Language (EL) simplifies accessing data stored in JavaBeans, session, request, and application scopes directly in JSP pages.

18. Discuss the usage of expression language (EL) in a JSP page with an example.

Example: ${user.name} accesses the name property of the user bean stored in session or request scope.

19. Discuss core JSTL tags and custom tags in JSP.

JSTL tags (<c:forEach>, <c:if>, <c:set>) provide common programming tasks, while custom tags extend JSP functionality with custom behaviors.

20. Discuss core JSTL tags and custom tags in JSP.

JSTL tags (<c:forEach>, <c:if>, <c:set>) provide common programming tasks, while custom tags extend JSP functionality with custom behaviors.

Unit - 3 (JSP Programming)

Long Questions

1. What is JSP and why do we need it? Discuss the benefits of using JSP over traditional Java servlets.

- JSP stands for Java Server Pages.
- JSP is java server-side technology to create dynamic web pages.
- JSP is extension of Servlet technology to help developers create dynamic pages with HTML like syntax.
- We can create user views in servlet also but the code will become very ugly and error-prone. Also, most of the elements on a web page are static, so the JSP page is more suitable for web pages.
- We should avoid business logic in JSP pages and try to use it only for view purpose.
- JSP scripting elements can be used for writing Java code in JSP pages but it's best to avoid them and use JSP action elements, JSTL tags or custom tags to achieve the same functionalities.
- One more benefit of JSP is that most of the containers support hot deployment of JSP pages. Just make the required changes in the JSP page and replace the old page with the updated jsp page in the deployment directory and the container will load the new JSP page.
- We don't need to compile our project code or restart server whereas if we make a change in servlet code, we need to build the complete project again and deploy it. Although most of the containers now provide hot deployment support for applications still it's more work that JSP pages.

Some benefits of using JSP over traditional Java servlets are:

1. Simplified development: JSP allows embedding Java code within HTML, reducing the need for explicit servlet code.
2. Rapid prototyping: JSP facilitates the quick and easy creation of dynamic web pages with its simple syntax.
3. Enhanced maintainability: JSP separates presentation logic from business logic, making it easier to update and maintain the codebase.
4. Reusability: JSP fragments, tag files, and custom tags promote code reuse and modular design.
5. Improved productivity: JSP provides a large set of built-in tags and libraries, such as JSTL, for common web development tasks.
6. Platform independence: JSP runs on any server that supports Java, providing portability and flexibility.

2. Explain the advantages and disadvantages of using JSP over other web technologies like Servlets or JavaScript frameworks?

JSP (JavaServer Pages) is a technology that allows developers to embed Java code within HTML pages to dynamically generate content on the server side. When comparing JSP to other web technologies like Servlets or JavaScript frameworks, there are several advantages and disadvantages to consider:

Advantages

1. Simplicity: JSP allows developers to mix Java code directly within HTML, making it easier to create dynamic web pages compared to writing Java Servlets, which require more code for similar functionality.
2. Reusable Components: JSP supports custom tag libraries, which can be used to create reusable components and simplify complex page structures.
3. Integration with Java EE: JSP is a part of the Java EE stack, making it easy to integrate with other Java EE technologies like Enterprise JavaBeans (EJB) and Java Persistence API (JPA).
4. Separation of Concerns: JSP promotes a separation of concerns by encouraging the use of JavaBeans for business logic and JSP pages for presentation. This separation makes it easier to maintain and update web applications.

Disadvantages

1. Performance: JSP pages can have performance overhead due to the need to compile them into Servlets at runtime. Compiled Servlets tend to perform better than JSP pages.
2. Mixing Logic with Presentation: While JSP encourages separation of concerns, it is still possible to mix business logic with presentation, leading to code that is hard to maintain and test.

3. Limited Front-End Functionality: JSP is primarily a server-side technology and lacks the rich front-end capabilities provided by modern JavaScript frameworks like React or Angular.
4. Steep Learning Curve: For developers who are new to Java and web development, JSP may have a steeper learning curve compared to simpler templating languages or JavaScript frameworks.

3. What are the JSP lifecycle phases? JSP lifecycle phases are:

1. Translation - JSP container checks the JSP page code and parse it to generate the servlet source code. For example in Tomcat you will find generated servlet class files at TOMCAT/work/Catalina/localhost/WEBAPP/org/apache/jsp directory. If the JSP page name is home.jsp, usually the generated servlet class name is homejsp and file name is homejsp.java
2. Compilation - JSP container compiles the jsp class source code and produce class file in this phase.
3. Class Loading - Container loads the class into memory in this phase.
4. Instantiation - Container invokes the no-args constructor of generated class to load it into memory and instantiate it.
5. Initialization - Container invokes the init method of JSP class object and initializes the servlet config with init params configured in deployment descriptor. After this phase, JSP is ready to handle client requests. Usually from translation to initialization of JSP happens when first request for JSP comes but we can configure it to be loaded and initialized at the time of deployment like servlets using load-on-startup element.
6. Request Processing - This is the longest lifecycle of JSP page and JSP page processes the client requests. The processing is multi-threaded and similar to servlets and for every request a new thread is spawned and ServletRequest and ServletResponse object is created and JSP service method is invoked.
7. Destroy - This is the last phase of JSP lifecycle where JSP class is unloaded from memory. Usually it happens when application is undeployed or the server is shut down.

4. What are JSP lifecycle methods?

JSP lifecycle methods are:

1. jsplnitf): This method is declared in JspPage and it's implemented by JSP container implementations. This method is called once in the JSP lifecycle to initialize it with config params configured in deployment descriptor. We can override this method using JSP declaration scripting element to initialize any resources that we want to use in JSP page.
2. JspServicef): This is the JSP method that gets invoked by JSP container for each client request by passing request and response object. Notice that method name starts with underscore to distinguish it from other lifecycle methods because we can't override this method. All the JSP code goes inside this method and it's overridden by default. We should not try to override it using JSP declaration scripting element. This method is defined in HttpJspPage interface.
3. jspDestroyQ: This method is called by container when JSP is unloaded from memory such as shutting down application or container. This method is called only once in JSP lifecycle and we should override this method to release any resources created in JSP init method

5. Differentiate between JSP and servlet.

1. JSP (JavaServer Pages) is an extension of servlet technology that allows the embedding of Java code within HTML pages, making it easier to create dynamic content. Servlets, on the other hand, are Java classes that handle HTTP requests and generate responses dynamically.
2. JSP pages are primarily used for presentation logic, while servlets are used for both handling requests and generating dynamic content.
3. JSP pages are translated into servlets during runtime, whereas servlets are Java classes that are compiled before execution.
4. JSP pages are more suitable for web page development, while servlets provide more flexibility and control over the request-handling process.

6. What are JSP directives? Explain with an example.

List the types of directives. JSP directives are instructions that provide directives to the JSP container on how to handle the JSP page during translation and execution. There are three types of JSP directives:

1. Page Directive: Specifies page-specific attributes, such as error handling, language, content type, session management, and more.
2. Include Directive: Includes the contents of an external file during translation.
3. Taglib Directive: Declares custom tag libraries and their prefixes for use on the JSP page.

7. What is a JSP expression? How is it different from a scriptlet?

A JSP expression is a piece of Java code that is enclosed within <%=%> tags in a JSP page. It is used to dynamically output a value or expression directly into the generated HTML response. The expression is evaluated at runtime, and the result is converted to a string and inserted into the HTML output. In contrast, a scriptlet (<% ... %>) allows you to embed arbitrary Java code within a JSP page. It can be used to perform more complex logic and computations, but its output is not directly written to the response. Instead, it can manipulate variables, call methods, or control the flow of the page.

8. Explain JSP scripting elements. ? Give an examples

JSP scripting elements allow you to embed Java code within a JSP page. There are three types of scripting elements:

1. Scriptlet: Enclosed within <% ... %> tags, it allows you to write Java code that is executed when the page is processed.
2. Declaration: Enclosed within <%! ... %> tags, it allows you to declare variables, methods, and other members that can be accessed throughout the JSP page.
3. Expression: Enclosed within <%= ... %> tags, it allows you to embed Java expressions whose values are converted to strings and outputted directly to the response.

9. What is the purpose of the page directive in a JSP page?

This is one of the other most asked JSP Interview questions. The page directive in a JSP (JavaServer Pages) page serves a critical role in providing instructions to the JSP container on how to process and manage the current page. It essentially acts as a configuration element at the top of the JSP file, allowing developers to set specific attributes that impact the behaviour and characteristics of the JSP page.

One of the primary purposes of the page directive is to define the content type that the JSP page will generate when it is rendered in the browser. This is done using the contentType attribute, and it specifies whether the JSP page will produce HTML, XML, JSON, or some other type of content. Setting the correct content type is crucial for ensuring that the browser interprets the page's output correctly.

Additionally, the page directive allows developers to specify the scripting language to be used within the JSP file, with options like Java or JavaScript. It also enables error handling by allowing the specification of an error page to be displayed in case an unhandled exception occurs during the execution of the JSP.

10. Differentiate between forward and redirect in JSP.

1. Forward: Forwarding is a server-side operation where the control is transferred from one JSP page to another JSP page or servlet within the same request. The browser is unaware of the forward, and the URL in the address bar remains the same. It allows sharing of request attributes and maintains a single request/response cycle.
2. Redirect: Redirecting is a client-side operation where the control is transferred to a different URL or resource. The server sends an HTTP redirect response to the browser, which then sends a new request to the redirected URL. The browser's address bar displays the redirected URL. Redirecting creates a new request/response cycle and does not share request attributes

11. What is the JSP implicit object? List some commonly used implicit objects.

JSP implicit objects are pre-defined objects provided by the JSP container that is available for use without explicitly declaring or instantiating them. Some commonly used implicit objects in JSP include:

- request: Represents the client's request to the server.
- response: Represents the server's response to the client.
- out: Used for writing output to the client.
- session: Represents the user's session.
- application: Represents the entire web application.
- pageContext: Provides access to various objects and information about the JSP page.
- config: Represents the configuration of the JSP page.
- exception: Represents any exception thrown during JSP page execution.

12. What is the significance of the JSP page directive isThreadSafe attribute, and when would you use it?

The isThreadSafe attribute in a JSP page directive allows you to specify whether a JSP page should be thread-safe or not. When set to true, it indicates that the JSP container should make the JSP page thread-safe by synchronising access to it, ensuring that multiple threads cannot concurrently execute the page.

This attribute is uncommonly used because most JSP pages are inherently thread-safe, as they follow a model where each request is handled by a separate thread, and JSP instances are not shared among requests. However, in rare cases where you might want to disable this automatic thread-safety mechanism, you can set isThreadSafe to false. This can be useful if you have specific optimization requirements and are confident that your JSP page does not have any shared state or critical sections that need synchronization.

It is worth noting that explicitly setting isThreadSafe to false should be done cautiously, as it can lead to issues with concurrency if not managed properly. In most cases, leaving it to the default value of true is recommended for safety and predictability.

13. Which categories can be divided JSTLtags, give examples.

JSTL tags are divided into five categories according to their functionality:

1. Core Tags - Core tags provide opportunities for iteration, exception handling, url, forward and redirect response, etc.
2. Formatting and Localization Tags - provide options for formatting Numbers, Dates, and support for il8n localization and resource bundles.
3. SQL Tags - JSTL SQL Tags support for working with databases like MySQL, Oracle, etc.
4. XML Tags - used to work with XML documents. For example, for parsing XML, transforming XML data and executing XPath expressions.
5. JSTL Functions Tags - provides a set of functions that allow you to perform various operations with strings, etc. For example, by concatenation or splitting strings.

14. What do you know about jsp expression language (JSP Expression Language - EL)?

In most cases, we use JSP for viewing purposes, and all business logic is present in the servlet or model classes. After receiving the client's request, it is processed in the servlet, and then we add attributes to the request/session/ context scope, which must be extracted in the JSP code. The request response , headers, cookies, and init parameters are also used to create response views .

We can use scriptlets in JSP expressions to get attributes and parameters in JSP with Java code and use it for views. The problem is that web designers usually do not know java code, which is why JSP 2.0 introduces an expression language (EL) through which we can get attributes and parameters using HTML tags.

The syntax expression language looks like $ {name}, and we can use EL implicit objects and expression language operators to extract attributes from different scopes and use them in the JSP page.

15. How do you handle JSP page redirection? Give an example of each JSP page redirection can be achieved in multiple ways:

1. Using the response.sendRedirect() method to redirect the client's browser to a different URL or JSP page.
2. Using the JSP <jsp:forward> action to forward the request to another resource or JSP page.
3. Utilizing JavaScript or HTML meta refresh tags for client-side redirection.
4. Handling redirection logic in Java servlets or controller classes based on business rules or user actions.

16. What is JSP standard tag library (JSTL)? How is it useful? What is the syntax for using JSTL core tags in JSP?

JSP Standard Tag Library (JSTL) is a collection of custom tags provided as part of the JavaServer Pages specification. It offers a set of tags for common tasks such as iteration, conditional execution, database access, and XML processing. JSTL simplifies JSP development by providing a standard way to perform these tasks, reducing the need for scriptlets and promoting a more modular and reusable code structure.

To use JSTL core tags, you need to include the JSTL core tag library in your JSP using the taglib directive: <%@ taglib prefix=""c"" uri=""http://java.sun.com/jsp/jstl/core"" %>. Then, you can use the tags with the c: prefix.

For example, <c:forEach> or <c:if>.

17. What is JSP session tracking? Discuss different methods.

JSP session tracking is a mechanism to maintain stateful interactions with users across multiple requests. Different methods of session tracking in JSP include:

Cookies: Storing a session ID in a browser cookie and associating it with user data on the server.

URL Rewriting: Appending the session ID to URLs as a query parameter.

Hidden Form Fields: Including the session ID as a hidden field in HTML forms.

HttpSession: Using the HttpSession object to store session-related data on the server.

SSL Session: Utilizing the SSL protocol to establish a secure session between the client and the server.

18. What is a JSP Custom Tag? How does it differ from JSP Standard Tag Libraries (JSTL)?

JSP Custom Tags are user-defined tags created by developers to encapsulate custom logic and functionality in JSP pages. They are implemented as Java classes and can be reused across multiple JSP pages.

JSTL (JavaServer Pages Standard Tag Library), on the other hand, provides a set of pre-defined tags for common tasks like iteration, conditional statements, and formatting. JSTL tags are not user-defined; they are part of the JSTL library.

In summary, JSP Custom Tags are created by developers for custom logic, while JSTL provides pre-built tags for common tasks. These are some of the most popular interview questions for jsp and servlet in Java.

19. What are the different scopes in JSP? Provide examples.

In JSP, there are four main scopes to store and access data:

1. Page scope: Data is accessible within the current JSP page only.
2. Request scope: Data is accessible across multiple pages within the same request.
3. Session scope: Data is accessible across multiple requests from the same client.
4. Application scope: Data is accessible by all clients and all requests throughout the application's lifetime.

Examples:

<jsp:useBean> with scope=" page" creates a page-scoped bean.

request.setAttribute(""name"", value) sets a request-scoped attribute.

session.setAttribute(""username"", value) sets a session-scoped attribute.

application.setAttribute(""counter"", value) sets an application-scoped attribute."

Explain the use of JSP actions for database access.

20. JSP actions provide a convenient way to interact with databases within JSP pages. Some commonly used JSP actions for database access are:

<jsp:useBean>: Instantiates or retrieves a JavaBean representing a database connection or a data access object (DAO).

<jsp:setProperty>: Sets properties of a JavaBean representing a database entity or a query parameter.

<jsp:getProperty>: Retrieves properties of a JavaBean representing a database entity or a query result.

<jsp:include>: Includes the content of a JSP page containing database-related logic or query execution.

<jsp:forward>: Forwards the request to a servlet or another JSP page for database processing.

Custom tag actions: Custom tags can be created to encapsulate complex database operations and improve code modularity and reusability.

Unit-3 (Hibernate)

MCQ Questions

1. Which Hibernate interface is used to create a session?

A. SessionFactory B. SessionBuilder

C. SessionFactoryBuilder D. SessionManager

2. Which of the following is a core interface of Hibernate?

A. Configuration B. Transaction

C. Query D. **All of the above**

3. Which method is used to save an object in Hibernate?

A. saveObject)) B. persists))

C. save() **D.** store))

4. Which annotation is used to specify an entity in Hibernate?

A. @Table B. **@Entity**

C. ©Persistence D. @Data

5. Which annotation is used to specify a primary key in Hibernate?

A. @Primary B. @key

C. @id **D.** @pk

6. Which method is used to perform the insert option in HQL query?

A. executeQuery)); B. execute));

C. executeUpdate)); D. createQuery));

7. SessionFactory is.

A. delayed B. self-oriented

C. thread-safe D. lazy

8. The configuration object class is used to create

A. SessionFactory B. Session

C. Transaction D. Repository

9. Which of the following elements is used to declare the persistent class in Hibernate configuration file?

A. <property> B. <session-factory>

C. <mapping> D. <hibernate-configuration>

10. Which of the following provides an interface between application and data stored in the database?

A. Transaction **B. Session**

C. TransactionFactory D.Connectionprovider

11. Which of the following is NOT a valid value for 'hbm2ddl.auto' property in hibernate configuration?

A. validate **B. truncate**

C. create D. update

12. Which of the following is the FIRST Hibernate object that is created in any Hibernate application?

A. Configuration B. Transaction

C. Session D. SessionFactory

13. Which of the following is NOT a valid id generator class?

A. hilo **B. primary**

C. identity D. sequence

14. What is the purpose of Hibernate Criteria API?

A. To write complex queries in HQL
B. To enable lazy loading of a relationship
C. To map a Java object to a database table
D. To programmatically construct queries in Java

15. What is the purpose of Hibernate Query Language (HQL)?

A. To map a Java object to a database table
B. To write complex queries in Java
C. To query the database using a programmatic approach
D. To write complex queries in Hibernate using an object-oriented syntax

16. Which method is used to fetch an entity by its primary key?

A. session.fetch(Class, id)
B. session.load(Class, id)
C. session.retrieve(Class, id)
D. session.get(Class, id)

17. Which method is used to add a restriction in the Criteria API?

A. criteria.add(Restriction)
B. criteria.add(Criterion)
C. criteria.add(Constraint)
D. criteria.add(Condition)

18. Which one of the following method is used to set the parameter value in HQL ?

A. setParameter()
B. setParametersQ
C. getParameter()
D. getParameters()

19. Which method is used to start the transaction ?

A. session.startTransaction();
B. session.beginTransaction();
C. session.Transaction();
D. session.beginsTransaction();

20. Which method is used commit the transaction ?

A. commit()
B. committowork()
C. save()
D. saverecord()

Unit-4 (Hibernate)

Fill in the Blanks

1. **deleted** method is used to delete an object in Hibernate.
2. **@OneToMany** relationship where one entity is associated with multiple entities.
3. Query objects use **HQL** to retrieve data from the relational database
4. The database table configuration is stored in **.hbm** file.
5. A **Transaction** represents a unit of work with the database and most of the RDBMS supports transaction functionality.
6. **<map>** element in hibernate maps java.util.SortedMap property ?
7. **SessionFactory** Hibernate interface is used to create a session?
8. **executeQueryQ** is used to execute an HQL query in Hibernate?
9. **Generator** element of hbm.xml automatically generate the primary key?
10. **Mapping file** are most common configuration methods of Hibernate Configuration.
11. **updated** method is used to update an existing record.
12. **@column** is used to specify the name of column.
13. **First level** is the default level cache in hibernate?
14. **Attached** is the state of object in Hibernate.
15. **get(Class, ID)** fetch an entity by its primary key?
16. **criteria.add(Criterion)** is used to add a restriction in the Criteria API?
17. **@ElementCollection** annotation is used to map a collection of basic or embeddable types.
18. **session.clearQ** method to clear the cache manually.
19. **beginTransactiond** methos is used to start the transaction ?
20. **@Entity** is used to specify an entity in Hibernate?

Unit-4 (Hibernate)
Short Questions

1. What is ORM in Hibernate?
ORM, which stands for *Object-Relational Mapping* acts as a translator that converts the Java object into the relational database. It makes it easier to store and retrieve information on Java objects, by using the ORM technique which reduces a lot of manual work.

2. What is Java Persistence API (JPA)?
Java Persistence API is a collection of classes and methods to persist or store a vast amount of data in a database using ORM.

3. What is HQL?
HQL stands for *Hibernate Query Language* it allows to expression of database queries using entity and property names, rather than relying on traditional SQL that operates on tables and columns. It uses an object-oriented way to operate with Java entity classes and properties.

4. What is a Session in Hibernate?
Session represents the single unit of work that acts as a gateway for interacting with the databases, Hibernate session is the primary interface for working with databases.

5. What is SessionFactory?
SessionFactory in Hibernate is to create and manage the *session* instances that-

- configure the management by reading
- managing the configuration setting of the hibernate environment
- database connection
- mapping metadata (the data inside the data called metadata)
- caching configurations.

6. What is the criteria API in Hibernate?
In hibernate it is a type-safe and programmatic way to make a database using Java code instead of native SQL queries or writing raw HQL (Hibernate Query Language) and helpful for making queries with dynamic sorting, projections and difficult conditions.

7. What happens when the no-args constructor is absent in the Entity bean?
If no-args constructor is absent, instantiation of objects by hibernate can't happen properly which leads to errors. No-args constructor in hibernate is an entity bean which is very essential as in instantiation of objects in hibernate done by its reflection of constructing java objects and reading data from the database.

8. What is lazy loading in Hibernate?
Lazy loading is a technique that is used to defer the loading of an object's child objects until they are actually needed. This can improve performance by reducing the number of database queries that need to be executed.

9. What is the difference between first level cache and second level cache?
First level cache *is a cache that is associated with a Session object. It is used to store objects that have been loaded by the Session.*
Second level cache *is a shared cache that is used to store objects that have been loaded by multiple Session objects. It can be used to improve performance by reducing the number of database queries that need to be executed.*

10. What does "lightweight" mean?
In the context of computers, "lightweight" describes an app, computer program, or device that doesn't use many system resources due to its small memory footprint (RAM) and low CPU usage.

11. What can you tell about the Hibernate Configuration File?

The Hibernate Configuration File (hibernate.cfg.xml) is an XML file used to configure the basic settings of Hibernate, such as the database URL, username, password, and dialect. It also contains the mapping information of the persistent classes and the resources required for connection pooling.

12. Name Hibernate's five collection types used in one-to-many relationship mappings.

The five collection types are:

- Array
- Bag
- List
- Map
- Set

13. Differentiate between get() and load() in Hibernate session.

The get() method retrieves an object from the database by its primary key and throws an exception if it is not found. The load() method is also used to retrieve an object from the database by its primary key, but it returns a proxy object if it is not found.

14. What is hibernate caching?

Hibernate caching refers to storing data in memory so that it can be retrieved quickly without hitting the database again. This improves performance and reduces the load on the database.

15. What is the difference between setMaxResults() and setFetchSizeQ for Query?

The setMaxResults() method limits the number of results a query returns, while the setFetchSize() method controls the number of rows retrieved from the database at a time. setMaxResults() limits the total number of results returned, while setFetchSize() controls the number of rows retrieved at a time to avoid memory issues.

16. How can you view the Hibernate-generated SQL on a console?

To enable viewing SQL on a console for debugging purposes, you must add the following in the Hibernate configuration file:

1 <property name="show_sql">true</property>

17. What are the inheritance mapping strategies?

There are 3 ways of inheritance mapping in hibernate.

1. Table per hierarchy
2. Table per concrete class
3. Table per subclass

18. What are the advantages of Hibernate over JDBC?

The main shortcoming of Java Database Connectivity(JDBC) is that its codes need to be more portable. It is dependent upon the integrated software along with our database. The case is not the same with Hibernate, as it uses objects capable of platform-independent implementation.

19. What is the purpose of a Hibernate Session, and how is it obtained?

The purpose of a Hibernate Session is to provide a single-threaded, short-lived object representing a unit of work for ORM operations. It provides methods to interact with the database to insert, update, delete, or retrieve objects.

20. What is the role of (©Transactional annotation in Hibernate?

The (©Transactional annotation in Hibernate is used to define the scope of a single database transaction. This annotation can be applied to methods or classes to indicate that a method or a series of operations should be executed within a transaction context.

When applied, Hibernate ensures that all database operations within the transaction are either successfully completed and committed or rolled back in case of an error. This helps maintain data integrity and consistency.

Unit-4 (Hibernate)
Long Questions

1. What is Hibernate? Advantages of using Hibernate ?

Hibernate is an open-source framework that makes it easier to store information in a relational database. It converts Java objects to corresponding database tables easily with ORM. It makes automatic database operations by CRUD (Create, Read, Update, Delete) operations. Thus no need to write SQL queries as Hibernate provides a seamless mechanism and better interacting features.

Advantages of using Hibernate

1. Hibernate prevents data corruption as it ensures that the changes to the database are completely applied.
2. Hibernate provides its query language (HQL) which is similar to SQL.
3. HQL speeds up the performance and generates automatic database tables based on the Java class structure.

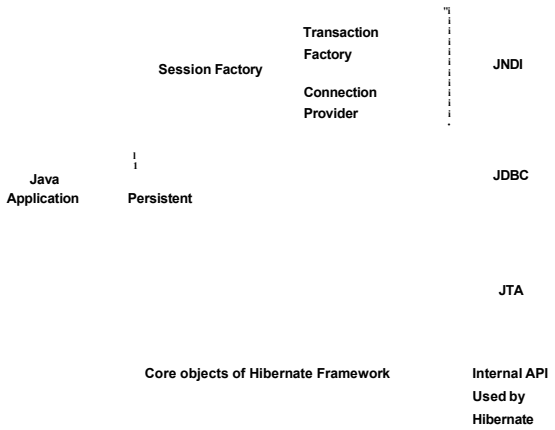
2. What are some of the important interfaces of the Hibernate framework?

1. The **SessionFactory** creates and manages the sessions that are occurring over a short conversation between your program and the database.
2. The session object provides an interface between the application and data stored in the database. It holds the first level cache(mandatory) of data. Sessions are the most important interfaces in the hibernate framework that gives you a tool like adding the data, changing the data, or getting data from the database and once you are done with the session, you need to close this session.
3. **Transaction interface** represents a database transaction including the committing changes to the database or the rollback.
4. **The query interface** is used for creating and executing the queries and retrieving the information from the database.
5. **SessionFactoryBuilder** is used to build a SessionFactory instance based on configuration settings for a new SessionFactory.

3. Explain Hibernate architecture.

Hibernate provides a clear architectural structure and proper segmentation of concerns in the application and is headed by a layer of architecture that has different components responsible for many tasks like

- querying the database,
- ORM (Object-Relational Mapping),
- Transactional management.



The main components of hibernate architecture as:

I. Application Layer

It is the upper-most layer where application belongs. It is linked with the Hibernate API to perform many tasks such as updating, deleting, saving and querying objects from the database.

II. Hibernate API

The Hibernate API interacts with the persistence layer by using a set of classes and interfaces of the application. Some key interfaces like 'Session', 'Transaction' and "SessionFactory'.

III. Configuration

This component can be specified through Java configuration classes or XML files and accountable for providing necessary properties and configuring hibernate for the dialects, database connections and other settings.

```
Configuration configuration = new Configuration));
configuration, configure)" hibernate.cfg.xml");
SessionFactory sessionFactory = configuration.buildSessionFactory));
```

IV. Session Factory:

It is a weighted object which is created only one instance per application and also responsible for governing hibernate sessions and creating instances.

```
SessionFactory sessionFactory = new Configuration))
        .configure("hibernate.cfg.xml")
        .buildSessionFactory));
```

v. Session:

It is a small lifespan object that shows a conversation between database and the application and is also responsible for directing the lifecycle of objects, providing transaction boundaries and performing CRUD operations.

```
Session session = sessionFactory.openSession));
Employee employee = session.get(Employee.class, IL);
session.saveOrllpdate(employee);
session.close));
```

vi. Transaction:

This interface provides methods to commit, rollback and begin the transactions and also used to handle the transactions in hibernate.

```
Session session = sessionFactory.openSession));
Transaction transaction = session.beginTransaction));
try {
   transaction.commit));
}
catch (Exception e) { transaction.rollback));}
finally { session.close));}
```

vii. Mapping Metadata:

It instructs the hibernate how to change the java objects into database records and vice versa. It is also noted through XML files or annotations and responsible for the mapping between database tables and Java classes.

```
@ Entity
@Table(name = "employees")
public class Employee {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
}
```

viii. Object-Relational Mapping (ORM):

It is the core or main functionality in the hibernate which guarantees that the data is correctly updated, stored and retrieved in the database while underlying SQL operations and also responsible for changing database records to Java objects and vice versa.

```
Session session = sessionFactory.openSession();
Transaction transaction = session.beginTransaction();
Employee employee = new Employee();
employee.setName("John Doe");
employee. setAge(30);
session.save(employee);
transaction.commit));
session.close));
```

4. What are the most commonly used annotations available to support hibernate mapping?

To interact with the database hibernates provided a bunch of annotations that you can use for mapping Java classes to database tables. Some common annotations in hibernate for mapping as:

1. @Entity:

It represents the database in the table and points to a Java class as an entity.

```
@ Entity
public class Employee {
}
```

2. @Table:

It explains the details of the table related to the entity.

```
@ Entity
@Table(name = "employees")
public class Employee {
}
```

3. @Id:

@Id denotes the primary key of the entity.

```
@ Entity
@Table(name = "employees")
public class Employee {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
}
```

4. @GeneratedValue:

It denotes how the primary key is made.

```
@ Entity
@Table(name = "employees")
public class Employee {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
   @Column(name = "employee_name")
   private String name;}
```

5. @OneToMany and @ManyToOne:
It settled a many-to-one or one-to-many relationship between entities.

```
@ Entity
@Table(name = "employees")
public class Employee {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
   @ManyToOne
   @JoinColumn(name = "departmentjd")
   private Department department;
}
```

6. @ManyToMany:
Defines a many-to-many relationship between entities. It explains the many-to-many relationship between the entities.

```
@ Entity
@Table(name = "courses")
public class Course {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
   @ManyToMany(mappedBy = "courses")
   private List<Student> students;
   //...
}
```

5. What is HQL? How to create HQL queries in Hibernate?
HQL stands for *Hibernate Query Language* it allows to expression of database queries using entity and property names, rather than relying on traditional SQL that operates on tables and columns. It uses an object-oriented way to operate with Java entity classes and properties.
HQL queries could be dynamic and flexible which supports the aggregation and joins as well such as SUM, and AVG, and uses joins to combine data from the different tables. HQL supports polymorphic queries, meaning you can write a single query that operates on a superclass and retrieves instances of both the superclass and its subclasses.

How to create HQL Queries?

1. First, create an HQL Query String including *entity name, property name,* and *relationships* in the query.
2. Create a query Object by using *org.hibernate.guery.Query* interface for creating and executing queries.
3. Also, query can be created by using *createQuery()*
4. Set Parameters using *setParameter()* is optional.
5. Execute the Query using methods like *list()* to retrieve the data or listing of results.
6. *unigueResult()* for retrieving a single result.

6. What are the states of the object in Hibernate?
In Hibernate, an object can be classified as one of the following states as follows:

1. Transient State
Whenever an object is created by using a *new* keyword and not connected with any of the Hibernate Session, it is called a Transient State. The object is not yet preserved in the database as hibernate is also unknown.

2. Persistent State

The Transient objects will be Persistent when they are linked with the Hibernate Session using the *save()* or *persistf)* method. At this stage, the object is controlled by hibernate and when the transaction is committed with the database when any changes happen it will be tracked and synchronised.

3. Detached State

When an object was once linked with the hibernate session that becomes detached and no longer in the current session's scope. This happens when a transaction is committed/rolled back, object is explicitly removed, or session is ejected.

4. Removed/Deleted State

When an object comes in a state where it is in the removed/deleted state as once it was persistent but expelled explicitly from the database using 'removeO' method or 'deleteQ' method in the hibernate session. The object can't be obtained without re-linking it to the new session and no longer associated with the database.

7. What is the difference between session.save() and session.persistf) method?

session.savef)	session, persistf)
It will give a generated identifier.	It will not generate anything.
It is necessary for non-assigned IDs.	It is not necessary and can be assigned directly.
It will be detached or transient instance.	It should be a Transient instance.
It can't cascade on its own.	It can be cascaded.
It will give the controlled instance.	It will be rest as transient.
It will give you an instant SQL INSERT.	It will not have instant SQL operation.
It is not secured with unsaved transient instances.	It is secured with unsaved transient instances.
It will be cascaded to reverse end of associations.	It will not be cascaded to the reverse end of associations.
It might cause the more database inter-relations.	It might cause the lesser database inter-relations.
It might pursue to more structured use of IDs.	It might pursue to more structured use of IDs but only in some scenes.

8. Can you tell the difference between getCurrentSession and openSession methods?

getCurrentSessionQ	openSessionQ
It will be handled by Hibernate and bind to the transaction.	It will be handled by the application and have to be manually structured.
It is only restricted to current transactions.	It is not restricted to any particular transaction.
It will automatically terminate at the end of the transaction.	It should be closed by hand.
Here hibernate controls the exception propagation.	Here application should control the exceptions.
It will have configuration-handle behavior.	Its explicit nature depends on how it is used.
It will be used again and again only in the same transactions.	It will be used in different transactions also.
It must be known in the present transaction context.	It is not known to any of the transaction context.
It is mostly suitable for short-lived operations.	It is mostly suitable for custom scenes or long-lived operations.

9. Differentiate between save () and saveOrUpdateQ methods in the hibernate session.

It will be available for only transient entities.	It will be available for detached as well as transient entities.
It will throw the exception if the entity presents in DB.	It will modify the entity if available in DB apart from saving it.
It will be the same as transient after being saved.	It will be the same as persistent after being saved.
It must be cascaded to linked entities.	It must be cascaded to entity mapping accordingly.
It should always use ID creation for auto-increment.	It supports multiple ID identifier tactics.
It didn't influence the structured state.	It can reconnect to the detached entity.
It perseveres the transient entity.	It perseveres the transient entity.
It always needed ID creation tactics.	The ID creation tactics may be used here.
It will generate the new entities.	It will generate new or modify available entities.

10. Mention two components of Hibernate configuration object?

It is usually created only once during application initialization. It represents a configuration or properties file required by the Hibernate.

The Configuration object provides two keys components -

DATABASE CONNECTION - This is handled through one or more configuration files supported by Hibernate. These files are hibernate. Properties and hibernate.cfg.xml.

CLASS MAPPING SETUP - This component creates the connection between the Java classes and database tables.

11. What is cascade in hibernate?

Cascading is about persistent actions involving one object propagating to other objects via an association. Cascading can apply to a variety of Hibernate actions, and it is typically transitive. The cascade attribute of the annotation that defines the association says what actions should cascade for that association.

Cascade = "all" means to apply all primary cascade types.

Cascade types are:

- delete / remove
- detach / evict
- merge
- lock
- persist
- refresh
- replicate
- save_update / update

12. What are the key characteristics of Hibernate?

Hibernate has following key characteristics:

Object/Relational Mapping (ORM): Hibernate provides ORM capabilities to developers. So then can write code in Object model for connecting with data in Relational model.

JPA Provider: Hibernate provides an excellent implementation of Java Persistence API (JPA) specification.

Idiomatic persistence: Hibernate provides persistence based on natural Object-oriented idioms with full support for inheritance, polymorphism, association, composition, and the Java collections framework. It can work with any data for persistence.

High Performance: Hibernate provides high level of performance supporting features like- lazy initialization, multiple fetching strategies, optimistic locking etc. Hibernate does not need its own database tables or fields. It can generate SQL at system initialization to provide better performance at runtime.

Scalability: Hibernate works well in multi server clusters. It has built in scalability support. It can work well for small projects as well as for large business software.

Reliable: Hibernate very reliable and stable framework. This is the reason for its worldwide acceptance and popularity among developer community.

Extensible: Hibernate is quite generic in nature. It can be configured and extended as per the use case of application.

13. What is the difference between a transient, persistent, and detached object in Hibernate?

In Hibernate, Objects can remain in three states transient, persistent, or detached. An object which is associated with the Hibernate session is called a persistent object.

Any change in this object will reflect in the database based on your flush strategy i.e. automatic flush whenever any property of object changes or explicit flushing by calling Session.flush() method.

On the other hand, if an object which is earlier associated with Session, but currently not associated with it is called a detached object.

You can reattach detached objects to any other session by calling either the update() or saveOrllpdate() method on that session. Transient objects have newly created an instance of persistence class, which is never associated with any Hibernate Session.

Similarly, you can call persistQ or save() methods to make a transient object persistent. Just remember, here transient doesn't represent the transient keyword in Java, which is an altogether different thing.

14. The difference between sorted and ordered collections in Hibernate? (detailed answer)

The main difference between sorted and ordered collection is that sorted collection sorts the data in JVM's heap memory using Java's collection framework sorting methods while the ordered collection is sorted using order by clause in the database itself.

A sorted collection is more suited for a small dataset but for a large dataset, it's better to use an ordered collection to avoid OutOfMemoryError in Java applications.

15. What is the general flow of Hibernate communication with MySQL?

The general flow of Hibernate communication with RDBMS is :

1. Load the Hibernate configuration file and create configuration object. It will automatically load all hbm mapping files
2. Create session factory from configuration object
3. Get one session from this session factory
4. Create HQL Query
5. Execute query to get list containing Java objects

16. How do you manage transactions using the Session API?

To manage transactions using the Session API, follow these steps:

1. Obtain a Session instance from the SessionFactory.
2. Begin the transaction by calling the **beginTransaction()** method on the Session.
3. Perform CRUD operations using the Session.
4. Commit the transaction using the **commit()** method, or roll back using the **rollback()** method in case of any exception.
5. Close the Session using the **close()** method.

17. Could you please provide an explanation of the distinctions between the save(), persist(), and saveOrUpdate() methods in Hibernate?

The save(), persist(), and saveOrllpdate() methods are different as follows:

a. save(): This method stores an object in the database, generating a new identifier if necessary, and returns the generated identifier. If the object is already persistent, it throws a NonllniqueObjectException.

b. persist(): This method makes a transient instance persistent but does not return the generated identifier. If the object is already persistent, it does nothing.

c. saveOrUpdate(): This method either saves a transient instance or updates the state of a detached instance based on the identifier. If the identifier is null, it saves the object; otherwise, it updates the existing record in the database.

18. What are the different types of primary key generation strategies in Hibernate?

The different types of primary key generation strategies in Hibernate are:

1. AUTO: Hibernate chooses the most appropriate strategy based on the database dialect.
2. IDENTITY: Relies on an auto-incremented identity column provided by the database.
3. SEQUENCE: Uses a database sequence to generate primary key values.
4. TABLE: Uses a separate table to store and generate primary key values.

19. What are the main annotations used in Hibernate for defining entities and their relationships?

The main annotations used in Hibernate for defining entities and their relationships are:

1. @Entity: Marks a class as an entity or a mapped superclass.
2. @Table: Specifies the table that an entity maps to.
3. (©Id: Identifies the primary key of an entity.
4. @GeneratedValue: Specifies the strategy for generating primary key values.
5. (©Column: Specifies the details of the column to which a persistent property or field is mapped.
6. (©OneToOne, @OneToMany, @ManyToOne, (©ManyToMany: Define the various relationships between entities.
7. (©JoinColumn: Specifies the column for joining an entity association or element collection.
8. (©Embeddable: Marks a class as embeddable within other entities.
9. (©Embedded: Specifies a persistent field or property of an entity whose value is an instance of an embeddable class.

20. What are the main validation annotations provided by Hibernate Validator?

The main validation annotations provided by Hibernate Validator are:

1. @NotNull: Ensures a value is not null.
2. (©NotEmpty: Ensures a value is not null or empty (applies to collections, arrays, and strings).
3. (©NotBlank: Ensures a string value is not null or consists of only whitespace characters.
4. (©Size: Validates that the size of a collection, array, or string is within the specified bounds.
5. (©Min, (©Max: Validates that a numeric value is within the specified range.
6. (©Pattern: Validates that a string value matches a specified regular expression pattern.
7. (©Email: Validates that a string value is a well-formed email address.
8. (©Past, (©Future, @PastOrPresent, (©FutureOrPresent: Validates that a date or time value is in the past, future, past or present, or future or present, respectively.
9. (©DecimalMin, (©DecimalMax: Validates that a decimal value is within the specified range.
10. (©Digits: Validates that a numeric value has a certain number of integral and fractional digits.

11. (©Positive, (©PositiveOrZero, (©Negative, (©NegativeOrZero: Validates that a numeric value is positive, positive or zero, negative, or negative or zero, respectively.

These validation annotations are part of the Bean Validation specification and can be used with Hibernate or any other Java persistence framework that supports Bean Validation.

Unit-5 (Spring)

MCQ Questions

1. What is the default scope of the Spring Bean?

A. singleton B. prototype
C. Crequest D. session

2. What is the purpose of the Spring IoC container?

A. To manage the lifecycle of beans and their dependencies B. To handle the configuration of the application
C. To provide a caching mechanism for the application D. To provide a security mechanism for the application

3. Which is the front controller in Spring MVC?

A. DispatcherServlet B) FrontDispatcherServlet
C. FrontControllerService D. None of the above

4. Spring MVC Framework is designed based on which Design Pattern?

A) Model-View-Controller (MVC) B) Layered pattern
C) Client-server pattern D) None of the above

5. Which Spring Framework module is used to implement cross-cutting concerns in Spring based-applications?

A) ORM **B) AOP**
C) JDBC D)TXM

6. JdbcTemplate implements the below design pattern?

A) Strategy design pattern **B) Template Design pattern**
C) Singleton design pattern D) Decorator design pattern

7. Which Spring annotation is used to handle HTTP POST requests?

A) @GetMapping B) @PutMapping
C) (©CreateMapping **D) @PostMapping**

8. Which annotation do we use to mark the class as a Service class/component?

A) @Component **B) (©Service**
C) (©Controller D) (©Repository

9. Which Spring annotation is used to extract the URI template variable value?

A) (©Pathvariable B) (©ParamRequest
C) (©ModelAttribute D) (©RequestMapping

10. Which interface represents the Spring IOC container?

A) Applicationcontent B) ApplicationContentFactory
C SessionFactory D) DispatchServlet

11. Which of the following annotations is NOT used to specify that a class is a Spring bean?

A) (©Service **B) (©Qualifier**
C) (©Component D) (©Controller

12. Which annotation is used to enable Spring's aspect-oriented programming (AOP) support?

A) (©Aspect B) (©AspectJ
C) (©EnableAop D) (©EnableAspectJAutoProxy

13. Which annotation is used to declare a class as a Spring bean in a Java based configuration class?

A) (©Service **B) (©Bean**
C) (©Component D) (©Autowired

14. Which of the following tag is required to enable component scanning in Spring XML configuration?

A) <context:scan package="com.example" />

B) <scan-components base-package="com.example" />

C) <component-scan base-package="com.example" />

D) <scan: component base-package-'com.example" />

15. Which annotation is used to enable transaction management in Spring?

A) @Transactional

B) @EnableTransactionManagement

C) @ConfigureTransaction

D) @EnableTransactions

16. Which is the correct option to enable Spring MVC in a Spring based application?

A) Using the @EnableMvc annotation

B) Using the <enable-mvc> tag

C) Using the @EnableWebMvc annotation

D) Using the (©EnableSpringMvc annotation

17. Which of the following is helpful in passing data from a controller method to a view in Spring MVC?

A) HttpRequest object

B) HttpServletResponse object

C) HttpSession object

D) ModelAttributes to the Model object

18. Which of the following is used to hold the model data and view information in Spring MVC?

A) ViewResolver

B) Model

C) ModelAttribute

D) ModelAndView

19. Which advice type is executed regardless of the method execution outcome, including exceptions in Spring AOP?

A) Before advice

B) After advice

C) Around advice

D) After advice

20. Which Spring annotation is used to inject the Spring bean?

A) @Bean

B) @Autowired

C) (©Service

D) @Inject

Unit-5 (Spring)

Fill in the blanks

1. **Spring Web MVC** module provide a web framework for building web applications
2. **Prototype** is the scope of creating a new instance every time the bean is requested from the spring container
3. DispatcherServlet consults **View Resolver** to map the logical view name with the actual view implementation
4. **<bean>** tag is used to define a bean in Spring XML configuration file.
5. **@Autowired** annotation is used to marks a bean for automatic injection
6. **@After** advice is executed once a joint point finishes?
7. **IOC** container is used to managed the bean object life cycle?
8. **Dependency Injection(DI)** is a design technique that eliminates dependencies from programming code to make the application easier to manage and test.
9. Spring framework's **@Autowired** capability allows you to insert object dependencies indirectly.
10. **@Around Advice** type of advice executes before and after a joint point?
11. **DispatcherServlet** is a class that accepts incoming requests and routes them to the appropriate resources such as controllers, models, and views.
12. **@Controller** designates a class as the controller?
13. **AOP** is used to increase modularity by cross-cutting concerns.
14. **@RequestMapping** annotation is used to associate a URL with a whole class or a specific handler function?
15. **Class** attribute is used to specify class name of the bean?
16. **scope** attribute is used to set the scope of the bean?
17. **Servlet-mapping** attribute used to handle web flow requests.
18. org.hibernate.SessionFactory can manage contextual sessions and allows to retrieve them by **getCurrentSessionQ**
19. **@Id** annotations is used to indicate an entity's primary key field?
20. **ModelAndView** is used to hold the model data and view information in Spring MVC?

Unit-5 (Spring)

Short Questions

1. What are the core features of the Spring Framework? The core features of the Spring Framework include Dependency Injection (DI), Aspect-Oriented Programming (AOP), and various modules for different functionalities.

2. What is Dependency Injection (DI) in Spring? Dependency Injection is a design pattern where the dependencies of a class are injected from the outside, rather than the class creating its own dependencies. Spring's DI container manages these dependencies.

3. Explain Inversion of Control (IoC) in Spring. Inversion of Control is a concept where the control over the flow of a program's execution is shifted from the program itself to a framework. In Spring, the IoC container manages the creation and injection of objects.

4. What is Aspect-Oriented Programming (AOP) in Spring? AOP is a programming paradigm that separates concerns by enabling modularization of cross-cutting concerns, such as logging, security, and transaction management. Spring AOP allows you to define aspects and apply them to your code.

5. What's the difference between BeanFactory and Applicationcontext? Applicationcontext is a more feature-rich containerthan BeanFactory, providing additional functionalities like internationalization, event propagation, and more.

6. How is Dependency Injection done in Spring? Dependency Injection is achieved in Spring by either constructor injection, setter injection, or method injection. The container injects the required dependencies.

7. What is the use of the (©Component annotation? The **(©Component** annotation is used to mark a Java class as a Spring-managed component, allowing the Spring container to create and manage instances of that class.

8. Explain the (©Autowired annotation. The **(©Autowired** annotation is used to automatically wire beans by type. It injects the dependency automatically without the need for explicit setter methods.

9. How does Spring handle Transactions? Spring provides a powerful abstraction for managing transactions using the **(©Transactional** annotation or programmatic configuration.

10. What is an ORM framework, and how does Spring support it? An Object-Relational Mapping (ORM) framework maps Java objects to database tables. Spring supports various ORM frameworks like Hibernate, JPA, and MyBatis.

11. What is Spring Data JPA? Spring Data JPA is a sub-project that simplifies the integration of JPA (Java Persistence API) with Spring applications.

12. How does Spring support Declarative Transaction Management? Spring supports declarative transaction management using annotations or XML configuration. Transactions can be defined at the method level using the **(©Transactional** annotation.

13. What is an Aspect in Spring AOP? An aspect is a module that encapsulates cross-cutting concerns. It's a reusable module that can be applied to multiple parts of the application.

14. What is a Joinpoint in Spring AOP? A joinpoint is a point during the execution of a program where an aspect can be applied. In Spring AOP, joinpoints are method executions, method calls, and more.

15. What is a Pointcut in Spring AOP? A pointcut is an expression that defines a set of joinpoints where an aspect's advice should be applied. It specifies the methods or classes where the aspect should be applied.

16. What is a Controller in Spring MVC? A controller in Spring MVC is responsible for processing user requests, interacting with the model, and returning the appropriate view to the client.

17. What is the purpose of the @ModelAttribute annotation? The **@ModelAttribute** annotation is used to bind a method parameter or method return value to a named model attribute. It's often used for form data binding.

18. How is form data handling done in Spring MVC?
Form data handling is done using the **@RequestParam, @ModelAttribute,** or **@RequestBody** annotations. These annotations help in mapping incoming data to method parameters.

19. Explain the concept of Advice in Spring AOP.
Advice is the action taken by an aspect at a particular joinpoint. Spring AOP provides different types of advice like Before, After, AfterReturning, AfterThrowing, and Around.

20. What is Spring EL (Expression Language)?
Spring EL is a language that supports querying and manipulating objects during runtime. It's used for bean property values, annotations, XML configuration, etc.

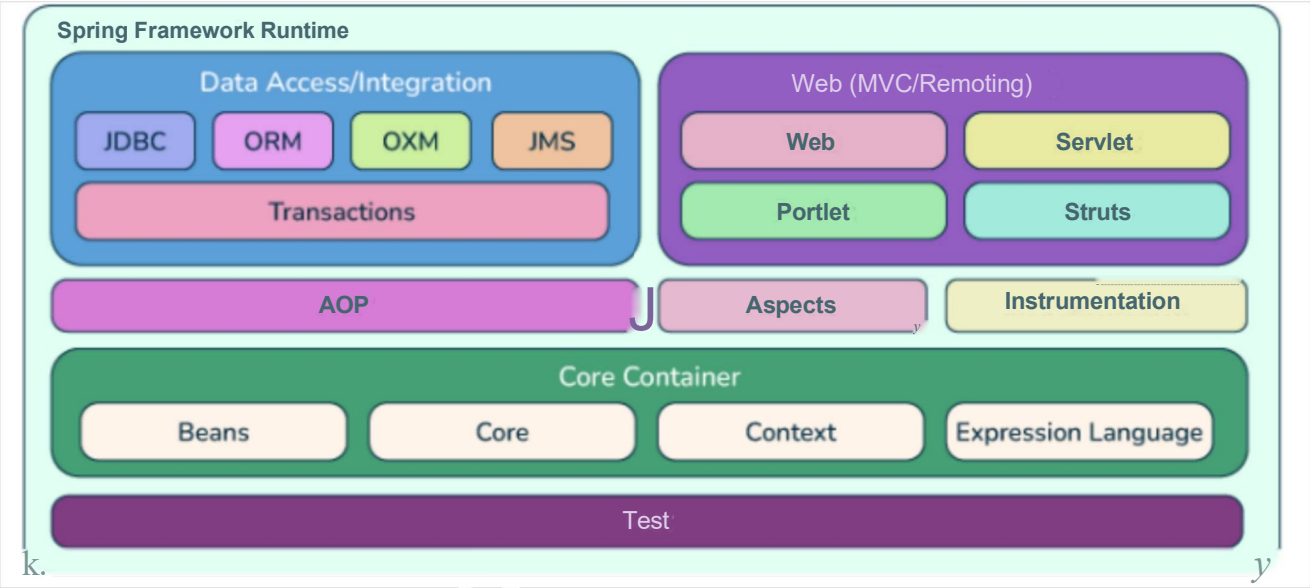
Unit-5 (Spring)

Long Questions

1. What is Spring Framework ? What are the advantages of using Spring Framework ?

Spring framework is an open-source Java framework that supports building robust Java applications. It mainly handles all the infrastructure-related aspects allowing the developer to focus more on application development, making it the world's most popular Java framework.

- Spring is used in every domain, even in big techs like Amazon, Google, etc.
- Features like *IoC* and *DI* provide a set of features and functionality.
- Increased productivity as redundant configuration is not required.
- Enormous community support.



Advantages of using Spring Framework

- **High Productivity:** Reduced boilerplate code(Lombok, etc), faster development(auto-config), and simplified testing(JUnit).
- **Easy to Maintain:** Loose coupling, separation of concerns, and cleaner code structure.
- **Security:** Dedicated security framework, built-in authentication and authorization, and data protection features.
- **Large Community Support:** Large and active community support, documentation support, etc.

2. Explain Inversion of Control(IoC) and types of IoC containers.

IoC stands for Inversion of Control means transferring the control of managing the dependencies and their injection when required from the application to the container/framework, It increases code scalability, maintainability, and easy testing.

Types of IoC Containers:

- **Bean Factory:** Basic container, that provides basic object creation and dependency injection for spring applications.
- **Application Context:** Advanced container, is an implementation of BeanFactory. Can manage object lifecycles, events, and resource access.

3. What is autowiring and name the different modes of it?

The IoC container autowires relationships between the application beans. Spring lets collaborators resolve which bean has to be wired automatically by inspecting the contents of the BeanFactory. Different modes of this process are:

- **no:** This means **no autowiring** and is the default setting. An explicit bean reference should be used for wiring.
- **byName:** The bean dependency is injected according to the **name of the bean.** This matches and wires its properties with the beans defined by the same names as per the configuration.
- **byType:** This injects the bean dependency based on **type.**
- **constructor:** Here, it injects the bean dependency **by calling the constructor** of the class. It has a large number of parameters.
- **autodetect:** First the container tries to wire using autowire by the constructor, if it isn't possible then it tries to autowire by byType.

4. Explain Bean life cycle in Spring Bean Factory Container.

The Bean life cycle is as follows:

- The IoC container instantiates the bean from the bean's definition in the XML file.
- Spring then populates all of the properties using the dependency injection as specified in the bean definition.
- The bean factory container calls setBeanName() which take the bean ID and the corresponding bean has to implement BeanNameAware interface.
- The factory then calls setBeanFactoryf) by passing an instance of itself (if BeanFactoryAware interface is implemented in the bean).
- If BeanPostProcessors is associated with a bean, then the preProcessBeforeInitialization() methods are invoked.
- If an init-method is specified, then it will be called.
- Lastly, postProcessAfterInitialization() methods will be called if there are any BeanPostProcessors associated with the bean that needs to be run post creation.

5. How is the configuration meta data provided to the spring container?

There are 3 ways of providing the configuration metadata. They are as follows:

- **XML-Based configuration:**
  The bean configurationsand their dependencies are specified in XML configuration files. This starts with a bean tag as shown below:

```
<bean id="interviewBitBean" class="org.intervuewBit.firstSpring.InterviewBitBean">
        <property name="name" value="InterviewBit"x/property>
</bean>
```

- **Annotation-Based configuration:**
  Instead of the XML approach, the beans can be configured into the component class itself by using annotations on the relevant class, method, or field declaration.

  - Annotation wiring is not active in the Spring container by default. This has to be enabled in the Spring XML configuration file as shown below Explain

```
<beans>
        <context:annotation-config/>
                <!- bean definitions go here -->
```

</beans>

- **Java-based configuration:**

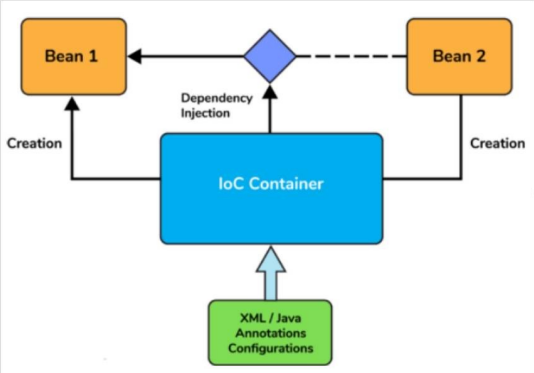
  Spring Framework introduced key features as part of new Java configuration support. This makes use of the **(©Configuration** annotated classes and **(©Bean** annotated methods. **Note that:**

  - o (©Bean annotation has the same role as the <bean/> element.
  - o Classes annotated with (©Configuration allow to define inter-bean dependencies by simply calling other (©Bean methods in the same class.

6. What do you understand by Dependency Injection?

The main idea in Dependency Injection is that you don't have to create your objects but you just have to describe how they should be created.

- The components and services need not be connected by us in the code directly. We have to describe which services are needed by which components in the configuration file. The IoC container present in Spring will wire them up together.



In Java, the 2 major ways of achieving dependency injection are:

- o Constructor injection: Here, the IoC container invokes the class constructor with a number of arguments where each argument represents a dependency on the other class.
- o Setter injection: Here, the spring container calls the setter methods on the beans after invoking a no-argument static factory method or default constructor to instantiate the bean

7. Can you create a controller without using (©Controller or (©RestController annotations?

Yes, you can create a controller in Spring without using the (©Controller or (©RestController annotations. The (©Controller and (©RestController annotations are just convenience annotations that provide specific functionalities, but you can achieve the same functionality by using other annotations or configuration.

To create a controller without using (©Controller or (©RestController, you can use the following approach:

- Implement the Controller Logic: Create a regular Java class that contains the logic for handling HTTP requests and generating responses.
- Use Appropriate Annotations: Instead of (©Controller or (©RestController, you can use other annotations to specify the request mappings and the response type.

8.What are the differences between (©Request Pa ram and (©Pathvariable annotations?

@RequestParam	@PathVariable
Extracts query parameters from the URL's query string.	Extracts values from the URI path itself (URL template).
Followed by the parameter name in the controller method's parameter list.	Followed by the variable name in curly braces {} within the URL mapping.
/example?name=John	/example/{id}
@RequestParam("name") String name	@PathVariable("id") String id
Suitable for optional parameters or data in the query string.	Useful for extracting dynamic values from the URL path.

9. What is the role of (©ModelAttribute annotation?

The @ModelAttribute annotation in Spring MVC is used to bind method parameters or method return values to model attributes. It plays a crucial role in the Model-View-Controller (MVC) architecture, where it helps transfer data between the Controller and the View.

The key role of (©ModelAttribute is to facilitate data transfer between the Controller and the View. It allows you to pre-populate form data when displaying forms to users and automatically bind user inputs to model attributes when processing form submissions. Additionally, it helps in adding common attributes (like reference data) to the model across multiple controller methods.

10. What are the differences between the <context:annotation-config> vs <context:component-scan> tags?

<context:annotation-config>: This is used to activate various annotations within Spring-managed beans. For example, if you have beans that are manually defined in your XML file and these beans have annotations like (©Autowired, @PostConstruct, (©PreDestroy, @Resource, etc., you need <context:annotation-config> to activate these annotations. However, it does not automatically detect and instantiate beans from the classpath.

<context:component-scan>: This tag does everything that <context:annotation-config> does, but it goes one step further. It scans the classpath for classes annotated with (©Component, (©Service, (©Repository, (©Controller, etc., and automatically registers them as beans in the Spring application context. In other words, it automatically detects and instantiates your beans. So, when you use <context:component-scan>, you don't need to explicitly define each bean in your XML configuration file, as long as they are annotated correctly and exist within the base-package specified by <context:component-scan>.

11. What's the difference between (©Component, (©Controller, (©Repository & (©Service annotations in Spring?

(©Component: This marks a java class as a bean. It is a generic stereotype for any Spring-managed component. The component-scanning mechanism of spring now can pick it up and pull it into the application context.
(©Controller: This marks a class as a Spring Web MVC controller. Beans marked with it are automatically imported into the Dependency Injection container.
(©Service: This annotation is a specialization of the component annotation. It doesn't provide any additional behavior over the (©Component annotation. You can use (©Service over (©Component in service-layer classes as it specifies intent in a better way.
(©Repository: This annotation is a specialization of the (©Component annotation with similar use and functionality. It provides additional benefits specifically for DAOs. It imports the DAOs into the DI container and makes the unchecked exceptions eligible fortranslation into Spring DataAccessException.

12. What do you mean by Aspect? Explain JoinPoint, PointCut, Advice and Weaving.

Aspect is a modularization of concern which cuts across multiple objects. Transaction management is a good example of a crosscutting concern in J2EE applications. Aspects are implemented using regular classes or regular classes annotated with the (©Aspect annotation in Spring Framework.

JoinPoint.
A point during the execution of a program is called JoinPoint, such as the execution of a method or the handling of an exception. In Spring AOP, a joinpoint always represents a method execution.

Pointcut?
The pointcut is a set of one or more joinpoints where advice should be executed. You can specify pointcuts using expressions or patterns.

Advice
An Action taken by an aspect at a particular joinpoint is known as an Advice. Spring AOP uses an advice as an interceptor, maintaining a chain of interceptors "around" the join point.

Weaving
The process of linking an aspect with other application types or objects to create an advised object is called Weaving.

13. What are the different types of Advices?

Different types of Advices in Spring AOP are:

a. Before: These types of advices execute before the joinpoint methods and are configured using @Before annotation mark.

b. After returning: These types of advices execute after the joinpoint methods completes executing normally and are configured using (©AfterReturning annotation mark.

c. After throwing: These types of advices execute only if joinpoint method exits by throwing an exception and are configured using (©AfterThrowing annotation mark.

d. After (finally): These types of advices execute after a joinpoint method, regardless of the method's exit whether normally or exceptional return and are configured using (©After annotation mark.

e. Around: These types of advices execute before and after a joinpoint and are configured using @ Around annotation mark.

14. What type of transaction Management Spring support?

This spring interview question is a little difficult as compared to previous questions just because transaction management is a complex concept and not every developer familiar with it. Transaction management is critical in any application that will interact with the database.

The application has to ensure that the data is consistent and the integrity of the data is maintained. Following two types of transaction management is supported by spring:

1. Programmatic transaction management
2. Declarative transaction management.

15.How is the configuration metadata provided to the Spring container?

There are three ways in which the configuration metadata is provided to the Spring container, enumerated as follows:

- Annotation-based Configuration - By default, annotation wiring is turned off in the Spring container. Using annotations on the applicable class, field, or method declaration allows it to be used as a replacement of using XML for describing a bean wiring.
- Java-based Configuration - This is the newest form of configuration metadata in Spring Framework. It has two important components:
  - @Bean annotation - Same as that of the <bean/> element
  - (©Configuration annotation - Allows defining inter-bean dependencies by simply calling other @Bean methods in the same (©Configuration class
- XML-based Configuration - The dependencies, as well as the services required by beans, are specified in configuration files that follow the XML format. Typically, these configuration files contain several application-specific configuration options and bean definitions.

Unit-5 (Spring Boot)

MCQ Questions

1. What feature of Spring Boot simplifies the configuration process?

A Auto-configuration B Annotations to configure beans and dependencies.
C XML-based configuration support. D A command-line interface for configuring beans.

2. Which annotation is used to create RESTful web services in Spring Boot?

A @RestController B (©Controller
C (©Component D (©Service

3. Which annotation is used to mark a class as a service component in Spring?

A. (©Component **(©Service**
B (©ControllerD (©Repository

4. Which class serves as the default implementation of the *JpaRepository* interface?

A SimpleJpaRepository B BJpaRepositorylmpl
C CCustomJpaRepository D DefaultJpaRepository

5. Which starter dependency is used for developing web applications or RESTful web services in Spring Boot?

A spring-boot-starter-data-jpa **B spring-boot-starter-web**
C spring-boot-starter-rest D spring-boot-starter-web-dependency

6. What is the purpose of the *@SpringBootApplication* annotation?

A To enable Spring Boot auto-configuration. B To define a Spring Boot starter class.
C To define a Spring Boot controller. D To define a Spring Boot service.

7 What is the use of the *@Transactional* annotation in Spring Boot?

A To define the transaction isolation level **B To manage database transactions automatically**
C To handle security configurations D To execute methods asynchronously

8 Which embedded servlet container is NOT supported by default in Spring Boot?

A Tomcat B Jetty
C Undertow **D Apache HTTP Server**

9 What is the default scope of a Spring bean in Spring Boot?

A Prototype B Request
C Singleton D Session

10. What is the purpose of the (©Service annotation in Spring Boot?

A To define a REST controller **B To mark a class as a service provider**
C To create scheduled tasks D To configure application properties

11. What is the role of the (©RequestBody annotation in a Spring Boot controller?

A Tobind a method parameter to a web request body B To define a request parameter
C To return a response body D To configure request headers

12. What is the purpose of the (©Pathvariable annotation in Spring Boot?

A To bind a method parameter to a path variable in a URL B To define a request parameter
C To configure request headers D To return a response body

13. Which component is responsible for handling view resolution in a Spring Boot web application?

A JdbcTemplate **B ViewResolver**
C DataSource D WebMvcConfigurer

14. How can you define a JPA repository in Spring Boot?

A By creating a class that extends JpaRepository
B By annotating a class with (©Repository
C By defining an interface that extends JpaRepository
D By using the (©JpaRepository annotation

15. In Spring Boot, how is a bean defined?

A Through XML configuration files
B By using the (©Bean annotation in a configuration class
C By declaring it in the application.properties file
D By using a dedicated BeanFactory

16. What is the role of the (©RequestParam annotation in a Spring Boot web application?

A To bind a method parameter to a web request parameter
B To define a RESTful endpoint
C To configure application security
D To define a scheduled task

17. What does the @SpringBootApplication annotation do?

A Enables JDBC
B Configures a web application
C Declares a configuration class
D Starts a Spring context

18. Spring Boot is best suited for creating what type of applications?

A Batch applications
B Web applications
C Enterprise applications
D Desktop applications

19. What is the purpose of the application.properties file in a Spring Boot application?

A To configure the application's properties
B To store application resources
C To manage external dependencies
D None of the above

20 In REST API design, what HTTP method is most commonly used to update resources?

A GET
BPOST
C PUT
D DELETE

Unit-5 (Spring Boot)

Fill in the blanks

1. **Tomcat** embedded servers does Spring Boot support?
2. **Spring Core** module of Spring Framework is the foundation for Spring Boot?
3. **@SpringBootApplication** annotation is used to define a Spring Boot application's main class?
4. ViewResolver component is responsible for handling view resolution in a Spring Boot web application?
5. **8080** default port for a web application in Spring Boot?
6. The default HTML template engine provided by Spring Boot is **Thymeleaf**
7. @ **Request Ma ppi ng** annotation configures the base URI for a controller in a Spring Boot REST API?
8. **Spring Security** module in Spring Boot provides support for securing web applications?
9. **@Autowired** is used to inject dependencies into Spring Boot applications.
10. **@Repository** annotation is used to define a JPA repository in Spring Boot?
11. **@RestController** annotation is primarily used for building RESTful web services in Spring Boot?
12. **@RequestBody** specify a request body in a Spring Boot REST controller method?
13. The **PUT** HTTP method is most commonly used to update resources?
14. The **spring-boot-starter-data-jpa** Spring Boot starter is used for integrating Spring Data JPA into an application?
15. The @Query used to execute a custom query with Spring Data JPA repository?
16. **ThymeLeaf** is the default HTML template engine provided by Spring Boot?
18. **HTTP** protocol is used in REST?
19. **500** HTTP Code Related to Internal Server Error?
20. **@Transactional** is used to manage transactions declaratively

Unit-5 (Spring Boot)

Short Questions

1. What is Spring Initializr?
A web-based tool to quickly generate Spring project template with predefined configurations and dependencies.

2. Which Spring Boot starter is used for integrating Spring Data JPA into an application?
spring-boot-starter-data-jpa is used for integrating Spring Data JPA into spring application

3. What does the (©Transactional annotation do in a Spring Boot application?
It is used to Manages the scope of a single database transaction

4. What role does the JdbcTemplate play in Spring Boot?
It is used to simplifies JDBC operations by handling boilerplate code.

5. Which annotation is used to enable Spring Security's web security support in a Spring Boot application?
(©EnableWebSecurity is used to enable Spring Security's web security support in a Spring Boot application.

6. What annotation is primarily used to indicate a class is a Spring Boot test class?
@SpringBootTest is used to indicate a class is a Spring Boot test class

7. What is Spring Initializer primarily used for?
It is used generating boilerplate code for Spring Boot applications.

8. Which Spring Boot Starter would you use for developing web applications?
spring-boot-starter-web is use for developing web application.

9. Which annotation ensures that HTTP methods like GET and POST are mapped to specific controller methods?
@RequestMapping

10. How do you change the default port (8080) on which a Spring Boot web application runs?
server.port in application.properties is used to change the default port in spring application.

11. Which Spring annotation is used to handle HTTP POST requests?
@PutMapping is used to handle HTTP Post request in spring boot application.

12. How can you specify the port on which a Spring Boot application runs?
By modifying the application.properties file.

13. What is the main difference between PUT and PATCH in a REST API?
PUT completely updates an existing resource, while PATCH modifies only the fields specified in the request.

14. What is the use of the @*Transactional* annotation in Spring Boot?
To manage database transactions automatically

15. What is the default scope of a Spring bean in Spring Boot?
Singleton

16. In Spring Boot, how is a bean defined?
By using the @Bean annotation in a configuration class

17. What is the role of the (©RequestParam annotation in a Spring Boot web application?
To bind a method parameter to a web request parameter.

18. What is the purpose of Spring Boot Actuator?
It is used to provide production-ready features such as monitoring and metrics.

19. What is the role of the "Content-Type" header in a REST API request?
It defines the format of the request payload

20. what is idempotency?
The property that a method can be called multiple times without different outcomes

Unit-5 (Spring Boot)

Long Questions

1. What Is Spring Boot ? What Are Its Main Features?

Spring Boot is essentially a framework for rapid application development built on top of the Spring Framework. With its auto-configuration and embedded application server support, combined with the extensive documentation and community support it enjoys, Spring Boot is one of the most popular technologies in the Java ecosystem as of date.

There are many useful features of Spring Boot. Some of them are mentioned below:

- **Auto-configuration -** Spring Boot automatically configures dependencies by using **(©EnableAutoconfiguration** annotation and reduces boilerplate code.
- **Spring Boot Starter POM -** These Starter POMs are pre-configured dependencies for functions like database, security, maven configuration etc.
- **Spring Boot CLI (Command Line Interface) -** This command line tool is generally for managing dependencies, creating projects and running the applications.
- **Actuator -** Spring Boot Actuator provides health check, metrics and monitors the endpoints of the application. It also simplifies the troubleshooting management.
- **Embedded Servers -** Spring Boot contains embedded servers like Tomcat and Jetty for quick application run. No need of external servers.

2. What are the differences between (©Component, (©Service, (©Repository, and (©Controller annotations?

These annotations are specializations of (©Component:

- (©Component: Generic stereotype for any Spring-managed component.
- (©Service: Specialization for service layer classes.
- (©Repository: Specialization for persistence layer classes.
- (©Controller: Specialization for presentation layer (MVC controllers).

3. What are the steps in the Spring Bean Life Cycle?

Apart from Spring Boot, the interviewer may also ask **Spring Bean Life Cycle interview questions** like this one.

The Spring Bean Life Cycle includes:

- **Instantiation:** Creating the bean instance.
- **Population:** Injecting dependencies.
- **Initialization:** Calling (©PostConstruct methods and custom init methods.
- **Ready for Use:** The bean is now ready to be used.
- **Destruction:** Calling (©PreDestroy methods and custom destroy methods before the bean is removed from the container.

4. What does the (©SpringBootApplication annotation do internally?

The **(©SpringBootApplication** annotation combines three annotations. Those three annotations are: **(©Configuration, (©EnableAutoConfiguration,** and **(©ComponentScan .**

- **(©AutoConfiguration :** This annotation automatically configuring beans in the class path and automatically scans the dependencies according to the application need.
- **@ComponentScan :** This annotation scans the components ((©Component, (©Service, etc.) in the package of annotated class and its sub-packages.
- **(©Configuration:** This annotation configures the beans and packages in the class path.

(©SpringBootApplication automatically configures the application based on the dependencies added during project creation and bootstraps the application by using run() method inside the main class of an application.

@SpringBootApplication = (^Configuration + @EnableAutoConfiguration + @ComponentScan

5. What are the basic Spring Boot Annotations?

- **(©SpringBootApplication:** This is the main annotation used to bootstrap a Spring Boot application. It combines three annotations: **(©Configuration , (©EnableAutoConfiguration ,** and **(©ComponentScan .** It is typically placed on the main class of the application.
- **(©Configuration:** This annotation is used to indicate that a class contains configuration methods for the application context. It is typically used in combination with (©Bean annotations to define beans and their dependencies.
- **(©Component:** This annotation is the most generic annotation for any Spring-managed component. It is used to mark a class as a Spring bean that will be managed by the Spring container.
- **(©RestController:** This annotation is used to define a RESTful web service controller. It is a specialized version of the (©Controller annotation that includes the (©ResponseBody annotation by default.
- **(©RequestMapping:** This annotation is used to map HTTP requests to a specific method in a controller. It can be applied at the class level to define a base URL for all methods in the class, or at the method level to specify a specific URL mapping.

6 How does Spring Boot's inversion of control (IoC) container impact application design?

Spring Boot's **Inversion of Control (IoC)** container plays a pivotal role in shaping the architecture and design of an application by providing a framework for managing dependencies. IoC in Spring Boot is primarily achieved through **Dependency Injection (DI),** which moves the responsibility of managing object creation and dependencies from the code to the Spring framework itself. This leads to cleaner, modular, and loosely coupled code.

Here's how the IoC container impacts application design:

Loose Coupling and Modularity : IoC promotes **loose coupling** by decoupling the creation and management of objects (or beans) from the actual business logic. Rather than having objects create and manage their own dependencies, the IoC container does it for them.

Improved Testability : By externalizing the creation of dependencies, IoC makes it easier to isolate components for testing. You can inject mock or stub objects in place of actual implementations during tests.

Centralized Configuration and Lifecycle Management: The IoC container provides a centralized place for configuring, managing, and controlling the lifecycle of all the beans (components) in the application.

Encourages Dependency Injection (DI) Best Practices : IoC in Spring Boot encourages the use of **Dependency Injection,** which is a core design principle of the Spring framework.

Supports Aspect-Oriented Programming (AOP) : IoC allows Spring to support **Aspect-Oriented Programming (AOP),** where you can inject cross-cutting concerns (e.g., logging, security, transaction management) into application components without affecting the core logic.

Improves Code Reusability and Flexibility : By managing the dependencies through IoC, the same beans can be reused across different parts of the application.

Supports Auto-Configuration and Convention Over Configuration : Spring Boot provides auto-configuration capabilities, which allow the IoC container to automatically configure beans based on the dependencies in the classpath. This reduces the need for explicit bean configuration.

Decouples Application Components from Configuration Details : The IoC container externalizes configuration details, allowing for application logic to remain clean and focused on its core functionality.

7. Difference between (©Controller and (©RestController

Features	(©Controller	(©RestController
Usage	It marks a class as a controller class.	It combines two annotations i.e. (©Controller and (©ResponseBody.
Application	Used for Web applications.	Used for RESTful APIs.
Request handling and Mapping	Used with (©RequestMapping annotation to map HTTP requests with methods.	Used to handle requests like GET, PUT, POST, and DELETE.

8. What are the differences between @SpringBootApplication and (©EnableAutoConfiguration annotation?

Features	(©SpringBootApplication	@EnableAutoConfiguration
When to use	When we want to use auto-configuration	When we want to customize auto-configuration
Entry point	Typically used on the main class of a Spring Boot application, serving as the entry point.	Can be used on any configuration class or in conjunction with (©SpringBootApplication.
Component Scanning	Includes **(©ComponentScan** annotation to enable component scanning.	Does not perform component scanning by itself.
Example	(©SpringBootApplication public class MyApplication { public static void main(String[] args) { SpringApplication.run(MyApplication.class, args);}}	(©Configuration (©EnableAutoConfiguration public class MyConfiguration {}

9. What are the (©RequestMapping and (©RestController annotations in Spring Boot used for?

(©RequestMapping: (©RequestMapping is used to map HTTP requests to handler methods in your controller classes. It can be used at the class level and method level. It supports mapping by:

- HTTP method - GET, POST, PUT, DELETE

- URL path
- URL parameters
- Request headers

@RestController: @RestController is a convenience annotation that combines **(©Controller** and **@ResponseBody .** It indicates a controller where every method returns a domain object instead of a view.

@RestController = ^Controller + @ResponseBody

10. Can you explain the different types of dependency injection supported by Spring Boot?

Spring Boot, like the broader Spring Framework, supports three main types of **Dependency Injection (DI): Constructor Injection, Setter Injection,** and **Field Injection.** These types are ways of providing dependencies to classes, making the application loosely coupled and easier to maintain. Here's a detailed explanation of each type:

1. Constructor Injection

This is the most preferred and widely recommended type of dependency injection. In **constructor injection,** dependencies are provided to the class via its constructor. The dependencies are passed when the object is instantiated.

Benefits:

- Ensures that the object is always in a fully initialized state when it is created.
- Dependencies are immutable once set, which improves code stability and avoids issues related to changing dependencies at runtime.
- Easier for testing with mock objects (ideal for unit tests).

Setter Injection

In **setter injection,** Spring injects dependencies via setter methods. After creating the object, Spring calls setter methods and injects the dependencies.

Benefits:

- Useful when you want to inject optional dependencies that may not be required for the object to function properly.
- Can modify or reconfigure dependencies later during the lifecycle of the object if needed (though this can lead to mutability issues).

Field Injection

In **field injection,** Spring directly injects dependencies into the fields of a class using the @Autowired annotation. No constructor or setter method is needed.

Benefits:

- The simplest form of injection as it requires the least amount of code (no constructor or setter).
- Commonly used in small projects or when prototyping because of its simplicity.

12. Difference between Spring Boot and Spring Framework

Spring Boot	Spring Framework
Spring Boot automatically configures your project, making setup simpler and faster.	Manual configuration is required, which might take a long and be complex.
Allows faster development with less boilerplate code.	Development is slower because more setup is needed.
Manages dependencies automatically, simplifying project setup.	Dependencies need to be managed manually, which can be complicated.
It includes an embedded server that allows you to run applications directly.	Requires an external server setup to deploy applications.
Designed for **microservices,** making them easy to create and deploy.	It can be used for microservices but requires more configuration.
It includes features like metrics and health checks that are out of the box.	Does not include these features by default; they need to be added manually.
Has a large community with frequent updates and extensive resources.	Also has a strong community, but it evolves more slowly compared to Spring Boot.

13. What differentiates Spring Data JPA and Hibernate?

Hibernate, a Java Persistence API (JPA) implementation, enables Object-Relational Mapping (ORM) by allowing users to store, retrieve, map, and change application data across Java objects and relational databases.	Spring Data JPA, a Spring Data sub-project, provides abstraction over the DAL (Data Access Layer) by combining JPA and Object-Relational Mapping implementations like Hibernate.
Hibernate converts Java data types to SQL (Structured Query Language) data types and Java classes to database tables, allowing developers to avoid scripting data persistence in SQL applications.	Spring Data JPA makes it easier to construct JPA repositories and aims to improve DAL implementation significantly.
It provides full control over the persistence logic, query language (HQL - Hibernate Query Language), caching, and transaction management.	It simplifies the data access layer by reducing boilerplate code and automating common tasks like CRUD operations and pagination.
Querying in Hibernate can be done using HQL, Criteria API, or native SQL.	It allows for custom queries using JPQL, native queries, or by using @Query annotations
Transaction management is done manually by using the Session API.	Integrates with Spring's declarative transaction management using @Transactional annotation, simplifying transaction management.
You need to manually handle a lot of boilerplate code like opening/closing sessions, transaction management, and CRUD operations unless you use additional tools	Reduces boilerplate code significantly. You define repository interfaces, and Spring Data JPA automatically provides implementations at runtime.
Requires explicit transaction handling using session.beginTransaction() and session.getTransaction().commit()).	Transactions are automatically handled, and you can apply transactions to methods without writing explicit transaction code.

14 What is the difference between prototype and singleton scopes in the context of Spring Boot beans?

	prototype
In the **singleton** scope, the Spring container creates only **one instance** of the bean for the entire Spring Application Context.	In the **prototype** scope, a **new instance** of the bean is created **every time.**
This single instance is shared and reused across the whole application whenever the bean is requested.	It creates for each time when requested from the Spring Application Context.
When a bean is marked as a singleton (or if no scope is specified, since this is the default), Spring creates the bean only once and stores it in the application context. Every subsequent request for that bean will return the same instance.	When a bean is marked as a prototype, Spring does not manage its full lifecycle. It only creates a new instance when the bean is requested, but it does not handle the destruction or any subsequent lifecycle events of the bean. Each request for the bean results in a fresh instance.
Singleton beans are ideal when you need a single shared instance of a component across the entire application, like services or DAOs that don't maintain any state specific to a particular request or user session.	Prototype beans are suitable when you need a new, separate instance every time, such as when the bean maintains state specific to a particular request or when each instance has a short-lived lifecycle (e.g., session-scoped components, temporary objects, etc.).

For Example:
(©Service public class MyService
{
// This is a singleton bean by default
}
In this example, MyService will be created once and shared across the entire application.
	For Example:
	@Scope("prototype")
	(©Component
	public class MyPrototypeBean {
	// A new instance is created each time the bean is requested
	}
	In this example, MyPrototypeBean will get a new instance every time it is requested from the Spring container.

15. How Many Layers Does Spring Boot Have?

*Spring Boot consists of **four layers,*** and they are as follows:

- **Presentation Layer:** this is the *top layer* and it is *responsible for handling user interactions and presenting data to the user.* This layer typically includes components such as controllers, views, and templates. Along with *handling HTTP requests,* it also manages *authentication* and *JSON conversions.*
- **Business Layer:** this layer in a Spring Boot application is *responsible for implementing the application's core business logic and taking care of validation and authorization.* This layer typically includes components such as services, repositories, and domain objects.
- **Persistence Layer:** this layer is *responsible for storing and retrieving data from a data store, such as a database,* and it typically includes components such as data sources, entity classes, and repositories.
- **Database Layer:** it is the *last layer* in a Spring Boot application and it includes all databases such as MySql, MongoDB, and others. This database layer is *in charge of carrying out CRUD operations* and may contain several databases.

16. What is the purpose of Spring Boot DevTools?

The purpose of Spring Boot DevTools is to enhance the development experience when working with Spring Boot applications. DevTools provides a range of features designed to improve productivity and efficiency. DevTools includes automatic restart for any changes in the project, which eliminates the need to manually restart the application during development. This feature is particularly useful for seeing changes in real-time without disrupting the development flow.

Spring Boot DevTools also offers additional features such as default property settings for a more streamlined development environment and remote application support for easier access and management. The tool improves the load time of applications by enabling caching, which is crucial for quick development iterations. Developers expect a more intuitive and responsive development process with Spring Boot DevTools, especially in large and complex projects.

17. What are the different types of Bean scopes in Spring Boot?

In Spring Boot, there are different types of Bean scopes available. These are:

- Singleton: This is the default scope in Spring Boot. A singleton bean is created only once in the container, and the same instance is returned everytime the bean is requested.
- Prototype: This scope creates a new instance of the bean every time it is requested.
- Request: This scope creates a new instance of the bean for every HTTP request.
- Session: This scope creates a new instance of the bean for every HTTP session.
- Global Session: This scope creates a new instance of the bean for every global HTTP session.

18. What is a Spring Boot starter POM? Why is it useful?

Starters are a set of convenient dependency descriptors that you can include in your application. The starters contain a lot of the dependencies that you need to get a project up and running quickly and with a consistent, supported set of managed transitive dependencies.

The starter POMs are convenient dependency descriptors that can be added to your application's Maven. In simple words, if you are developing a project that uses Spring Batch for batch processing, you just have to include spring-boot-starter-batch that will import all the required dependencies for the Spring Batch application. This reduces the burden of searching and configuring all the dependencies required for a framework.

20. Differentiate between RequestMapping and GetMapping.

RequestMapping	**@GetMapping**
Used to map web requests to specific handler classes and methods	It not used to map onto specific handler classes
It maps both GET and POST requests	It maps only GET requests
It is class level annotation	It uses get mapping annotation rather than requestmapping
Used to configure base path and not for any further levels.	It is used to overcome drawbacks of (©RequestMapping
Developers avoid using this annotation due to its complexity and lengthy code	It has less code length compared to (©RequestMapping