Module 9) Java – Software Design Patter and Project

- Introduction to Software Design Patterns:

Q-1 Definition and purpose of design patterns.

ANS - Definition of Design Patterns: A design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design. Design patterns are not finished designs that can be directly converted into code; instead, they are templates or guidelines that help to solve problems in a manner that is more efficient, maintainable, and scalable.

- Purpose of Design Patterns: The primary goals of design patterns are to:

1. Promote Reusability: Design patterns encourage the reuse of software structures, making it easier to create robust systems without reinventing the wheel for common problems.

2. Improve Code Maintainability: By using proven solutions, the code becomes easier to understand, modify, and extend, which is crucial for maintaining large and compleAx systems.

3. Enhance Communication: Design patterns provide a common language for developers to communicate effectively about software design decisions, making collaboration easier.

4. Increase Flexibility: Design patterns can be adapted to various situations, allowing developers to create systems that can change and scale with minimal changes to the existing codebase.

5. Minimize Redundancy: Using design patterns helps avoid duplicating code by leveraging well-known and tested solutions, which ultimately leads to more efficient software.

Q-2 Classification: Creational, Structural, and Behavioral patterns.

ANS:

1. Creational Patterns focus on object creation mechanisms (e.g., Factory, Singleton).

2. Structural Patterns focus on object composition and relationships between objects (e.g., Adapter, Composite, Decorator).

3. Behavioral Patterns focus on object interaction and responsibilities (e.g., Observer, Strategy, Command).

Q-3 Examples of popular patterns: Singleton, Factory, Observer, Decorator, etc.

ANS - 1. Singleton Pattern: Purpose: Ensures that a class has only one instance and provides a global point of access to it.

Example Use Case: Managing a shared resource like a configuration setting, logging, or database connection.

How it works: The Singleton pattern restricts instantiation of a class to a single instance and provides a method to retrieve that instance.

2. Factory Method Pattern: Purpose: Defines an interface for creating objects, but allows subclasses to alter the type of objects that will be created.

Example Use Case: Creating different types of objects (e.g., different shapes) but hiding the instantiation details.

How it works: A base class or interface defines a factory method, and subclasses implement this method to create instances of specific types.

3. Observer Pattern: Purpose: Defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

Example Use Case: Event handling systems, such as GUI frameworks, where multiple components need to listen to changes in a single object.

How it works: A subject maintains a list of its observers and notifies them of any state changes.

4. Decorator Pattern: Purpose: Allows for the dynamic addition of behavior to an object at runtime without affecting the behavior of other objects from the same class.

Example Use Case: Adding new features to objects, such as adding logging or validation to a function without modifying the original function.

How it works: The decorator wraps the original object to add new functionalities.

- Introduction to MVC Pattern:

Q-1 Model-View-Controller (MVC) architecture explained.

ANS - The Model-View-Controller (MVC) architecture is a design pattern widely used in software development to separate an application into three interconnected components, making it easier to manage, develop, and scale. Here's an explanation of the three core components:

1. Model: The Model represents the data and the business logic of the application. It is responsible for:

Data management: Interacting with databases, managing records, and processing data.

Business logic: Applying rules, calculations, or processing that define the operations in the app.

State: Storing the current state of the data and the application.The Model is independent of the user interface, meaning it doesn't need to know anything about how the data is presented to the user. It only knows about the data itself and how to manipulate or process it.

2. View: The View is the user interface (UI) of the application. It's responsible for displaying the data from the Model in a way that users can interact with. The View:

Presents data: Displays information to the user in a readable format (e.g., HTML, images, buttons).

Captures user input: It can include forms, buttons, or other UI elements to allow users to interact with the system.

The View doesn't interact directly with the Model's data or business logic; it relies on a Controller to update it.

3. Controller: The Controller acts as an intermediary between the Model and the View. It handles user input from the View, processes it (if needed), and updates the Model accordingly. It then updates the View to reflect any changes in the data. The Controller:

Handles user actions: When a user interacts with the View (e.g., clicking a button), the Controller receives the action and decides what should happen next.

Updates the Model: After processing user input or performing business logic, the Controller updates the Model.

Manages communication: It communicates between the Model and the View to ensure that the correct data is presented to the user.

Q-2 Separation of concerns and how MVC helps in structuring applications.

ANS - Separation of Concerns (SoC) is a design principle in software development that aims to organize a system by dividing it into distinct sections, each addressing a specific aspect or responsibility of the system. This helps in reducing complexity, making code more modular, and improving maintainability.

=> How MVC Helps in Structuring Applications: The Model-View-Controller (MVC) design pattern is a widely adopted software architectural pattern that is used to separate the concerns of an application into three distinct components.

1. Model: The Model represents the data and business logic of the application. It is responsible for handling the data, its structure, and its state. The Model interacts with the database or other data sources and performs operations like saving, updating, or deleting data.

This part deals with the application's core functionality, data, and logic, independent of the user interface or how it is presented.

2. View: The View is responsible for presenting the data from the Model to the user. It displays the user interface (UI) elements and can take the form of web pages, graphical user interfaces, or other types of visual outputs.

The View's concern is to display data to the user, but it should not contain business logic or directly manipulate data. It only focuses on rendering the output based on the state of the Model.

3. Controller: The Controller acts as an intermediary between the Model and the View. It handles user input, processes it (sometimes interacting with the Model), and updates the View accordingly. The Controller is responsible for processing user input, invoking appropriate changes to the Model, and updating the View. It doesn't deal with direct data storage or presentation.

- Introduction to Data Access Object (DAO):

1) Purpose of the DAO pattern in decoupling data access logic from business logic.

ANS - The DAO (Data Access Object) pattern is a structural pattern in software design that aims to decouple the data access logic from the business logic in an application. Here's how it serves that purpose:

1. Separation of Concerns: The primary purpose of the DAO pattern is to separate the concerns of how data is accessed and manipulated from the core business logic of the application.

Business logic focuses on implementing the application's core functionality (e.g., calculating values, making decisions).Data access logic focuses on interacting with the database or other storage systems (e.g., querying, inserting, updating, or deleting data).

2. Code Maintainability and Flexibility: With a DAO, changes in the data access technology (e.g., switching from a relational database to NoSQL or moving from one ORM framework to another) do not require changes in the business logic. Only the DAO layer needs to be updated.

3. Reusability: By centralizing the data access logic in DAOs, multiple parts of the application can reuse the same methods to access data, avoiding duplication and inconsistencies.

For example, if the application needs to access a "User" object in various places, the same DAO method (e.g., getUserById()) can be used wherever it's needed.

4. Testing: Decoupling the data access layer makes it easier to mock or stub the DAO during unit testing of business logic. You can simulate different data responses without needing access to an actual database.

5. Encapsulation of Data Access Operations:The DAO encapsulates the data access operations, abstracting the complexity of interacting with databases or other storage systems. This makes the rest of the application simpler and less concerned with the details of data retrieval or persistence.

6. Consistency and Standardization:

ANS - With a DAO, you can standardize how data is accessed across the application. For example, all database operations can follow a uniform interface and pattern, leading to better consistency and fewer errors.

2) How DAO works in combination with MVC to interact with databases.

ANS - DAO (Data Access Object) and MVC (Model-View-Controller) are patterns used to organize code and interact with data. Let’s break down how DAO works in combination with MVC to interact with databases:

The MVC architecture, the DAO pattern is typically used in the Model layer to interact with the database. Here’s how it works:

Model Layer (with DAO): The Model contains the business logic of the application. The DAO objects are used within the Model to retrieve and store data from/to the database.

The DAO abstracts the database interaction. For example, the DAO will contain methods for querying, inserting, updating, and deleting data from the database.

The Model will use the DAO to retrieve or persist data as needed.

Controller Layer: The Controller is responsible for handling user requests and deciding which Model to invoke. When the user performs an action (e.g., submitting a form), the Controller will interact with the Model.

The Controller might call the DAO methods through the Model to fetch or modify data in the database.

View Layer: The View displays data to the user. It doesn't directly interact with the database. It relies on the Controller to provide the data, which is typically fetched from the Model layer (via the DAO).

The View can display information returned by the Model layer after the Controller processes it. If the Controller updates data via DAO, the View reflects the updated data.

2. Session Management (Session, Cookie, Hidden Form Field, URL Rewriting):

- Session Management Overview:

Q-1 Why session management is essential in web applications.

ANS - Session management is essential in web applications for several key reasons, primarily related to user experience, security, and functionality. Here’s a breakdown of why it’s crucial:

Stateless Nature of HTTP: The HTTP protocol is stateless, meaning that each request from the client to the server is independent, and the server doesn't remember previous interactions. To provide a continuous and personalized user experience

User Authentication: When users log in, session management helps maintain their authentication state

Secure Login: Sessions help securely store authentication information, like user IDs or tokens, to validate users without asking for credentials with every request.

Access Control: Through session management, users can be granted or denied access to certain areas of the web application based on their role or permissions (admin, user, guest, etc.).

Preventing Unauthorized Access: Sessions can be protected through measures like session IDs, cookies, and token-based authentication (JWT). This ensures only authorized users can access restricted resources, preventing unauthorized access after initial login.

Session Expiry: Sessions can have an expiry time (e.g., after inactivity), helping to limit the window for potential attackers to hijack sessions.

Personalized User Experience:

Custom Preferences: By storing session data, web applications can offer personalized experiences (e.g., language preference, theme, items in a shopping cart).

Dynamic Content: Without session management, every page load would be like a fresh start. With sessions, a user can interact with dynamic content that remembers their previous actions.

Performance & Scalability:

Optimized Resource Usage: Server-side session storage allows applications to track active users and manage resources efficiently, such as limiting the number of concurrent sessions or serving personalized content.

Session Persistence: In cases where a user navigates away and returns, session management enables the server to “remember” them and maintain a seamless experience across different interactions or devices.

Support for Distributed Systems:

In modern web applications, session management helps with load balancing across multiple servers or cloud environments.

Q-2 Difference between client-side and server-side session management.

ANS - 1. Client-Side Session Management:

Storage Location: All session data is stored on the client's browser (on the user's device).

How It Works: The server generates a unique session identifier (session ID) and sends it to the client, often via cookies. The client sends this session ID back to the server with every subsequent request.

Data Security: The data stored on the client is not secure, as it can be manipulated or stolen (e.g., by accessing the cookies or storage).

Advantages:

Reduced Server Load: Since the session data is stored on the client, the server doesn't need to manage large amounts of session data for each user.

Scalability: As there's no need to store session data on the server, the system can scale more easily.

Disadvantages:

Limited Storage: There is limited space for storing data on the client (e.g., cookies have a size limit).

Security Risks: Sensitive data could be exposed or tampered with if not encrypted.

2. Server-Side Session Management:

Storage Location: All session data is stored on the server, often in memory or a database.

How It Works: When the client makes a request, it sends a session ID (often stored in a cookie or URL) to the server.

Data Security: The session data is stored securely on the server and not exposed to the client, making it safer from tampering or unauthorized access.

Advantages:

Security: Since session data is stored on the server, it is more secure and cannot be easily tampered with by the client.

Larger Data Storage: The server can handle larger session data because it's not limited by client storage constraints.

Disadvantages:

Server Load: The server has to manage and store session data for every user, which can increase the load, especially for large-scale applications.

Scalability: If the application is distributed across multiple servers, session data management can be complex and may require solutions like session replication or shared storage

- Session:

Q-1 Definition of a session and its importance in tracking user activity.

ANS - A session is a period of interaction between a user and a website or application, typically defined by the time between the user's arrival and departure. It starts when a user first visits a site or uses an app and ends when the user leaves, logs out, or remains inactive for a certain period of time.

In technical terms, a session often involves:

Start and end timestamps: The time a user begins and finishes their interaction.

Session ID: A unique identifier assigned to each session, often stored in a cookie or local storage.

User actions: Various activities during the session, like page views, clicks, or interactions with features.

Importance of Sessions in Tracking User Activity:

Personalization: Sessions allow websites to remember user preferences and behaviors during the interaction. This enables personalized content, such as recommending products or services based on prior actions within the same session.

Analytics: By tracking sessions, websites and apps can gather valuable data about user behavior, such as how long users stay on a site, which pages they visit, and what actions they take. This information helps businesses understand their audience better and improve user experiences.

Conversion Tracking: In e-commerce or lead generation, tracking sessions is critical for measuring conversions (e.g., purchases, form submissions). By associating a session with a user’s actions, businesses can analyze which steps led to successful outcomes and optimize their processes.

Session Duration & Engagement: Understanding how long users stay on a website or app helps measure engagement levels. A short session might indicate poor content or user experience, while a longer session suggests users are interacting with the site more deeply.

Security: Sessions play a role in security, particularly in ensuring that user authentication and authorization are maintained during the interaction. Tracking session data helps prevent unauthorized access or session hijacking.

Retargeting: Sessions help track users who visit a site but don't convert immediately. Marketers can use session data to create retargeting campaigns, aiming to bring those users back to the site for a future conversion.

Q-2 How to create, retrieve, and destroy sessions using Java servlets.

ANS - In Java servlets, managing sessions is essential for maintaining state between client requests, as HTTP is a stateless protocol. Sessions are commonly used to store user-specific data across multiple requests, such as login credentials, user preferences, and shopping cart information.

1. Creating a Session: To create a session, you use the HttpServletRequest object to call getSession() method. If the client has an existing session, it will return the existing session. If there is no existing session, it creates a new one.

Example : // Inside a servlet method (doGet or doPost)

```
HttpSession session = request.getSession(); // Creates a new session or retrieves an existing one

session.setAttribute("username", "john_doe"); // Store data in the session
```

2. Retrieving Session Data:

To retrieve session data, you use the getAttribute() method of HttpSession.

Example : // Retrieve session data

```
HttpSession session = request.getSession(false); // Get session if it exists

if (session != null) {

   String username = (String) session.getAttribute("username"); // Retrieve the "username" attribute

   // Use the data as needed

}
```

3. Destroying a Session:

To destroy a session, you can call invalidate() on the HttpSession object. This removes all attributes associated with the session and makes the session invalid.

Example : // Invalidate the session (destroy it)

```
HttpSession session = request.getSession(false);

if (session != null) {

   session.invalidate(); // Session is now invalid, and any data associated with it is lost

}
```

- Cookies:

Q-1 What cookies are and how they store small amounts of data on the client-side.

ANS: Cookies are small text files that are stored on a user's device (client-side) by a web browser. They are used to store data related to the user's interactions with websites, and this data can be retrieved and used on subsequent visits.

Creation of Cookies: When a user visits a website, the website can instruct the browser to create a cookie with specific data. This data is typically in the form of key-value pairs.

Storage on the Client-Side: The browser stores these cookies locally on the user's device, usually in a dedicated cookie storage area. These cookies remain stored on the device until they expire, are deleted by the user, or are overwritten by new cookies.

Data Transfer Between Server and Client: Each time the user makes a request to the website (for example, by navigating to a different page or refreshing the page), the browser sends the relevant cookies back to the server as part of the HTTP request headers.

Expiration of Cookies: Cookies can have an expiration date set by the server. If no expiration date is set, the cookie is considered a "session cookie" and will be deleted when the user closes the browser.

Q-2 Creating, reading, updating, and deleting cookies in Java servlets.

ANS - In Java servlets, cookies are commonly used to store small pieces of data on the client side. They can be created, read, updated, and deleted using the javax.servlet.http.Cookie class.

1. Creating Cookies

To create a cookie, instantiate a Cookie object with a name and a value. You can also set the cookie's expiration time (in seconds) using the setMaxAge(int seconds) method.

EX. import javax.servlet.*;

import javax.servlet.http.*;

import java.io.*;

public class CookieExampleServlet extends HttpServlet {

```
   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

      // Create a new cookie

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


      // Set cookie to expire in 1 day (86400 seconds)

      userCookie.setMaxAge(86400);
```

```
        // Add the cookie to the response
        response.addCookie(userCookie);

        // Respond to the client
        PrintWriter out = response.getWriter();
        out.println("Cookie has been set!");
    }
}
```

2. Reading Cookies

To read cookies, use the request.getCookies() method, which returns an array of Cookie objects. You can then iterate over the array to find the desired cookie by name.

EX.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class ReadCookieServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // Get the cookies from the request
        Cookie[] cookies = request.getCookies();

        if (cookies != null) {
            for (Cookie cookie : cookies) {
                if ("username".equals(cookie.getName())) {
                    String username = cookie.getValue();
                    // Output the username value from the cookie
                    PrintWriter out = response.getWriter();
                    out.println("Username from cookie: " + username);
                }
            }
        } else {
            PrintWriter out = response.getWriter();
```

```
            out.println("No cookies found.");
        }
    }
}
```

3. Updating Cookies

To update a cookie, you need to create a new Cookie object with the same name as the existing cookie, and then set a new value for it. After that, you can send the updated cookie to the client by calling response.addCookie().

EX.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class UpdateCookieServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // Create a cookie with the same name as the existing cookie to update it
        Cookie userCookie = new Cookie("username", "newJohnDoe");

        // Set the cookie to expire in 1 day (86400 seconds)
        userCookie.setMaxAge(86400);

        // Add the updated cookie to the response
        response.addCookie(userCookie);

        PrintWriter out = response.getWriter();
        out.println("Cookie has been updated!");
    }
}
```

4. Deleting Cookies

To delete a cookie, you can set its maximum age to 0 and then add it to the response. This will instruct the browser to remove the cookie.

EX. import javax.servlet.*;

import javax.servlet.http.*;

import java.io.*;

```
public class DeleteCookieServlet extends HttpServlet {
   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
      // Get the cookies from the request
      Cookie[] cookies = request.getCookies();

      if (cookies != null) {
         for (Cookie cookie : cookies) {
            if ("username".equals(cookie.getName())) {
               // Set the cookie's max age to 0 to delete it
               cookie.setMaxAge(0);
               response.addCookie(cookie);
               break;
            }
         }
      }

      PrintWriter out = response.getWriter();
      out.println("Cookie has been deleted.");
   }
}
```

- Hidden Form Fields:

Q-1 Explanation of hidden form fields and their role in passing data between pages.

ANS - Hidden form fields are a type of HTML input field that allows web developers to pass data between pages or between the client and server without the data being visible to the user. These fields are not displayed on the page but are included in the HTML form, enabling the transmission of data when the form is submitted.

Role of Hidden Form Fields:

1. Passing Data Between Pages: Hidden fields can store information that needs to be sent from one page to another, especially in scenarios where the user does not need to interact with that data. For instance, in multi-step forms or wizards, hidden fields can store values.

2. State Management: Hidden fields help maintain the state of the application between requests. For example, in an online shopping cart, hidden fields might store the product IDs and quantities of items the user has added to their cart, which are then submitted to the server when the user proceeds to checkout.

3. Preserving Session Data: Hidden fields are often used to store session data such as session IDs or authentication tokens. This allows the server to identify the user and retrieve the relevant data associated with their session when they submit the form.

4. Security Information: Hidden fields can be used to store information for security purposes, like CSRF (Cross-Site Request Forgery) tokens, which are used to verify that the request is legitimate and not from a malicious source.

5. Pre-populating Forms: When a form is displayed, hidden fields can be used to pre-populate data that the user has entered earlier, such as in a form that allows users to edit their profile. The hidden fields may store the user's original data or ID so it can be sent back to the server for processing.

EX. <form action="submit_page.php" method="POST">

```
  <input type="hidden" name="user_id" value="12345">
  <input type="hidden" name="session_token" value="abc123xyz">
  <input type="submit" value="Submit">
</form>
```

- URL Rewriting:

Q-1 How URL rewriting can be used to track sessions when cookies are disabled.

ANS - URL rewriting is a technique that can be used to track sessions in web applications when cookies are disabled or unavailable. It involves embedding session-related information directly in the URL of a webpage, rather than relying on cookies to store session data.

How URL Rewriting Works:

1. Session ID in URL: The server generates a unique session ID for each user (similar to a cookie's purpose) and appends this session ID to the URL of each page. This session ID is typically passed as a query string parameter or as part of the URL path.

Example: https://example.com/welcome?session_id=123456

Alternatively, the session ID can be part of the URL path: https://example.com/welcome/123456

2. Tracking User's Session: Every time a user requests a page on the website, the server retrieves the session ID from the URL, and uses it to identify the user's session and retrieve the corresponding session data (e.g., user preferences, authentication status, etc.).

3. Session Continuity: As the user navigates from page to page, the session ID is passed with each URL, ensuring that the user's session is maintained even if cookies are disabled. Every subsequent page load includes the session ID in the URL, which allows the server to recognize and track the session.