Advance Java Imp

Q1. Short Answer Type Questions.

1. What is a servlet?

A servlet is a Java programming language class that is used to extend the capabilities of a server. Servlets run on a web server and handle requests and responses in a web application. They are primarily used to create dynamic web content, process form data, and manage sessions. Serv lets operate within the Java EE framework and are an essential part of Java web applications.

2. What is a protocol?

A protocol is a set of rules and conventions for communication between network devices. It defines how data is transmitted, formatted, and processed, ensuring that devices can understand each other. Common protocols include HTTP, FTP, and TCP/IP.

3. What is a socket?

A socket is an endpoint for sending and receiving data across a network. It is a combination of an IP address and a port number, allowing communication between applications over a network. Sockets can be used for both connection-oriented (TCP) and connectionless (UDP) communication.

4. What is TCP/IP?

TCP/IP (Transmission Control Protocol/Internet Protocol) is a suite of communication protocols used to interconnect network devices on the internet. TCP ensures reliable transmission of data, while IP handles addressing and routing. Together, they form the foundation of internet communication.

5. What is the sleep() method?

The sleep() method in Java is a static method of the Thread class that pauses the execution of the current thread for a specified period. It is used to delay the execution of a thread, allowing other threads to execute or to control the timing of operations.

6. What are the types of ResultSet?

ResultSet is an interface in JDBC that represents the result set of a query. The types of ResultSet include:

- **TYPE_FORWARD_ONLY**: The cursor can only move forward.
- **TYPE_SCROLL_INSENSITIVE**: The cursor can move both forward and backward, but does not reflect changes made to the database after the ResultSet was created.
- **TYPE_SCROLL_SENSITIVE**: The cursor can move both forward and backward and reflects changes made to the database.

7. List any 4 implicit objects of JSP.

JSP provides several implicit objects that are available for use in JSP pages. Four of these are:

- **request**: Represents the HTTP request.
- **response**: Represents the HTTP response.
- **session**: Represents the session for the user.
- **application**: Represents the servlet context for the web application.

8. What is Hibernate?

Hibernate is an object-relational mapping (ORM) framework for Java that simplifies database interactions. It allows developers to work with Java objects instead of SQL queries, providing a more intuitive way to manage database operations. Hibernate handles the mapping of Java classes to database tables and supports various database operations.

9. What is a port?

A port is a numerical identifier in networking that is used to distinguish different services or applications running on a device. Ports allow multiple applications to use the network simultaneously without interference. Common port numbers include 80 for HTTP and 443 for HTTPS.

10. What is a session?

A session is a temporary and interactive information interchange between two or more communicating devices, or between a user and a server. In web applications, a session is used to maintain state and store user-specific data across multiple requests.

11. Write down 2 methods of the Connection interface.

Two methods of the Connection interface in JDBC are:

- **createStatement()**: Creates a Statement object for sending SQL statements to the database.
- **prepareStatement(String sql)**: Creates a PreparedStatement object for sending precompiled SQL statements to the database.

12. What is the method to set the thread priority?

The method to set the thread priority in Java is **setPriority(int newPriority)**, which is a method of the Thread class. It allows you to specify the priority of a thread, which can affect the order in which threads are scheduled for execution.

13. Write down 2 classes used in socket programming.

Two classes commonly used in socket programming are:

- **Socket**: Represents a client socket that connects to a server.
- **ServerSocket**: Represents a server socket that listens for incoming client connections.

14. What is an IP address?

An IP address is a unique identifier assigned to each device connected to a network that uses the Internet Protocol for communication. It serves two main functions: identifying the host or network interface and providing the location of the device in the network.

15. What is the doGet() method of a servlet?

The doGet() method is a part of the HttpServlet class in Java and is called by the server to handle HTTP GET requests. It processes the request and generates a response, typically used for retrieving data from the server.

16. What are the parameters of the service() method of a servlet?

The service() method of a servlet takes two parameters:

- **HttpServletRequest request**: Represents the request from the client.
- **HttpServletResponse response**: Represents the response to be sent to the client.

### 17. What is JSP?

JavaServer Pages (JSP) is a technology used for developing web pages that include dynamic content. JSP allows developers to embed Java code directly into HTML pages, enabling the creation of interactive web applications. It is built on top of the Java Servlet API and is compiled into servlets by the server.

18. What is the getConnection method?

The getConnection method is a static method of the DriverManager class in JDBC that establishes a connection to a specified database. It takes parameters such as the database URL, username, and password, and returns a Connection object that can be used to interact with the database.

19. What is the use of cookies?

Cookies are small pieces of data stored on the client-side by the web browser. They are used to remember user preferences, session information, and track user behavior across sessions. Cookies enhance the user experience by allowing websites to retain information about users.

20. What is the use of the Runnable interface?

The Runnable interface is used to define a task that can be executed by a thread. It contains a single method, **run()**, which contains the code that defines the task. Implementing the Runnable interface allows for more flexible thread management compared to extending the Thread class.

21. Explain thread priority.

Thread priority is a way to indicate the relative importance of a thread compared to others. In Java, thread priorities range from 1 (lowest) to 10 (highest), with the default priority being 5. The thread scheduler uses these priorities to determine the order in which threads are executed, although it does not guarantee execution order.

22. What is the use of HQL?

Hibernate Query Language (HQL) is an object-oriented query language used in Hibernate to perform database operations. HQL allows developers to write queries using Java objects and their properties instead of SQL tables and columns, making it easier to work with the object-oriented paradigm.

23. What are the directives in JSP?

Directives in JSP are special instructions that provide global information about an entire JSP page. Common directives include:

- **page**: Defines page-level attributes such as content type and error handling.
- **include**: Includes another file at the time the JSP is compiled.
- **taglib**: Declares a tag library for custom tags.

24. What is networking?

Networking refers to the practice of connecting computers and other devices to share resources and information. It involves the use of protocols, hardware, and software to facilitate communication between devices over local or wide area networks.

25. Write the method for creating a connection.

To create a connection in JDBC, you typically use the following method:

java

RunCopy code

1Connection conn = DriverManager.getConnection(url, username, password);

This method establishes a connection to the database specified by the URL using the provided username and password.

26. What is the yield() method?

The yield() method is a static method of the Thread class that suggests to the thread scheduler that the current thread is willing to yield its current use of the CPU. It allows other threads of the same priority to execute, potentially improving the responsiveness of the application.

27. What is the use of the socket class?

The Socket class in Java is used to create client-side sockets for connecting to a server. It provides methods for sending and receiving data over the network, enabling communication between client and server applications.

28. What is JDBC?

Java Database Connectivity (JDBC) is an API that allows Java applications to interact with databases. It provides a standard interface for connecting to relational databases, executing SQL queries, and retrieving results, enabling developers to build database-driven applications.

29. What is a scriptlet tag?

A scriptlet tag in JSP is used to embed Java code within HTML. It is enclosed within **<%** and **%>** tags, allowing developers to write Java code that can be executed on the server side before the page is sent to the client.

30. What are wait() and notify() in multithreading?

The wait() and notify() methods are used for inter-thread communication in Java.

- **wait()**: Causes the current thread to wait until another thread invokes notify() or notifyAll() on the same object.
- **notify()**: Wakes up a single thread that is waiting on the object's monitor. These methods are essential for coordinating actions between threads.

31. What is the DriverManager class?

The DriverManager class is part of the JDBC API and manages a list of database drivers. It is responsible for establishing connections to databases by selecting the appropriate driver based on the connection URL provided.

32. What is ORM?

Object-Relational Mapping (ORM) is a programming technique used to convert data between incompatible type systems in object -oriented programming languages. ORM allows developers to interact with a database using high-level programming constructs, such as objects, rather than low-level SQL queries. This abstraction simplifies database operations and enhances productivity by allowing developers to work with familiar object-oriented paradigms.

33. What is the role of PreparedStatement?

PreparedStatement is an interface in JDBC that represents a precompiled SQL statement. It allows developers to execute parameterized queries, which can improve performance and security by preventing SQL injection attacks. PreparedStatement also provides methods for setting parameters and executing queries efficiently.

34. What is UDP?

User Datagram Protocol (UDP) is a connectionless communication protocol used for transmitting data over a network. Unlike TCP, UDP does not establish a connection before sending data and does not guarantee delivery, order, or error checking. It is often used for applications that require fast transmission, such as video streaming and online gaming.

35. What is a thread?

A thread is a lightweight process that can run concurrently with other threads within a program. Threads share the same memory space and resources, allowing for efficient execution of tasks. In Java, threads can be created by extending the Thread class or implementing the Runnable interface, enabling multitasking and improved application performance.

Q2. Long Answer Type Questions.
1. Explain JSP Life Cycle with diagram.

The JSP life cycle consists of several phases:

1. **Translation**: The JSP file is translated into a servlet by the server.
2. **Compilation**: The generated servlet is compiled into bytecode.
3. **Loading**: The servlet class is loaded into memory.
4. **Instantiation**: An instance of the servlet is created.
5. **Initialization**: The **init()** method is called to initialize the servlet.
6. **Request Handling**: The **service()** method is called to handle requests.
7. **Destruction**: The **destroy()** method is called before the servlet is removed from memory.
   (A diagram would typically illustrate these phases in a flowchart format.)

2. List & explain JDBC drivers.

JDBC drivers are software components that enable Java applications to interact with databases. There are four types of JDBC drivers:

1. **Type 1: JDBC-ODBC Bridge Driver**: Translates JDBC calls into ODBC calls. It is not recommended for production use due to performance issues.
2. **Type 2: Native-API Driver**: Converts JDBC calls into database-specific native calls. It requires native libraries to be installed on the client machine.
3. **Type 3: Network Protocol Driver**: Translates JDBC calls into a database-independent protocol, which is then converted to a database protocol by a server.
4. **Type 4: Thin Driver**: Pure Java driver that converts JDBC calls directly into the database-specific protocol. It is platform-independent and widely used.

3. Differentiate between doGet() and doPost() methods.

- **doGet()**: Handles HTTP GET requests. It is used to retrieve data from the server. Parameters are sent in the URL, making it suitable for non-sensitive data. It has limitations on the amount of data that can be sent (typically around 2048 characters). It is idempotent, meaning multiple identical requests will have the same effect as a single request.
- **doPost()**: Handles HTTP POST requests. It is used to send data to the server, such as form submissions. Data is sent in the request body, allowing for larger amounts of data to be transmitted. It is not idempotent, meaning multiple identical requests may have different effects.

4. Explain thread synchronization with example.

Thread synchronization is a mechanism that ensures that two or more concurrent threads do not simultaneously execute some particular program segment, which can lead to data inconsistency. In Java, synchronization can be achieved using the **synchronized** keyword.

Example:

```
class Counter {
  private int count = 0;
  public synchronized void increment() {
    count++;
  }
  public int getCount() {
    return count;
  }
}
```

In this example, the **increment()** method is synchronized, ensuring that only one thread can execute it at a time, preventing race conditions.

5. What are cookies? Explain the use of cookies in session tracking with example.

Cookies are small pieces of data stored on the client-side by the web browser. They are used to remember user preferences, session information, and track user behavior across sessions.

In session tracking, cookies can store a unique session ID that the server can use to identify the user's session.

Example:

```
Cookie userCookie = new Cookie("sessionId", "12345");
userCookie.setMaxAge(60*60); // 1 hour
response.addCookie(userCookie);
```

In this example, a cookie named "sessionId" is created and sent to the client, allowing the server to track the user's session.

6. Explain JSP directives in detail.

JSP directives provide global information about an entire JSP page and are defined using the **<%@ ... %>** syntax. There are three main types of directives:

1. **page**: Defines page-level attributes such as content type, error handling, and import statements.
   Example: **<%@ page contentType="text/html;charset=UTF-8" language="java" %>**
2. **include**: Includes another file at the time the JSP is compiled, allowing for code reuse.
   Example: **<%@ include file="header.jsp" %>**
3. **taglib**: Declares a tag library for custom tags, enabling the use of custom JSP tags in the page.
   Example: **<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>**

7. Explain Statement interface in detail along with PreparedStatement and CallableStatement.

The Statement interface is used to execute SQL queries against a database. It is suitable for executing simple SQL statements without parameters.
PreparedStatement is a sub-interface of Statement that allows for precompiled SQL statements with parameters. It improves performance and security by preventing SQL injection attacks.
CallableStatement is used to execute stored procedures in the database. It can accept input and output parameters, making it suitable for complex database operations.
Example:

```
PreparedStatement pstmt = connection.prepareStatement("SELECT * FROM users WHERE id = ?");
pstmt.setInt(1, userId);
ResultSet rs = pstmt.executeQuery();
```

8. Explain any 4 methods used in thread life cycle.

1. **start()**: Starts the execution of a thread. The thread moves from the NEW state to the RUNNABLE state.
2. **run()**: Contains the code that defines the thread's task. It is called when the thread is started.
3. **sleep(long millis)**: Causes the currently executing thread to sleep for a specified period, allowing other threads to execute.
4. **join()**: Waits for a thread to die. It allows one thread to wait for the completion of another thread.

9. Explain Socket and ServerSocket class in detail.

The Socket class is used to create client-side sockets for connecting to a server. It provides methods for sending and receiving data over the network. The ServerSocket class is used to create server-side sockets that listen for incoming client connections. It accepts client requests and establishes a connection.

Example:

```
// Server
ServerSocket serverSocket = new ServerSocket(8080);
Socket clientSocket = serverSocket.accept(); // Waits for a client connection

// Client
Socket socket = new Socket("localhost", 8080); // Connects to the server
```

10. What is Statement? Explain the types of statements in JDBC.

Statement is an interface in JDBC used to execute SQL queries against a database. There are three types of statements:

1. **Statement**: Used for executing simple SQL queries without parameters.
2. **PreparedStatement**: Used for executing precompiled SQL queries with parameters, improving performance and security.
3. **CallableStatement**: Used for executing stored procedures in the database, allowing for input and output parameters.

11. Explain thread life cycle with diagram.

The thread life cycle consists of several states:

1. **New**: The thread is created but not yet started.
2. **Runnable**: The thread is ready to run and waiting for CPU time.
3. **Blocked**: The thread is blocked waiting for a monitor lock.
4. **Waiting**: The thread is waiting indefinitely for another thread to perform a particular action.
5. **Timed Waiting**: The thread is waiting for a specified period.
6. **Terminated**: The thread has completed its execution.
   (A diagram would typically illustrate these states and transitions.)

12. What are the implicit objects in JSP? Explain any 4.

Implicit objects in JSP are predefined objects that are available for use in JSP pages without explicit declaration. Four common implicit objects are:

1. **request**: Represents the HTTP request made by the client. It contains data such as form parameters and request headers.
2. **response**: Represents the HTTP response that will be sent to the client. It allows setting response headers and content type.
3. **session**: Represents the session for the user, allowing storage of user-specific data across multiple requests.
4. **application**: Represents the servlet context for the web application, allowing access to application-wide parameters and attributes.

13. Explain the architecture of Hibernate.

Hibernate architecture consists of several components:

1. **Configuration**: Manages the configuration settings for Hibernate, including database connection details.
2. **SessionFactory**: A factory for creating Session objects, which are used to interact with the database.
3. **Session**: Represents a single unit of work with the database, allowing CRUD operations.

4. **Transaction**: Represents a database transaction, ensuring data integrity.
5. **Query**: Used to create and execute HQL or SQL queries.
6. **Entity**: Represents a persistent object that maps to a database table.

14. What is ResultSet interface in JDBC? Explain methods in ResultSet interface.

ResultSet is an interface in JDBC that represents the result set of a query. It provides methods to navigate and manipulate the data returned by a SQL query. Key methods include:

1. **next()**: Moves the cursor to the next row in the ResultSet.
2. **getString(int columnIndex)**: Retrieves the value of the specified column as a String.
3. **getInt(String columnLabel)**: Retrieves the value of the specified column as an int.
4. **close()**: Closes the ResultSet and releases any resources associated with it.

15. What are the states of the object in Hibernate?

In Hibernate, an entity can be in one of the following states:

1. **Transient**: The entity is created but not yet associated with a session or database.
2. **Persistent**: The entity is associated with a session and is being tracked by Hibernate. Changes to the entity are synchronized with the database.
3. **Detached**: The entity is no longer associated with a session but still exists in the database. Changes made to it will not be synchronized until it is reattached to a session.
4. **Removed**: The entity is marked for deletion from the database and will be removed upon the next transaction commit.

16. What is session tracking? What are the different ways of session tracking in servlets?

Session tracking is a mechanism that allows the server to maintain state and track user interactions across multiple requests. Different ways of session tracking in servlets include:

1. **Cookies**: Storing session IDs in cookies on the client-side.
2. **URL Rewriting**: Appending session IDs to URLs when cookies are not supported.
3. **Hidden Form Fields**: Including session IDs as hidden fields in HTML forms.
4. **HttpSession**: Using the HttpSession interface to store user-specific data on the server.

17. What is thread priority? How to set the thread priorities?
Thread priority is a way to indicate the relative importance of a thread compared to others. In Java, thread priorities range from 1 (lowest) to 10 (highest), with the default priority being 5. You can set the thread priority using the **setPriority(int newPriority)** method of the Thread class.
Example:

```
Thread thread = new Thread();
thread.setPriority(Thread.MAX_PRIORITY); // Set to highest priority
```

18. Explain inter-thread communication with an example.
Inter-thread communication is a mechanism that allows threads to communicate with each other and synchronize their actions. In Java, this can be achieved using the **wait()**, **notify()**, and **notifyAll()** methods.
Example:

```
class SharedResource {
  private int count = 0;

  public synchronized void increment() {
    count++;
    notify(); // Notify waiting threads
  }
  public synchronized void waitForCount(int target) throws
InterruptedException {
    while (count < target) {
      wait(); // Wait until notified
    }
    System.out.println("Count reached: " + count);
  }
}

public class InterThreadCommunicationExample {
  public static void main(String[] args) {
    SharedResource resource = new SharedResource();

    Thread incrementer = new Thread(() -> {
      for (int i = 0; i < 5; i++) {
        resource.increment();
        System.out.println("Incremented count to: " + (i + 1));
      }
    });

    Thread waiter = new Thread(() -> {
      try {
        resource.waitForCount(5);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    });

    waiter.start();
    incrementer.start();
  }
```

```
}
```

In this example, the **waitForCount** method waits until the count reaches a specified target, while the **increment** method notifies any waiting threads when the count is incremented.

19. Differentiate between Statement and PreparedStatement interface.

- **Statement**:
    - Used for executing simple SQL queries without parameters.
    - SQL statements are compiled and executed each time they are called, which can lead to performance overhead.
    - Vulnerable to SQL injection attacks as user input is directly included in the SQL query.
- **PreparedStatement**:
    - Used for executing precompiled SQL queries with parameters.
    - SQL statements are compiled once and can be executed multiple times with different parameters, improving performance.
    - Provides built-in protection against SQL injection by using parameterized queries.

20. Explain the life cycle of a thread.

The life cycle of a thread consists of several states:

1. **New**: The thread is created but not yet started.
2. **Runnable**: The thread is ready to run and waiting for CPU time.
3. **Blocked**: The thread is blocked waiting for a monitor lock.
4. **Waiting**: The thread is waiting indefinitely for another thread to perform a particular action.
5. **Timed Waiting**: The thread is waiting for a specified period.
6. **Terminated**: The thread has completed its execution.
   (A diagram would typically illustrate these states and transitions.)

21. Explain methods of ServerSocket class with syntax.

The ServerSocket class provides several methods for handling client connections:

1. **accept()**: Listens for a connection to be made to this socket and accepts it.

Socket clientSocket = serverSocket.accept();

2. **bind(SocketAddress addr)**: Binds the server socket to a specific address (IP and port).

serverSocket.bind(new InetSocketAddress("localhost", 8080));

3. **close()**: Closes the server socket and releases any resources associated with it.

serverSocket.close();

4. **setSoTimeout(int timeout)**: Sets a timeout for the server socket, after which an IOException will be thrown if no connection is made.

serverSocket.setSoTimeout(10000); // 10 seconds

22. What is the difference between execute(), executeQuery(), and executeUpdate()?

- **execute()**: Used for executing any SQL statement, including SELECT, INSERT, UPDATE, DELETE, and DDL statements. It returns a boolean indicating whether the result is a ResultSet or an update count.
- **executeQuery()**: Specifically used for executing SQL SELECT statements. It returns a ResultSet object containing the data retrieved from the database.
- **executeUpdate()**: Used for executing SQL statements that modify the database, such as INSERT, UPDATE, or DELETE. It returns an integer representing the number of rows affected by the operation.

23. Explain methods of Socket class with example.

The Socket class provides methods for client-side socket communication:

1. **getInputStream()**: Returns an input stream for reading data from the socket.

InputStream input = socket.getInputStream();

2. **getOutputStream()**: Returns an output stream for sending data to the socket.

OutputStream output = socket.getOutputStream();

3. **close()**: Closes the socket and releases any resources associated with it.

socket.close();

4. **connect(SocketAddress endpoint)**: Connects the socket to the specified remote address.

socket.connect(new InetSocketAddress("localhost", 8080));

24. Write advantages and disadvantages of Spring.

Advantages of Spring:

1. **Inversion of Control (IoC)**: Promotes loose coupling through dependency injection, making applications easier to manage and test.
2. **Aspect-Oriented Programming (AOP)**: Supports cross-cutting concerns like logging and transaction management, improving code modularity.
3. **Comprehensive Framework**: Provides a wide range of features, including data access, transaction management, and web development, reducing the need for multiple frameworks.
4. **Integration**: Easily integrates with other frameworks and technologies, such as Hibernate, JPA, and JMS.

Disadvantages of Spring:

1. **Complexity**: The learning curve can be steep for beginners due to the extensive features and configurations.
2. **Performance Overhead**: The use of reflection and proxies can introduce performance overhead in certain scenarios.
3. **Configuration**: XML or annotation-based configuration can become cumbersome in large applications, leading to potential misconfigurations.

25. Explain JSP tags with example.

JSP tags are used to embed Java code and control the behavior of JSP pages. There are several types of JSP tags:

1. **Standard Tags**: Built-in tags for common tasks, such as **<jsp:include>** for including other resources.
   Example: **<jsp:include page="header.jsp" />**
2. **Custom Tags**: User-defined tags that encapsulate reusable functionality.
3. **JSTL (JavaServer Pages Standard Tag Library)**: A collection of tags for common tasks like iteration and conditionals.
   Example:

```
<c:forEach var="item" items="${itemList}">
  <p>${item}</p>
</c:forEach>
```

26. What are the advantages and disadvantages of multithreading?

Advantages of Multithreading:

1. **Improved Performance**: Allows concurrent execution of tasks, leading to better resource utilization and faster application performance.
2. **Responsiveness**: Keeps applications responsive by allowing background tasks to run without blocking the main thread.
3. **Resource Sharing**: Threads share the same memory space, making communication between them more efficient than inter-process communication.

Disadvantages of Multithreading:

1. **Complexity**: Writing and debugging multithreaded programs can be challenging due to issues like race conditions and deadlocks.
2. **Overhead**: Context switching between threads can introduce overhead, potentially negating performance benefits in certain scenarios.
3. **Synchronization Issues**: Requires careful management of shared resources to avoid data inconsistency and ensure thread safety.

27. Explain life cycle of servlet.

The life cycle of a servlet consists of the following phases:

1. **Loading**: The servlet class is loaded into memory by the servlet container.
2. **Instantiation**: An instance of the servlet is created.
3. **Initialization**: The **init()** method is called to initialize the servlet, allowing it to perform any startup tasks.
4. **Request Handling**: The **service()** method is called to handle client requests. This method can call **doGet()**, **doPost()**, etc., based on the request type.
5. **Destruction**: The **destroy()** method is called before the servlet is removed from memory, allowing it to release resources and perform cleanup tasks.

28. Differentiate between HTTP servlet and Generic servlet.

- **HTTP Servlet**:
  - Extends **HttpServlet** and is specifically designed to handle HTTP requests.
  - Provides methods like **doGet()**, **doPost()**, **doPut()**, and **doDelete()** for handling different HTTP methods.
  - Automatically handles HTTP-specific features like session management and cookies.
- **Generic Servlet**:
  - Extends **GenericServlet** and is protocol-independent, meaning it can handle any type of request.
  - Requires the developer to implement the **service()** method to handle requests.
  - Less commonly used for web applications compared to HTTP servlets.

29. Explain life cycle of JSP with suitable diagram.

The life cycle of a JSP page consists of several phases:

1. **Translation**: The JSP file is translated into a servlet by the server.

2. **Compilation**: The generated servlet is compiled into bytecode.
3. **Loading**: The servlet class is loaded into memory.
4. **Instantiation**: An instance of the servlet is created.
5. **Initialization**: The **init()** method is called to initialize the servlet.
6. **Request Handling**: The **service()** method is called to handle requests, which may invoke **doGet()**, **doPost()**, etc.
7. **Destruction**: The **destroy()** method is called before the servlet is removed from memory, allowing for cleanup.
   (A diagram would typically illustrate these phases in a flowchart format.)

30. What is synchronization? Explain.
Synchronization is a mechanism that ensures that two or more concurrent threads do not simultaneously execute a particular section of code, which can lead to data inconsistency. In Java, synchronization can be achieved using the **synchronized** keyword, which can be applied to methods or blocks of code.
Example:
```
class Counter {
  private int count = 0;

  public synchronized void increment() {
    count++;
  }

  public int getCount() {
    return count;
  }
}
```
In this example, the **increment()** method is synchronized, ensuring that only one thread can execute it at a time, preventing race conditions.

31. What is multithreading? Explain.

Multithreading is a programming concept that allows multiple threads to run concurrently within a single process. Each thread represents a separate path of execution, enabling tasks to be performed simultaneously. This can lead to improved application performance and responsiveness, especially in applications that require parallel processing or handle multiple user requests. In Java, multithreading can be implemented by either extending the **Thread** class or implementing the **Runnable** interface. Threads share the same memory space, which allows for efficient communication but also requires careful management to avoid issues like race conditions and deadlocks.

32. What are the scripting elements of JSP?

JSP scripting elements allow developers to embed Java code within JSP pages. The main types of scripting elements are:

1. **Declarations**: Used to declare variables and methods that can be used in the JSP page. They are defined within **<%! ... %>** tags.
   Example:

```
<%! int count = 0; %>
```

2. **Scriptlets**: Used to embed Java code that is executed when the JSP page is requested. They are defined within **<% ... %>** tags.
   Example:

```
<% count++; %>
```

3. **Expressions**: Used to output data to the client. They are defined within **<%= ... %>** tags.
   Example:

```
<%= "Current count: " + count %>
```

33. What are the steps to connect to the database in Java?

To connect to a database in Java, follow these steps:

1. **Load the JDBC Driver**: Load the appropriate JDBC driver for the database.
   Example:

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

2. **Establish a Connection**: Use the **DriverManager.getConnection()** method to create a connection to the database.
   Example:

```
Connection connection = DriverManager.getConnection(url, username, password);
```

3. **Create a Statement**: Create a **Statement** or **PreparedStatement** object to execute SQL queries.
   Example:

```
Statement statement = connection.createStatement();
```

4. **Execute Queries**: Use the statement object to execute SQL queries and retrieve results.
   Example:

```
ResultSet resultSet = statement.executeQuery("SELECT * FROM your_table");
```

5. **Close the Connection**: Close the ResultSet, Statement, and Connection objects to release resources.
   Example:

```
resultSet.close();
statement.close();
connection.close();
```

34. Explain doGet() and doPost() methods.

The **doGet()** and **doPost()** methods are part of the HttpServlet class and are used to handle HTTP requests in servlets.

- **doGet()**:
  - Handles HTTP GET requests, typically used for retrieving data from the server.
  - Parameters are sent in the URL, making it suitable for non-sensitive data.
  - It is idempotent, meaning multiple identical requests will have the same effect as a single request.

Example:

```
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
   // Handle GET request
}
```

- **doPost()**:
  - Handles HTTP POST requests, typically used for sending data to the server, such as form submissions.
  - Data is sent in the request body, allowing for larger amounts of data to be transmitted.
  - It is not idempotent, meaning multiple identical requests may have different effects.

Example:

```
protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
   // Handle POST request
}
```

35. List & explain all the interfaces used in JDBC.**

JDBC provides several interfaces for interacting with databases. Key interfaces include:

1. **Driver**: This interface handles the communication with the database. It is implemented by the database vendor's JDBC driver.
2. **Connection**: Represents a connection to a specific database. It provides methods for creating statements, managing transactions, and closing the connection.
3. **Statement**: Used to execute SQL queries against the database. It is suitable for executing simple SQL statements without parameters.
4. **PreparedStatement**: A sub-interface of Statement that allows for precompiled SQL statements with parameters, improving performance and security.
5. **CallableStatement**: Used to execute stored procedures in the database, allowing for input and output parameters.

6. **ResultSet**: Represents the result set of a query. It provides methods to navigate and manipulate the data returned by a SQL query.
7. **ResultSetMetaData**: Provides information about the types and properties of the columns in a ResultSet.
8. **DatabaseMetaData**: Provides information about the database as a whole, such as its structure, capabilities, and supported features.

36. Differentiate between TCP socket and UDP socket.

- **TCP Socket**:
  - Connection-oriented protocol that establishes a reliable connection between the client and server before data transmission.
  - Ensures data integrity and order, making it suitable for applications that require reliable communication, such as web browsing and file transfers.
  - Uses a three-way handshake to establish a connection and provides error-checking and retransmission of lost packets.
- **UDP Socket**:
  - Connectionless protocol that sends data without establishing a connection.
  - Does not guarantee delivery, order, or error-checking, making it suitable for applications that prioritize speed over reliability, such as video streaming and online gaming.
  - Allows for faster data transmission but may result in lost or out-of-order packets.

Q3. Programs.

1. Write a Java program to display odd numbers between 1 to 100, with each number displayed after 2 seconds (multithreading).

Ans:-

```
public class OddNumbers {
    public static void main(String[] args) {
        for (int i = 1; i <= 100; i += 2) {
            System.out.println(i);
            try {
                Thread.sleep(2000); // Sleep for 2 seconds
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

2. Write a servlet application to display "Hello Java" message on the browser.

Ans:-

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        response.setContentType("text/html");
        response.getWriter().println("<h1>Hello Java</h1>");
    }
}
```

3. Write a JSP script to display the factorial of a given number.

Ans:-

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ page import="java.util.*" %>
<html>
<body>
  <form method="post">
    Enter a number: <input type="text" name="number">
    <input type="submit" value="Calculate Factorial">
  </form>
  <%
    String numberStr = request.getParameter("number");
    if (numberStr != null) {
      int number = Integer.parseInt(numberStr);
      long factorial = 1;
      for (int i = 1; i <= number; i++) {
        factorial *= i;
      }
 out.println("<h2>Factorial of " + number + " is " + factorial + "</h2>");
    }
  %>
</body>
</html>
```

4. Write a Java program to display the date of the server machine on the client's machine.

Ans:-

```
import java.io.IOException;
import java.io.PrintWriter;
import java.text.SimpleDateFormat;
import java.util.Date;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/date")
public class DateServlet extends HttpServlet {
   protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy
HH:mm:ss");
      Date date = new Date();
      out.println("<h1>Server Date and Time: " + formatter.format(date) +
"</h1>");
   }
}
```

5. Write a JDBC program to insert a record into the student table (Assume the student table is already created with columns rno, name, percentage).

Ans:-

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class InsertStudent {
   public static void main(String[] args) {
      String url = "jdbc:mysql://localhost:3306/yourdatabase";
      String user = "username";
      String password = "password";
      String query = "INSERT INTO student (rno, name, percentage) VALUES (?,
?, ?)";
```

```
        try (Connection conn = DriverManager.getConnection(url, user,
password);
             PreparedStatement pstmt = conn.prepareStatement(query)) {
            pstmt.setInt(1, 1);
            pstmt.setString(2, "John Doe");
            pstmt.setFloat(3, 85.5f);
            pstmt.executeUpdate();
            System.out.println("Record inserted successfully.");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

6. Write a Java program using multithreading to print "Hello Java" message for 500 times.

Ans:-

```
public class HelloJavaThread extends Thread {
    public void run() {
        for (int i = 0; i < 500; i++) {
            System.out.println("Hello Java");
        }
    }

    public static void main(String[] args) {
        HelloJavaThread thread = new HelloJavaThread();
        thread.start();
    }
}
```

7. Write a JSP application to accept a user name and greet the user.

Ans:-

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
```

```
<html>
<body>
  <form method="post">
    Enter your name: <input type="text" name="username">
    <input type="submit" value="Greet">
  </form>
  <%
    String username = request.getParameter("username");
    if (username != null) {
      out.println("<h2>Hello, " + username + "!</h2>");
    }
  %>
</body>
</html>
```

8. Write a Java program to display the IP address of a machine.

Ans:-

```
import java.net.InetAddress;
import java.net.UnknownHostException;

public class DisplayIP {
  public static void main(String[] args) {
    try {
      InetAddress ip = InetAddress.getLocalHost();
      System.out.println("IP Address: " + ip.getHostAddress());
    } catch (UnknownHostException e) {
      e.printStackTrace();
    }
  }
}
```

9. Write a Java program to print from 100 to 1 (use sleep() method).

Ans:-

```
public class PrintReverse {
    public static void main(String[] args) {
        for (int i = 100; i >= 1; i--) {
            System.out.println(i);
            try {
                Thread.sleep(1000); // Sleep for 1 second
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

10. Write a Java program to create a table named student with attributes Rno, Sname, Per.

Ans:-

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class CreateStudentTable {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/yourdatabase";
        String user = "username";
        String password = "password";
        String query = "CREATE TABLE student (Rno INT PRIMARY KEY, Sname
VARCHAR(50), Per FLOAT)";

        try (Connection conn = DriverManager.getConnection(url , user,
password);
             Statement stmt = conn.createStatement()) {
            stmt.executeUpdate(query);
            System.out.println("Table created successfully.");
        } catch (Exception e) {
```

```
        e.printStackTrace();
      }
   }
}
```
11. Write a Java program to count the number of records in a table.

Ans:-

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class CountRecords {
   public static void main(String[] args) {
      String url = "jdbc:mysql://localhost:3306/yourdatabase";
      String user = "username";
      String password = "password";
      String query = "SELECT COUNT(*) FROM student";

      try (Connection conn = DriverManager.getConnection(url, user,
password);
           Statement stmt = conn.createStatement();
           ResultSet rs = stmt.executeQuery(query)) {
         if (rs.next()) {
            System.out.println("Number of records: " + rs.getInt(1));
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```
12. Write a servlet program to accept two numbers from the user and print the addition of those numbers in blue color.

Ans:-

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/add")
public class AdditionServlet extends HttpServlet {
  protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
      int num1 = Integer.parseInt(request.getParameter("num1"));
      int num2 = Integer.parseInt(request.getParameter("num2"));
      int sum = num1 + num2;

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<h1 style='color:blue;'>Sum: " + sum + "</h1>");
  }
}
```

13. Write a JDBC program to display the details of employees (eno, ename, department, sal) whose department is 'Computer Application'.

Ans:-

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class DisplayEmployees {
  public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/yourdatabase";
        String user = "username";
        String password = "password";
        String query = "SELECT * FROM employees WHERE department =
'Computer Application'";

        try (Connection conn = DriverManager.getConnection(url, user,
password);
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(query)) {
            while (rs.next()) {
                System.out.println("ENO: " + rs.getInt("eno") + ", Name: " +
rs.getString("ename") + ", Department: " + rs.getString("department") + ",
Salary: " + rs.getFloat("sal"));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```
14. Write a JSP program to accept Name & age of a voter and check whether he/she is eligible for voting or not.

Ans:-

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<html>
<body>
    <form method="post">
        Name: <input type="text" name="name"><br>
        Age: <input type="text" name="age"><br>
        <input type="submit" value="Check Eligibility">
    </form>
    <%
        String name = request.getParameter("name");
        String ageStr = request.getParameter("age");
        if (ageStr != null) {
            int age = Integer.parseInt(ageStr);
            if (age >= 18) {
                out.println("<h2>" + name + ", you are eligible to vote.</h2>");
            } else {
                out.println("<h2>" + name + ", you are not eligible to vote.</h2>");
            }
        }
    %>
</body>
</html>
```

15. Write a JDBC program to delete the records of employees whose names start with the character 'A'.

Ans:-

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class DeleteEmployees {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/yourdatabase";
        String user = "username";
        String password = "password";
        String query = "DELETE FROM employees WHERE ename LIKE 'A%'";

        try (Connection conn = DriverManager.getConnection(url, user,
password);
             PreparedStatement pstmt = conn.prepareStatement (query)) {
            int rowsAffected = pstmt.executeUpdate();
            System.out.println(rowsAffected + " records deleted successfully.");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```
16. Write a JSP program to calculate the factorial of a given number.

Ans:-

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<html>
<body>
    <form method="post">
        Enter a number: <input type="text" name="number">
        <input type="submit" value="Calculate Factorial">
    </form>
    <%
        String numberStr = request.getParameter("number");
        if (numberStr != null) {
            int number = Integer.parseInt(numberStr);
            long factorial = 1;
            for (int i = 1; i <= number; i++) {
                factorial *= i;
            }
            out.println("<h2>Factorial of " + number + " is " + factorial + "</h2>");
        }
    %>
</body>
</html>
```
17. Write a servlet program to display the details of employees in tabular format (Employee table structure: eno, ename, sal, design).

Ans:-

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/employeeDetails")
public class EmployeeDetailsServlet extends HttpServlet {
  protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<table
border='1'><tr><th>ENO</th><th>Name</th><th>Salary</th><th>Designatio
n</th></tr>");

    String url = "jdbc:mysql://localhost:3306/yourdatabase";
    String user = "username";
    String password = "password";
    String query = "SELECT * FROM employees";

    try (Connection conn = DriverManager.getConnection(url, user,
password);
       Statement stmt = conn.createStatement();
       ResultSet rs = stmt.executeQuery(query)) {
      while (rs.next()) {
            out.println("<tr><td>" + rs.getInt("eno") + "</td><td>" +
rs.getString("ename") + "</td><td>" + rs.getFloat("sal") + "</td><td>" +
rs.getString("design") + "</td></tr>");
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
      out.println("</table>");
   }
}
```

18. Write a JDBC program to delete the records of students whose names start with the character 'm'.

Ans:-

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class DeleteStudents {
   public static void main(String[] args) {
      String url = "jdbc:mysql://localhost:3306/yourdatabase";
      String user = "username";
      String password = "password";
      String query = "DELETE FROM student WHERE Sname LIKE 'm%'";

      try (Connection conn = DriverManager.getConnection(url, user,
password);
          PreparedStatement pstmt = conn.prepareStatement(query)) {
         int rowsAffected = pstmt.executeUpdate();
         System.out.println(rowsAffected + " records deleted successfully.");
      } catch (Exception e) {
         e.printStackTrace();
      }
```

```
  }
}
```

19. Write a JSP program to display all the prime numbers between 1 to n.

Ans:-

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<html>
<body>
  <form method="post">
    Enter a number: <input type="text" name="number">
    <input type="submit" value="Display Primes">
  </form>
  <%
    String numberStr = request.getParameter("number");
    if (numberStr != null) {
      int n = Integer.parseInt(numberStr);
      out.println("<h2>Prime numbers between 1 and " + n + ":</h2>");
      for (int i = 2; i <= n; i++) {
        boolean isPrime = true;
        for (int j = 2; j <= Math.sqrt(i); j++) {
          if (i % j == 0) {
            isPrime = false;
            break;
          }
        }
        if (isPrime) {
          out.print(i + " ");
        }
      }
    }
  %>
</body>
```

```
</html>
```
20. Write a JDBC program to insert a record into the patient table (Use prepared statement).

Ans:-

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class InsertPatient {
  public static void main(String[] args) {
     String url = "jdbc:mysql://localhost:3306/yourdatabase";
     String user = "username";
     String password = "password";
     String query = "INSERT INTO patient (id, name, age) VALUES (?, ?, ?)";

     try (Connection conn = DriverManager.getConnection(url, user,
password);
         PreparedStatement pstmt = conn.prepareStatement(query)) {
        pstmt.setInt(1, 1);
        pstmt.setString(2, "Jane Doe");
        pstmt.setInt(3, 30);
        pstmt.executeUpdate();
        System.out.println("Record inserted successfully into patient table.");
     } catch (Exception e) {
        e.printStackTrace();
     }
  }
}
```
21. Write a Java servlet program to accept a name from the user & display it on the browser (use HTML).

Ans:-

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/displayName")
public class DisplayNameServlet extends HttpServlet {
  protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
      String name = request.getParameter("name");
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<h1>Hello, " + name + "!</h1>");
  }
}
```

22. Write a Java program to delete the salary column from the Emp table (Assume the Emp table with attributes ENo, EName, and salary is already created).

Ans:-

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class DeleteSalaryColumn {
  public static void main(String[] args) {
      String url = "jdbc:mysql://localhost:3306/yourdatabase";
      String user = "username";
      String password = "password";
      String query = "ALTER TABLE Emp DROP COLUMN salary";
```

```
        try (Connection conn = DriverManager.getConnection(url, user,
password);
             Statement stmt = conn.createStatement()) {
            stmt.executeUpdate(query);
            System.out.println("Salary column deleted successfully from Emp
table.");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Q4. Short Notes.

1. Hibernate

Hibernate is an object-relational mapping (ORM) framework for Java that simplifies database interactions. It allows developers to work with Java objects instead of SQL queries, providing a more intuitive way to manage database operations. Hibernate handles the mapping of Java classes to database tables and supports various database operations, including CRUD (Create, Read, Update, Delete). It also provides features like caching, lazy loading, and transaction management, making it a popular choice for Java applications.

2. Connection Interface

The Connection interface in JDBC represents a connection to a specific database. It provides methods for creating statements, managing transactions, and closing the connection. Key methods include:

- **createStatement()**: Creates a Statement object for sending SQL statements to the database.
- **prepareStatement(String sql)**: Creates a PreparedStatement object for sending precompiled SQL statements to the database.

- **setAutoCommit(boolean autoCommit)**: Sets the auto-commit mode for the connection.
- **close()**: Closes the connection and releases any resources associated with it.

3. Thread Priorities

Thread priority is a way to indicate the relative importance of a thread compared to others. In Java, thread priorities range from 1 (lowest) to 10 (highest), with the default priority being 5. You can set the thread priority using the **setPriority(int newPriority)** method of the Thread class. Higher priority threads are more likely to be executed before lower priority threads, but the actual scheduling is determined by the thread scheduler and may vary across different platforms.

4. Runnable Interface

The Runnable interface is a functional interface in Java that represents a task that can be executed by a thread. It contains a single method, **run()**, which contains the code that defines the task. Implementing the Runnable interface allows for more flexible thread management compared to extending the Thread class, as it enables the separation of the task from the thread itself.
Example:
RunCopy code

```
public class MyRunnable implements Runnable {
   public void run() {
      System.out.println("Running in a separate thread.");
   }
}
```

5. Thread Synchronization

Thread synchronization is a mechanism that ensures that two or more concurrent threads do not simultaneously execute a particular section of

code, which can lead to data inconsistency. In Java, synchronization can be achieved using the **synchronized** keyword.
Example:

```
class Counter {
   private int count = 0;

   public synchronized void increment() {
      count++;
   }

   public int getCount() {
      return count;
   }
}
```

In this example, the **increment()** method is synchronized, ensuring that only one thread can execute it at a time, preventing race conditions.

6. Run Method

The **run()** method is defined in the Runnable interface and contains the code that is executed when a thread is started. When a thread is created using the Runnable interface, the **run()** method must be overridden to define the task that the thread will perform.
Example:

```
public class MyRunnable implements Runnable {
   public void run() {
      System.out.println("Thread is running.");
   }
}
```

7. Statement Interface

The Statement interface in JDBC is used to execute SQL queries against a database. It is suitable for executing simple SQL statements without parameters. Key methods include:

- **executeQuery(String sql)**: Executes a SQL SELECT statement and returns a ResultSet object.
- **executeUpdate(String sql)**: Executes a SQL INSERT, UPDATE, or DELETE statement and returns the number of affected rows.
- **close()**: Closes the Statement object and releases any resources associated with it.

8. HttpServlet

HttpServlet is a class in Java that extends the GenericServlet class and is specifically designed to handle HTTP requests. It provides methods such as **doGet()**, **doPost()**, **doPut()**, and **doDelete()** for handling different HTTP methods. HttpServlet is commonly used in web applications to process client requests and generate dynamic content.

9. Generic Servlet and HTTP Servlet

- **Generic Servlet**: A protocol-independent servlet that can handle requests of any type. It requires the developer to implement the **service()** method to handle requests. It is less commonly used for web applications compared to HTTP servlets.
- **HTTP Servlet**: A subclass of GenericServlet that is specifically designed to handle HTTP requests. It provides methods like **doGet()** and **doPost()** for handling different HTTP methods and automatically manages HTTP-specific features like session management and cookies.

10. Cookies

Cookies are small pieces of data stored on the client-side by the web browser. They are used to remember user preferences, session information, and track user activity across web pages. In Java, cookies can be created and managed

using the **javax.servlet.http.Cookie** class. Cookies can be set with attributes such as name, value, expiration time, and path. They are sent from the server to the client and then sent back to the server with subsequent requests.
Example:

```
Cookie cookie = new Cookie("username", "JohnDoe");
cookie.setMaxAge(3600); // 1 hour
response.addCookie(cookie);
```

11. ResultSet Interface
The ResultSet interface in JDBC represents the result set of a query. It provides methods to iterate through the rows of data returned by a SQL query. Key methods include:

- **next()**: Moves the cursor to the next row in the ResultSet.
- **getString(int columnIndex)**: Retrieves the value of the specified column as a String.
- **getInt(String columnLabel)**: Retrieves the value of the specified column as an int.
- **close()**: Closes the ResultSet and releases any resources associated with it.

12. notify(), notifyAll(), and wait()
These methods are part of Java's object synchronization mechanism. They are used for inter-thread communication.

- **wait()**: Causes the current thread to wait until another thread invokes **notify()** or **notifyAll()** on the same object.
- **notify()**: Wakes up a single thread that is waiting on the object's monitor.
- **notifyAll()**: Wakes up all threads that are waiting on the object's monitor. These methods must be called from a synchronized context.

Example:

```
synchronized (obj) {
   obj.wait(); // Wait for notification
   obj.notify(); // Notify waiting threads
}
```

13. Applications of Spring

The Spring Framework is a powerful framework for building Java applications. It provides comprehensive infrastructure support for developing Java applications, including:

- **Dependency Injection**: Simplifies the management of application components and their dependencies.
- **Aspect-Oriented Programming (AOP)**: Allows separation of cross-cutting concerns like logging and transaction management.
- **Spring MVC**: A web framework for building web applications following the Model-View-Controller pattern.
- **Spring Boot**: A project that simplifies the setup and development of new Spring applications with minimal configuration.