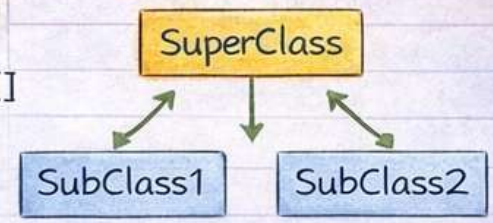
— THE COMPLETE —

JAVA NOTES

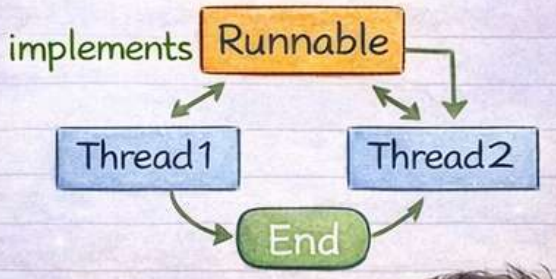
@curious_.programmer

```java
public class MyClass {
    public static void main(String[]
        [] args) {
        System.out.println("Hello,Java!");
    }
}
```

Inhentance

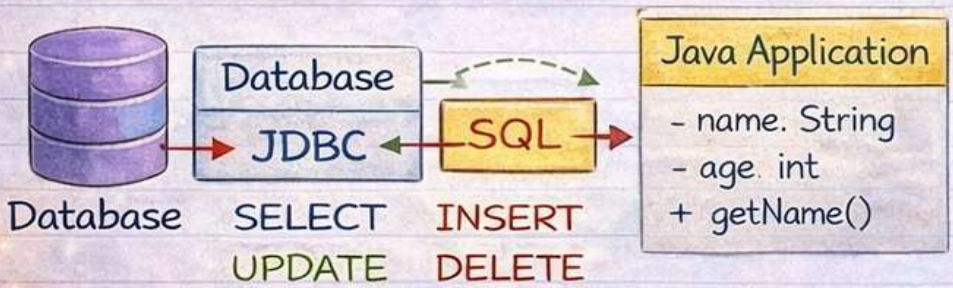


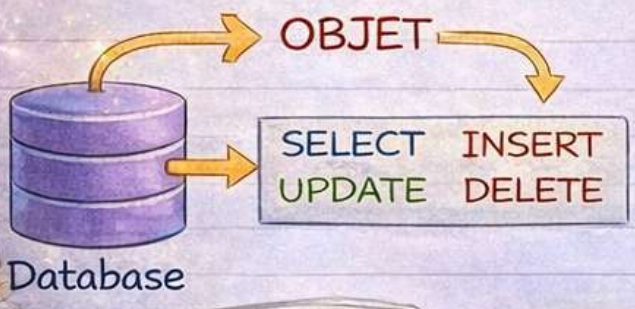
Interface



Exception Handling

```
try       1/0
catch     (ArithmeticException e)
finally   //clean up code;
```





Chapter 1: Java Database Connectivity (JDBC)

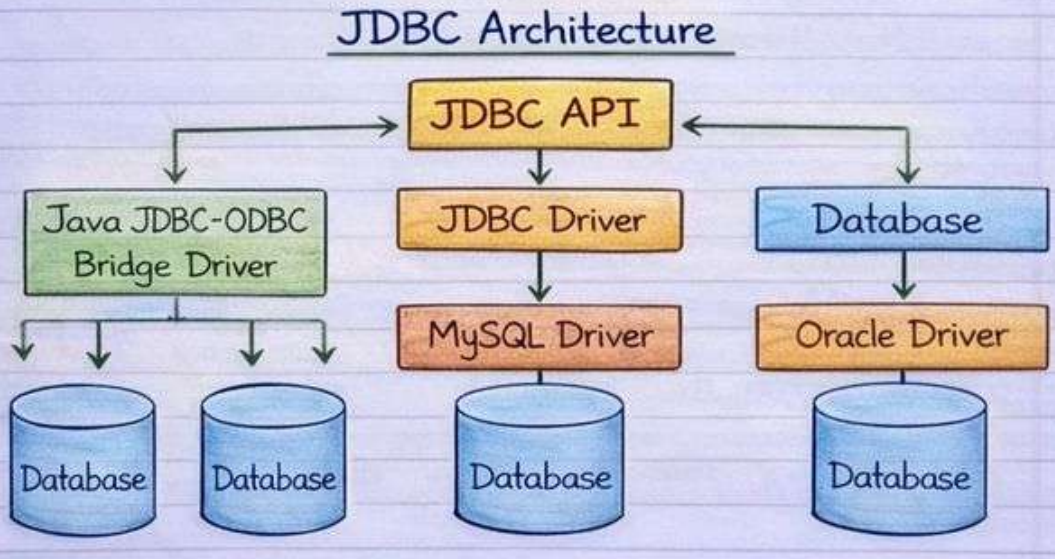
@curious_programmer

Introduction to JDBC

JDBC, or Java Database Connectivity, is a Java API that enables Java applications to interact with databases. It provides a standard set of interfaces that allow Java programs to execute SQL queries, update records, retrieve results, and interact with various relational database management systems in a database-independent manner.

JDBC Architecture

The JDBC architecture consists of several key components that work together to enable database connectivity for Java applications.

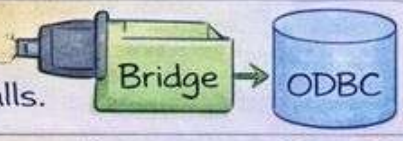
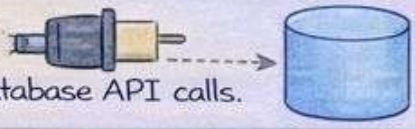
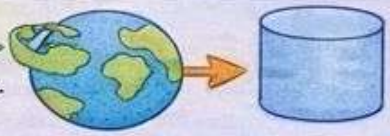
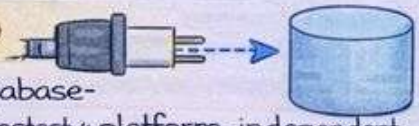


JDBC Drivers (Type 1-4)

@curious_programmer

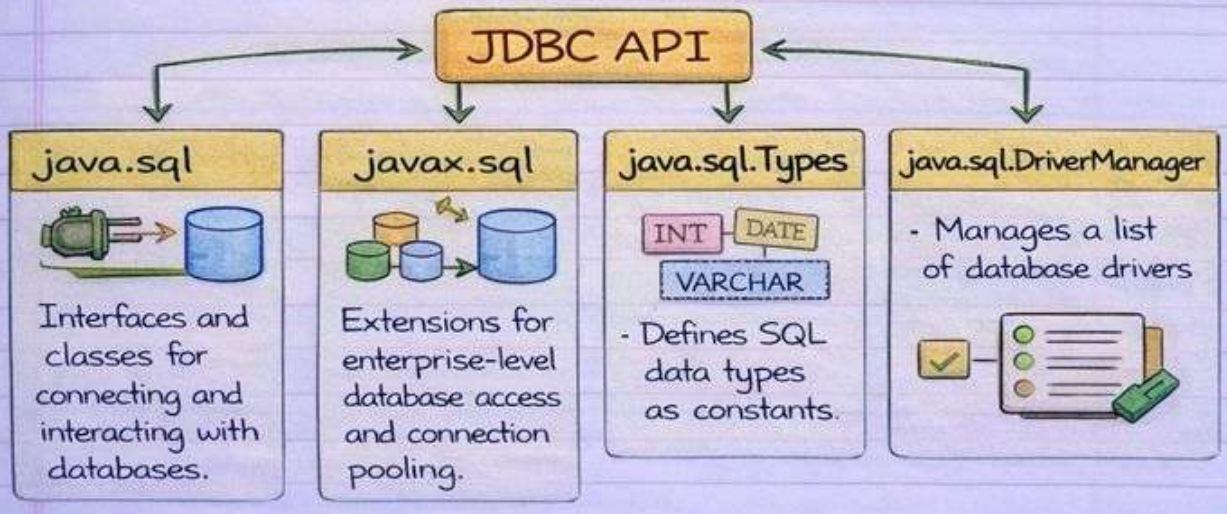
JDBC Drivers (Type 1-4)

JDBC drivers are used to connect Java applications to databases. There are four types of JDBC drivers, each with its own architecture.

Type 1	JDBC-ODBC Bridge Driver	
	- Converts JDBC calls into ODBC calls.	
Type 2	Native-API Driver	
	- Converts JDBC calls into native database API calls.	
Type 3	Network Protocol Driver	
	- Converts JDBC calls into a database-independent network protocol.	
Type 4	Thin Driver (Pure Java Driver)	
	- Directly converts JDBC calls to database-specific protocol using pure Java, fastest & platform-independent	

JDBC API & Packages

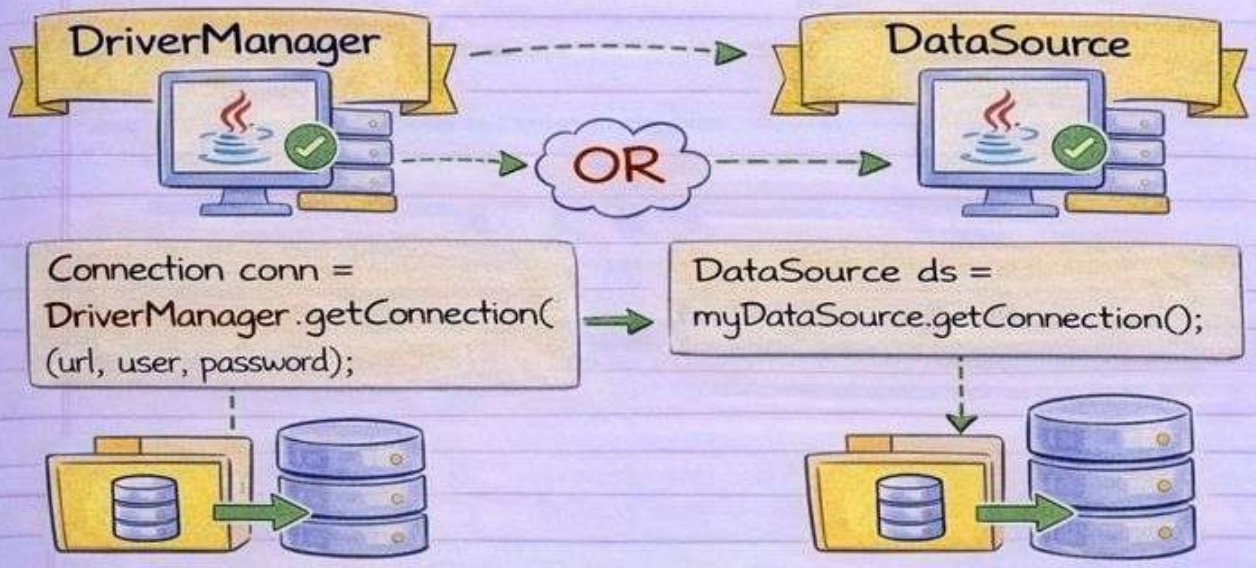
JDBC API provides a set of classes & interfaces allowing Java applications to connect and interact with databases.



@curious_programmer

Steps to Connect Java Application with Database: DriverManager and DataSource

@ curious_programmer




```
Connection conn =
DriverManager.getConnection(
(url, user, password);
```

```
DataSource ds =
myDataSource.getConnection();
```


Connection

- ✓ Represents a connection to the database.

```
Connection conn = . . ;
```

```
Statement stmt = conn.createStatement();
stmt.executeQuery("SELECT * FROM users");
```

PreparedStatement

- ✓ Precompiled SQL queries
- ✓ Uses placeholders (?) for parameters.

```
Statement stmt = conn.
createStatement("SELECT * "
                FROM users");
```

PreparedStatement

- ✓ Precompiled SQL queries
- ✓ Uses placeholders (?) for parameters.

```
PreparedStatement psmt =
 conn.prepareStatement("SELECT
      * FROM users WHERE id = ?);
pstmt.setInt(1, 123);
pstmt.executeQuery();
```

CallableStatement

- ✓ Used to call stored procedures
- ✓ Uses placeholders (?) for parameters.

```
CallableStatement cstmt =
conn.prepareCall("
cstmt.setInt(1, 123));
cstmt.execute();
```

@ curious_programmer

Executing SQL Queries

@ curious_programmer

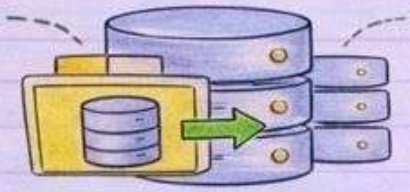
CRUD Operations using JDBC

Connection -- Create - Read - Update - Delete

```
Connection conn = DriverManager.getConnection(...);
```

Create

✓ Insert new records into the database.



Update

✓ Update existing records.

Create

✓ Insert new records into the database.

```
PreparedStatement psmt =
conn.prepareStatement("INSERT
    INTO users (name, email)
    VALUES (?, ?');

psmt.setString(1, "John");
psmt.setString(2, "john@example.com");
psmt.executeUpdate();
```

Read

✓ Retrieve records from the database.

```
Statement stmt = conn.
createStatement();
ResultSet rs = stmt.executeQuery
("SELECT * FROM users");

while (rs.next() {
 System.out.println(rs.getString("name
    + " + rs.getString("email"));
```

Update

✓ Update existing records.

```
PreparedStatement psmt =
conn.prepareStatement("UPDATE
    users SET email = ?) "
psmt.setString(1, "newemail
    @example.com");

psmt.executeUpdate();
```

Delete

✓ Delete records from the database.

```
PreparedStatement psmt =
conn.prepareStatement("DELETE FROM
    users WHERE id = ? );

psmt.setInt(1, 123);
psmt.executeUpdate();
```

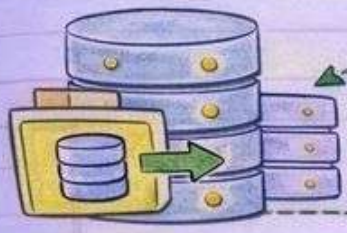
@ curious_programmer

ResultSet Types and Concurrency

@ curious_programmer

ResultSet Types

```
ResultSet rs = stmt.executeQuery("SELECT * FROM table");
```



TYPE_FORWARD_ONLY

- ✔ Default type, moves forward only

TYPE_SCROLL_INSENSITIVE

- ✔ Scrollable, does not reflect changes in database

TYPE_SCROLL_SENSITIVE

- ✔ Scrollable, reflects changes in the database

Batch Processing

Concurrency Levels

- ✔ Read-only access

```
Statement stmt = conn.createStatement();
stmt.addBatch("INSERT INTO users ...");
stmt.addBatch("UPDATE users SET ...");
stmt.addBatch("DELETE FROM users WHERE ...");
```

- ✔ stmt.executeBatch();

- ✔ Adds multiple SQL statements to a batch.
- ✔ Executes the batch with a single call.
- ✔ Improves performance by reducing the number of database calls.

@ curious_programmer

Stored Procedures

@ curious_programmer

Stored Procedures

```
CallableStatement cstmt = conn.prepareCall("
[?call getUserInfo(?)]");
```

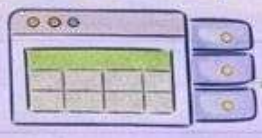
- ✓ Stored in the database
- ✓ Reusable database logic.
- ✓ Encapsulates SQL statements

Transaction Management

ACID Properties

- ✓ Atomic, Consistent, Isolated, Durable

BEGIN



COMMIT

BEGIN Transaction

- ✓ Conn.setAutoCommit(false);

```
conn.create Statement().executeUpdate("UPDATE users SET
    email = ? WHERE id = ?");
stmt.setString(1, "newemail@example.com");
stmt.setInt(2, 123);
```

BEGIN Transaction

- ✓ Conn.createStatement(),

```
conn.createStatement().executeUpdate("
    "UPDATE users SET email = ?");
    WHERE id = ?");
stmt.setString(1, "newemail@example.com");
stmt.setInt(2, 123);
```

COMMIT & ROLLBACK

- ✓ Commit saves the changes
- ✗ Rollback undoes the changes

```
conn.commit();
✗ conn.rollback();
```

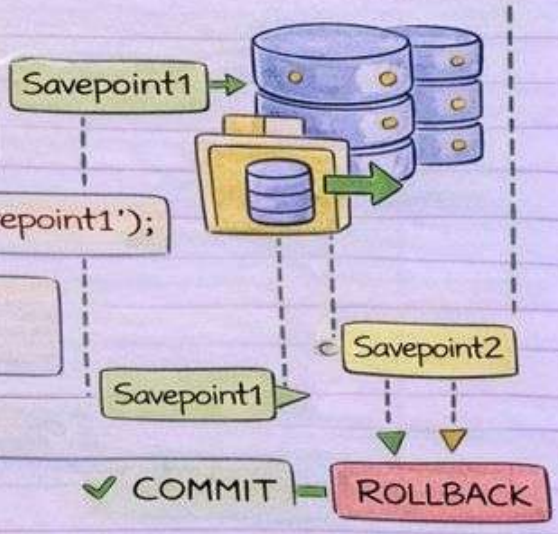
@ curious_programmer

Savepoints & Exception Handling in JDBC

@ curious_programmer

Savepoints

```
Connection conn = ...
conn.setAutoCommit(false);
Savepoint sp1 = conn.setSavepoint("Savepoint1");
stmt.executeUpdate("UPDATE users SET
  status = "inactive" WHERE id = 1);
Savepoint sp2 = conn.setSavepoint(');
✔ conn.rollback(sp1);
✔ stmt.executeUpdate("UPDATE users SET status = "inactive" WHERE id=2')
✔ conn.commit();
```



Exception Handling in JDBC

- ✔ Handling SQL exceptions.
- ✔ Handling database-specific SQL errors.
- ✔ Ensures proper resource cleanup.


```
try {
  Connection conn = DriverManager.getConnection(...);
  Statement stmt = conn.createStatement();
  ResultSet rs = stmt.executeQuery("SELECT * FROM users);
  catch (SQLException e) {
      System.err.println("Error: ' + e.getMessage');
  finally {
    if (conn == null) conn.close(); // Close connection ffapen
  }
```

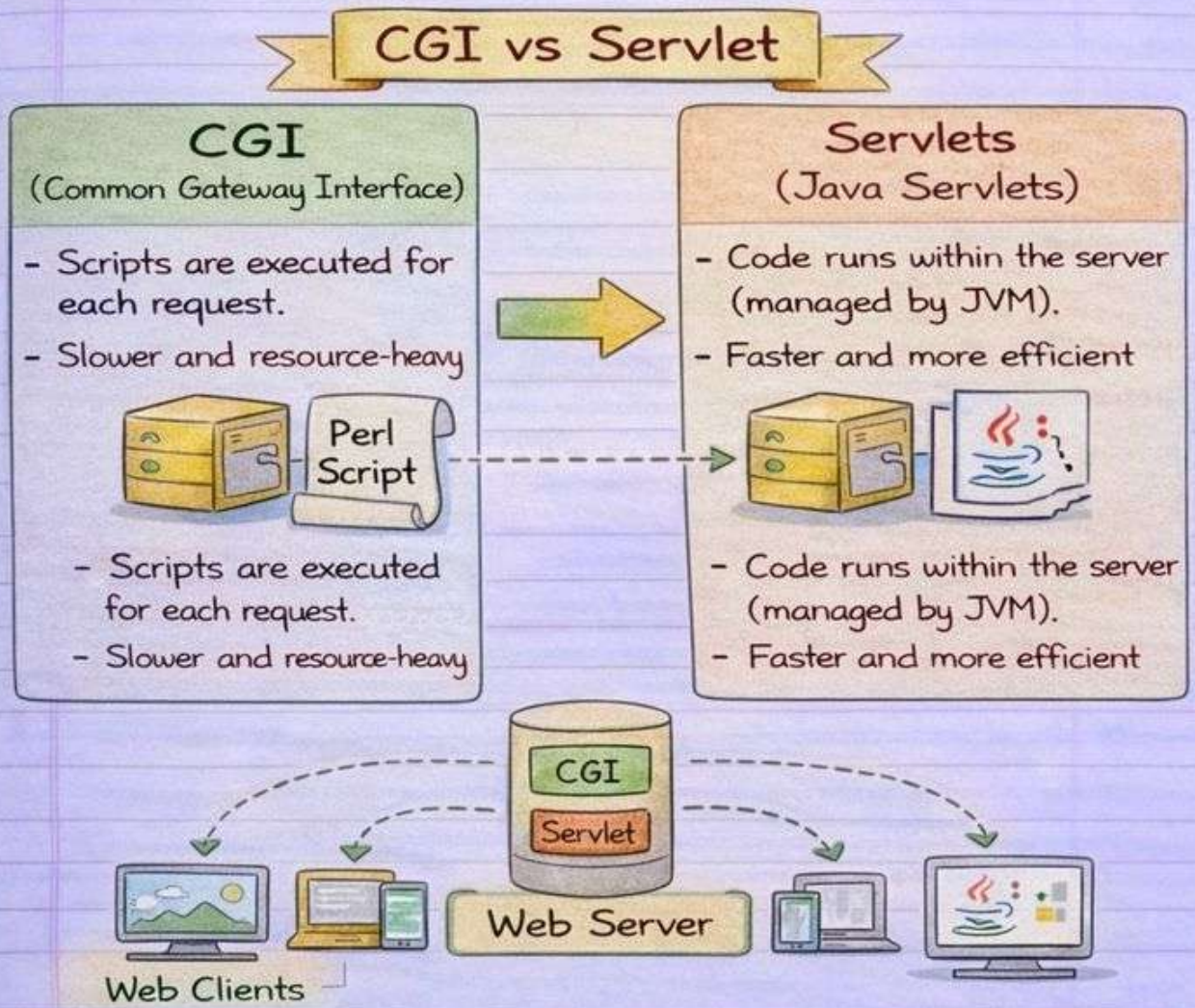
@ curious_programmer

Chapter 2: Java Servlets

@ curious_programmer

✔ Introduction to Web Applications

Web applications are dynamic websites that interact with users, often requiring data processing. Java Servlets help make web applications faster and more efficient by replacing CGI for handling requests.

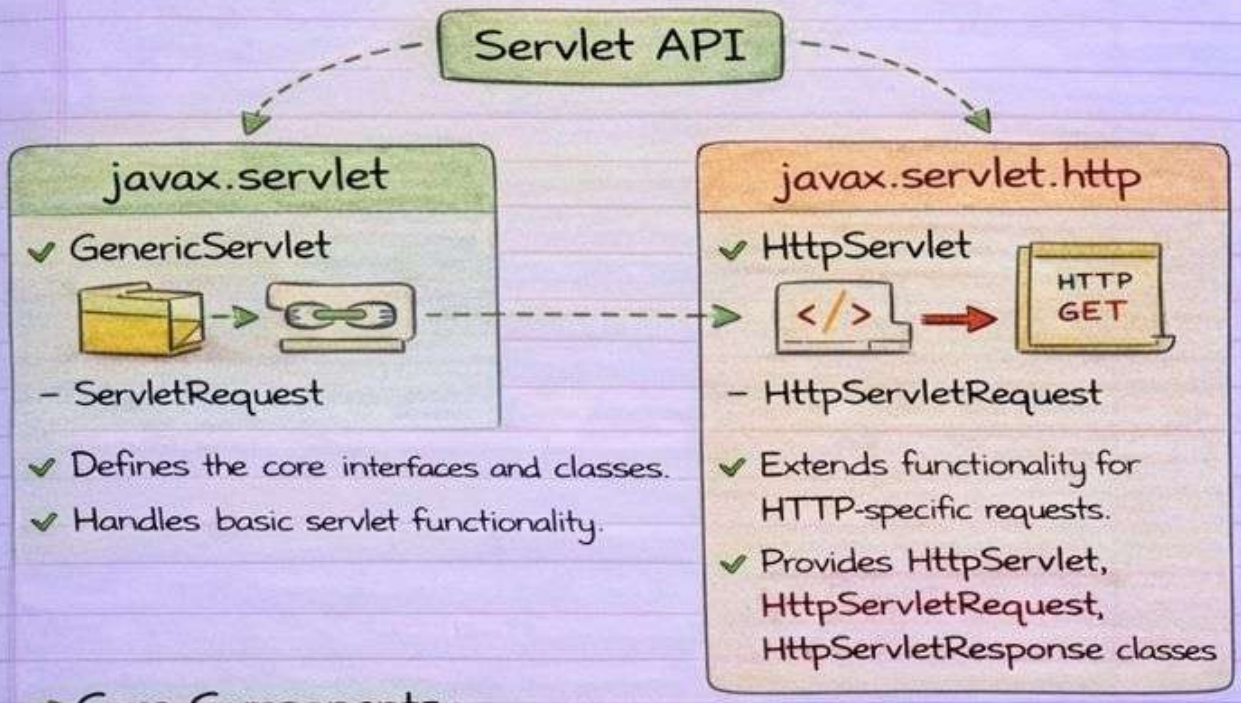


Servlet Architecture

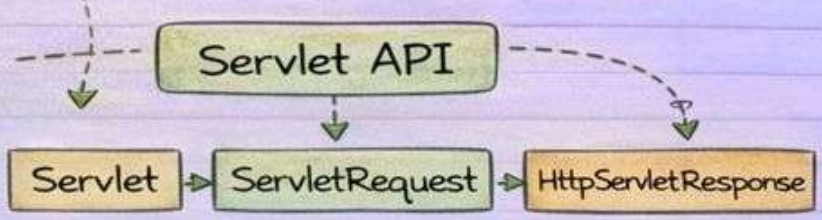
@curious_programmer

✓ Servlet API (javax.servlet, javax.servlet.http)

The Servlet API provides classes and interfaces for building web applications using Java. The main packages are javax.servlet and javax.servlet.http, which offer functionality for handling requests and responses in a server-side Java environment.



→ Core Components
- Servlet
- ServletRequest
- ServletResponse



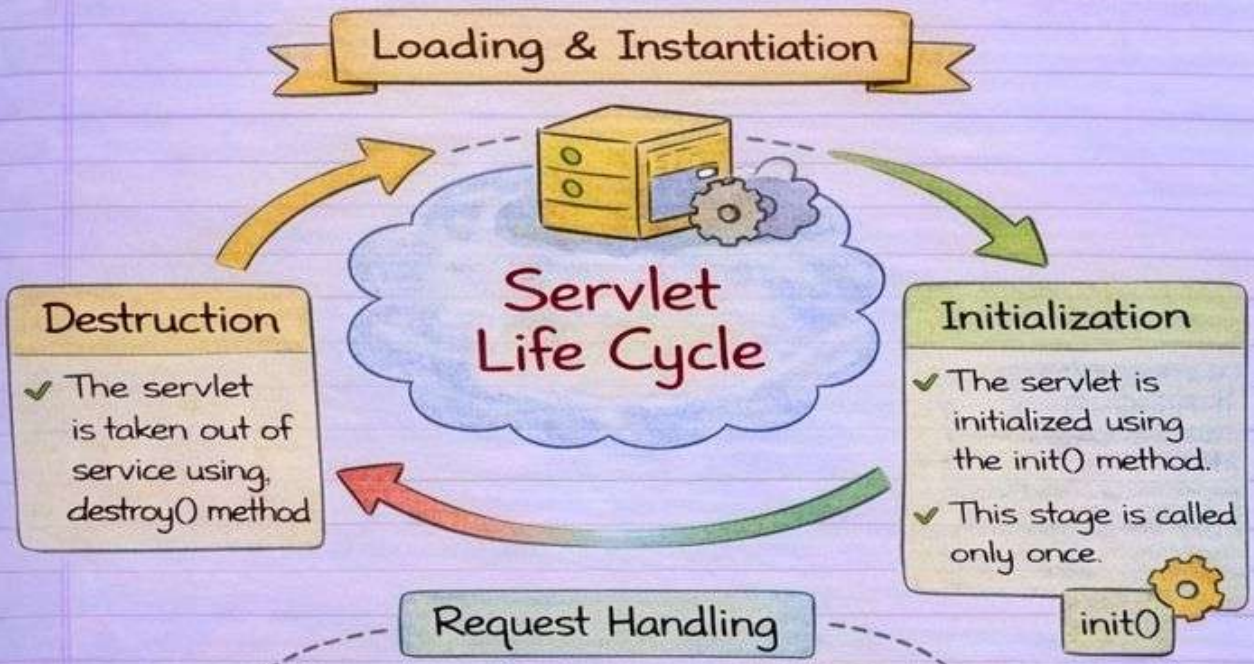
@curious_programmer

Servlet Life Cycle

@curious_programmer

Servlet Life Cycle

A Servlet is a Java class that runs on a server to handle client requests and generate dynamic web content. The life cycle of a servlet is managed by the servlet container, which consists of four main stages:



@curious_programmer

GenericServlet vs HttpServlet

@curious_programmer

GenericServlet vs HttpServlet

What's the Difference?

GenericServlet

GenericServlet

- Part of javax.servlet package.
- Protocol-independent base class.
- Can handle any type of request (not limited to HTTP).

HttpServlet

- Extends GenericServlet & part of javax.servlet.http package.
- Designed specifically for handling HTTP requests.
- Provides HTTP-specific methods like:
  - GET,
  - POST,
  - doGet(),
  - doPost().

Key Differences

- **Superclass:**

GenericServlet → </> HttpServlet

- Defines a protocol-independent base class for creating servlets.
- Extends GenericServlet to specialize in handling HTTP requests.

- **Request Handling:**
  - Handles any type of request (protocol-agnostic). → HTTP →

→ ServictRequest → HttpServlet

- Designed specifically for HTTP via < > methods like doGet(), doPost(), etc.

@curious_programmer

Handling HTTP Requests & Responses

@ curious_programmer

✔ Handling HTTP Requests & Responses

Servlets handle HTTP requests and generate dynamic responses. This involves several steps.

1. Client Sends Request

- ✔ The client (browser) sends an HTTP request to the server.

2. Server Processes Request

- ✔ The server passes the request to the servlet for processing.

3. Servlet Generates Response

- ✔ The servlet processes the request, generates a dynamic HTML response.

4. Servlet Sends Response

- ✔ The servlet sends the HTTP response back to the client (browser).

5. Client Receives Response

- ✔ The client displays the response (web page) in the browser.

@ curious_programmer

GET vs POST

& Form Data Processing

✓ GET vs POST?

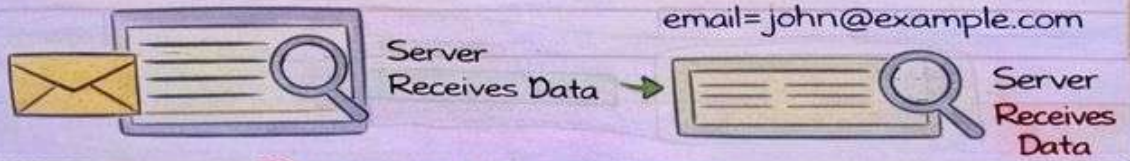
GET and POST are two common HTTP methods used to send form data to a server. Both methods have different characteristics and uses.

Key Differences

1 GET	< POST
✓ Sends data appended to URL as query parameters.	✓ Sends data in the request body (hidden from URL).
✓ Visible in browser's address bar.	✓ Not visible in browser's address bar.
✓ Limited amount of data.	✓ No size limitations for data transfer.
✓ Not secure for sensitive data.	✓ Secure for sending sensitive data (passwords, etc).

Form Data Processing

GET	POST
✓ example.com/form?name=John &email=john@example.com	✓ FormData: name=John email=john@example.com



@ curious_programmer

Servlet Configuration

@ curious_programmer

web.xml

web.xml

```
<servlet>
<servlet-mapping>
```

- ✔ Configuration file located in WEB-INF folder.
- ✔ Uses XML to define servlets, URL mappings, init parameters. etc.
- ✔ Helps maintain configuration outside the code.
- ✔ Supports advanced configurations. like filters, listeners, etc.
- ✔ More verbose and requires maneal mearteronee.

Annotations (@WebServlet)

```
@WebServlet"(/exampleServlet")
public class MyServlet
    extends HtpServlet {...}
1.
```

- ✔ Used directly in Java source code.
- ✔ Uses @WebServlet annotation to define servlets and URL mappings.
- ✔ Simpler and less verbose than web.xm.
- ✔ Supports basic configurations, URL mappings, and init parameters.
- ✔ Eccase to update and moaitaim.

web.xml

- ✔ Configuration file located in WEB-INF folder.
- ✔ Uses XML to define servlets, URL mappings, init parameters.

Annotations

- ✔ Used directly in Java source code.
- ✔ Uses @WebServlet annotation to define servlets and URL mappings.

@ curious_programmer

Servlet Session Tracking

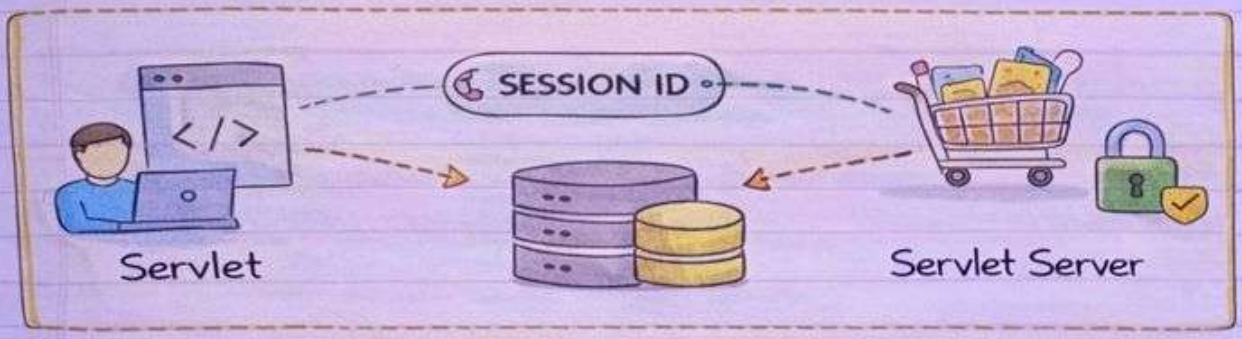
@ curious_programmer

Servlet Session Tracking?

- ✔ Session tracking is a way to maintain state (data) about a client's interactions with a web application across multiple requests.
- ✔ As HTTP is stateless, session tracking helps in remembering user-specific data.
- ✔ Uses session IDs to manage user-specific data (e.g., login status, shopping cart, etc.).
- ✔ Improves user experience by maintaining continuity of interactions.

Why is Session Tracking Required?

- ★ HTTP is stateless; it does not retain information between requests.
- ★ Tracks user activities like logins, shopping cart contents, and preferences.
- ★ Personalizes user experience by storing user-specific data.
- ★ Improves application security by managing sessions and preventing unauthorized access.



@ curious_programmer

Session Tracking Techniques:

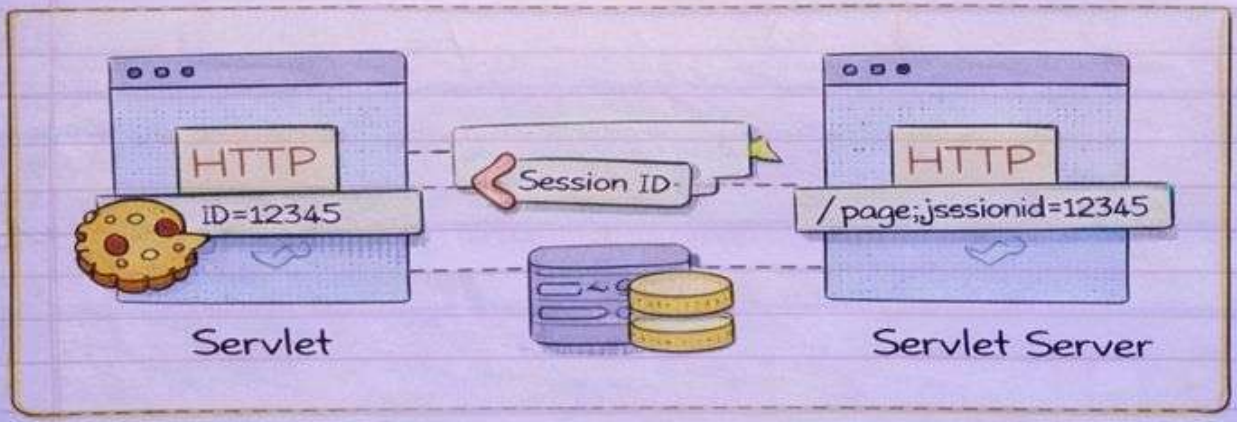
@ curious_programmer

Cookies

- ✔ Small text files stored on the user's browser.
- ✔ Contains a session ID for identifying users.
- ✔ Sent back to the server with each request,
- ✔ Can store user preferences and tracking information.
- ✔ May have expiration time for validity.

URL Rewriting

- ✔ Appends the session ID to the URL as a parameter.
- ✔ Used when cookies are disabled or not prefered.
- ✔ Session ID appears in the URL like /page;jsessionid=12345
- ✔ Can be less secure as session ID is visible.
- ✔ Requires careful management of URLs.



@ curious_programmer

Session Tracking Techniques:

@ curious_programmer

Hidden Form Fields

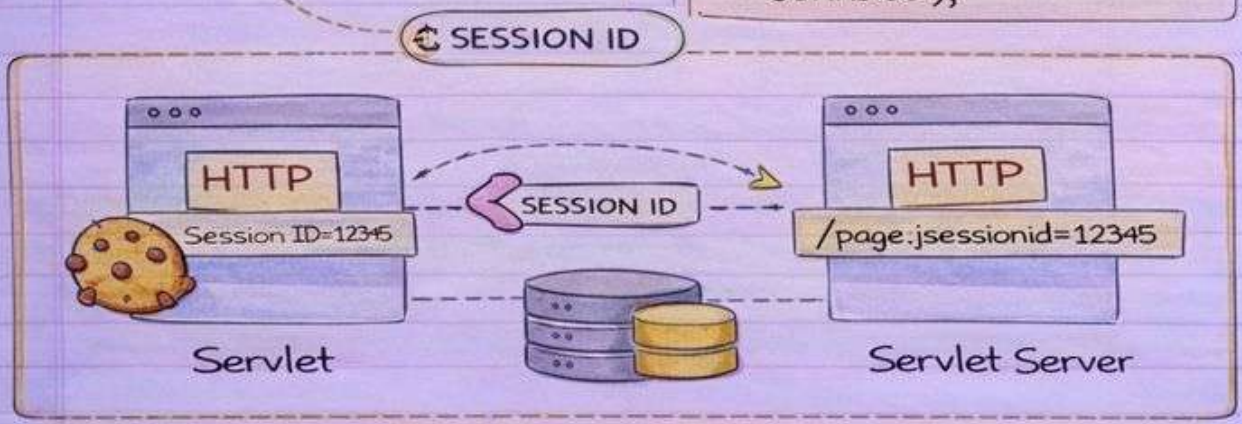
- Hidden input fields are embedded in forms to store session IDs.
- Session ID is sent back to the server with each form submission.
- Used in multi-page forms and basic session management.
- Session ID stored in HTML form as:

```
<input type="hidden"
name="sessionId" value="12345">
```

HttpSession

- Java-based session tracking mechanism.
- Server-side technology that stores session data on the server.
- Automatically creates a HttpSession object to manage session data.
- Accessible with methods like getSession(), setAttribute(), etc.

```
HttpSession session =
request.getSession();
session.setAttribute("username",
"JohnDoe");
```



@ curious_programmer

Servlet Filters vs Servlet Listeners

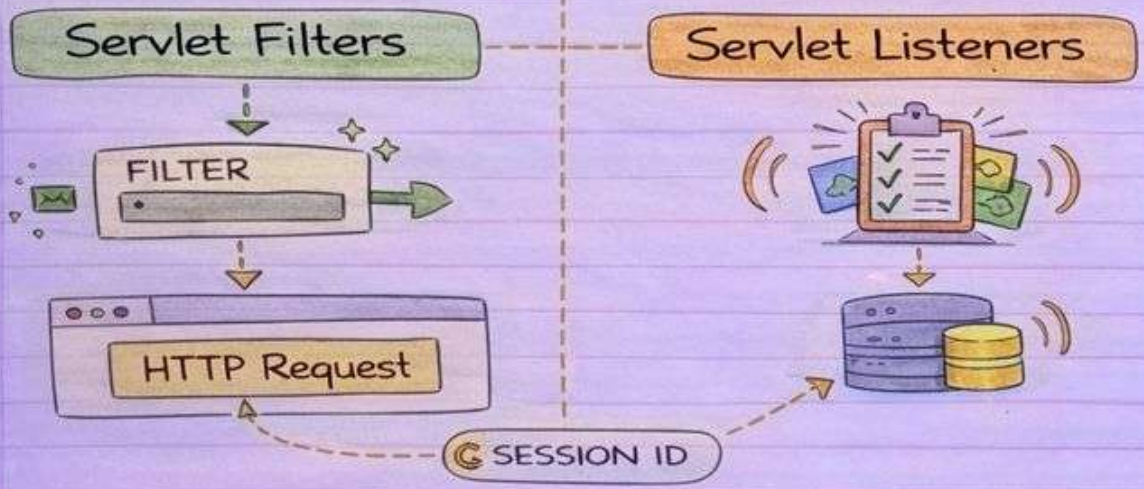
@ curious_programmer

Servlet Filters

- ✔ Used to interecpt and modify requests and responses.
- ✔ Perform tasks like logging, authentication, and data compression.
- ✔ Executed before or after a servlet processes a request.

Servlet Listeners

- ✔ Used to monitor and respond to specific events in the web application.
- ✔ Listen for events like initialization, attribute changes, and session creation/destruction.
- ✔ Triggered automatically when specific actions occur (e.g., session start, context load).

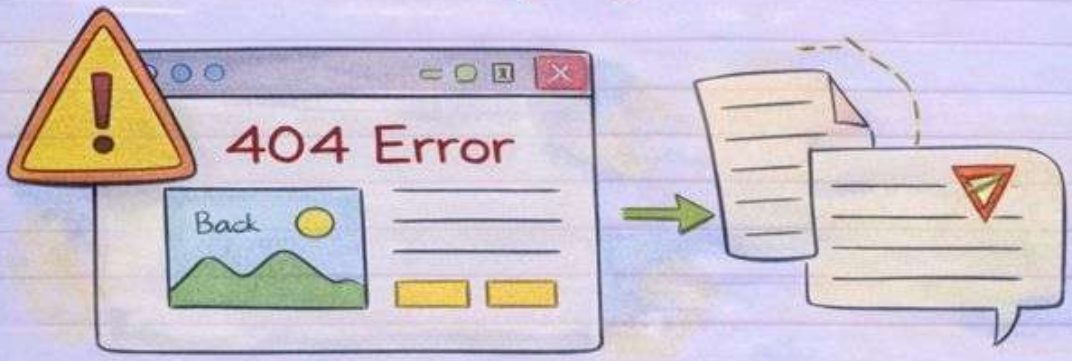


✔ Servlet Fiiters do not respond respon for events in the web application.

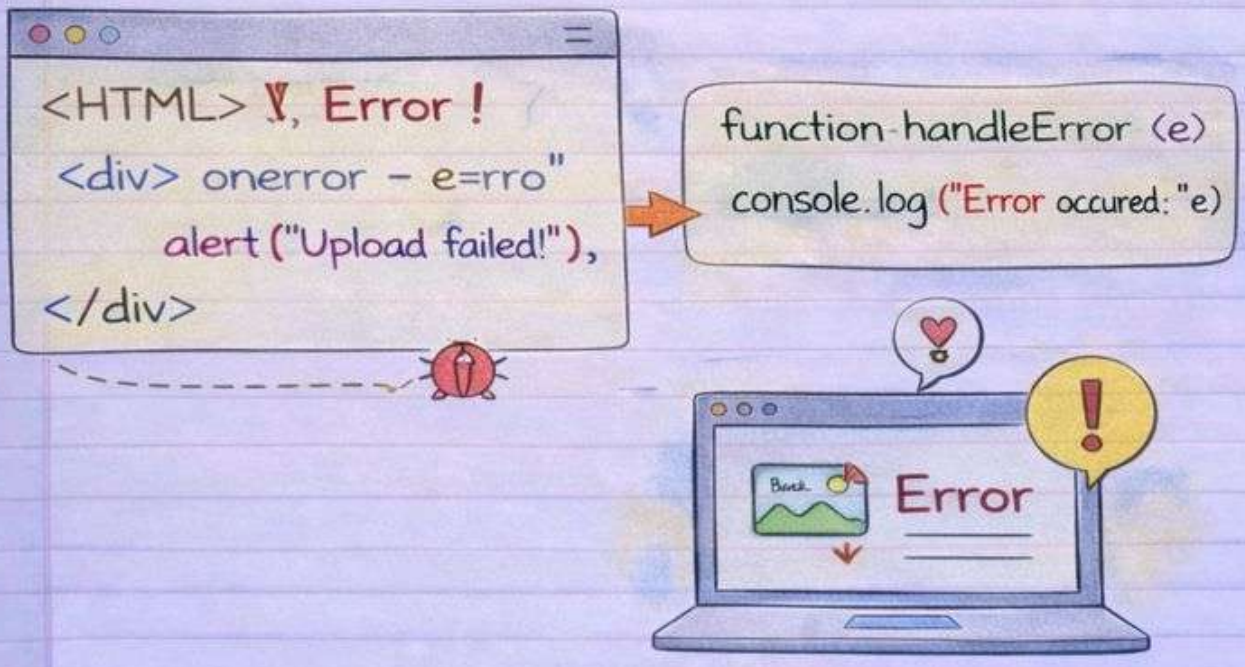
@ curious_programmer

Error Handling

@ curious_programmer



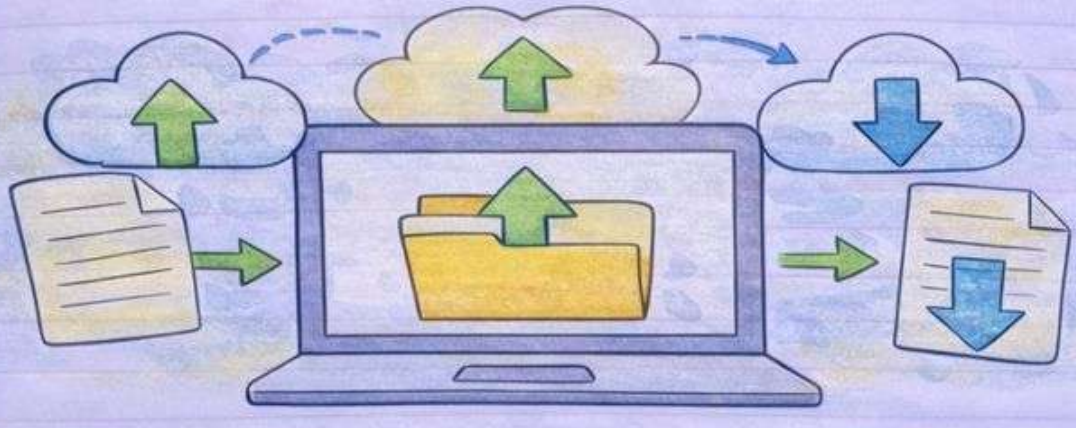
- ✓ Displays clear error messages to guide users,
- ✓ Handles form validation with custom error alerts.
- ✓ Provents broken links and incorrect submissions.



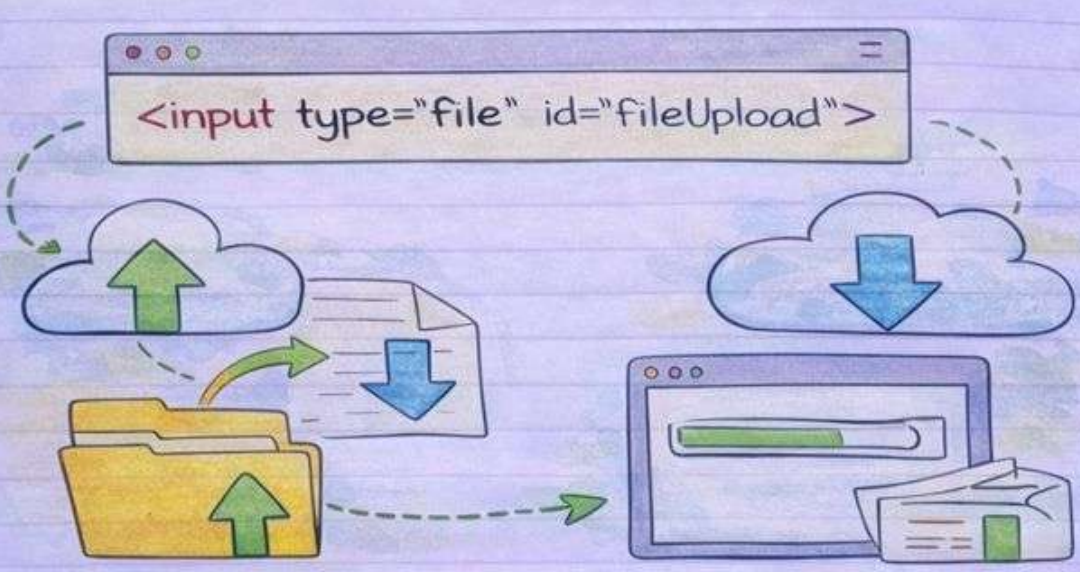
@ curious_programmer

File Upload & Download

@curious_programmer



- Allows users to upload files via forms.
- Enables downloading of content like images, PDFs, and more.
- Uses <input> element with type="file" for uploads.



@curious_programmer

Security in Servlets

@curious_programmer

✔ Secure User Authentication
- Uses login pages and sessions to authenticate users.
- Ensures only authorized users can access resources.

✔ Validate & Sanitize Inputs
- Checks user-provided input for harmful content.
- Sanitizes input to prevent Cross-Site Scripting (XSS) and SQL Injection attacks.

✔ Prevent Data Leaks
- Enforces HTTPS to encrypt data sent over the network.
- Protects sensitive information from being intercepted during transmission.

✔ Protects against attacks
- Defends against Cross-Site Scripting (XSS) attacks.
- Prevents SQL Injection by sanitizing inputs.
- Guards against other vulnerabilities like CSRF (Cross-Site Request Forgery).

@curious_programmer

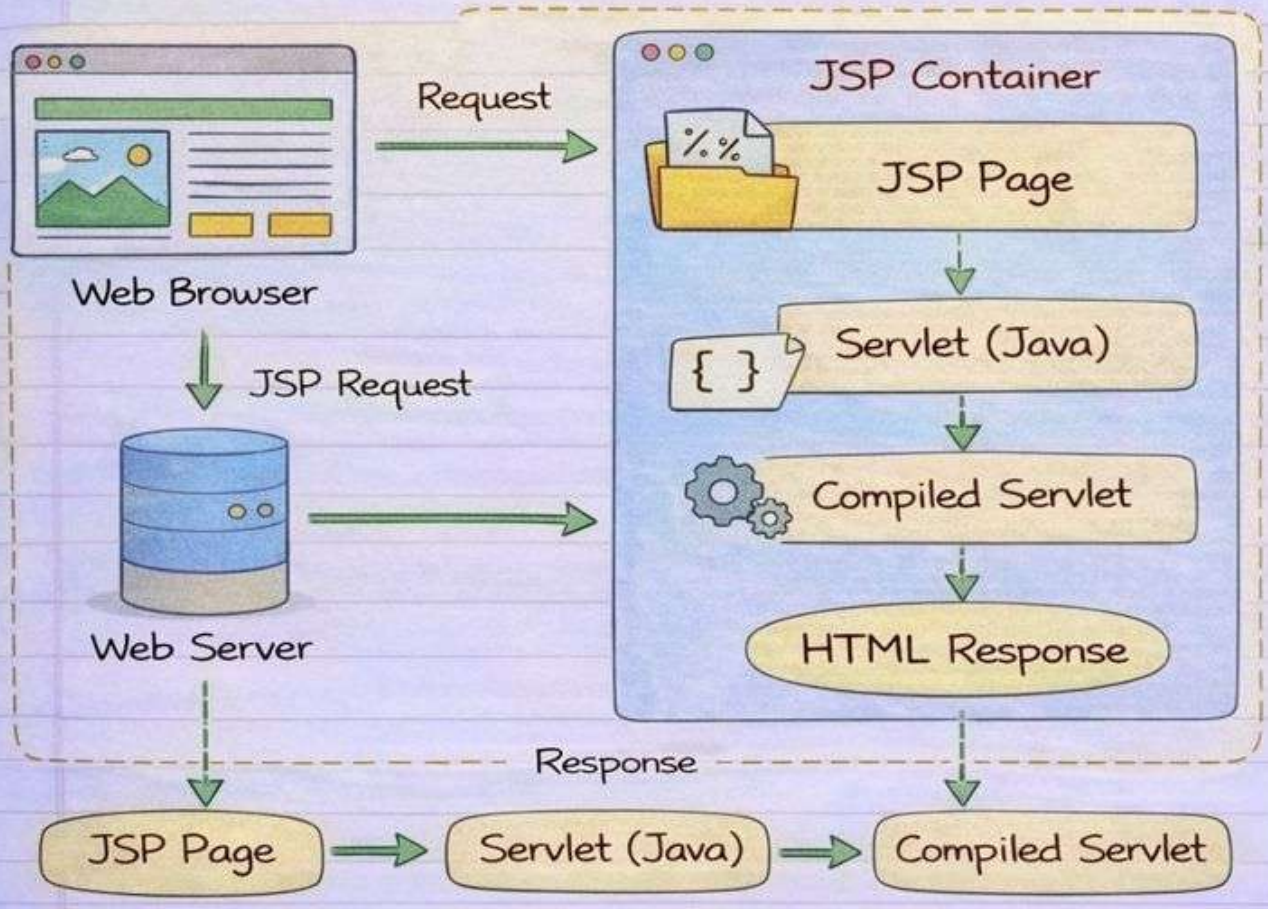
Chapter 3: JavaServer Pages (JSP)

@ curious_programmer

✔ Introduction to JSP

JSP stands for JavaServer Pages. It is a server-side technology used to create dynamic, interactive web pages by embedding Java code into HTML. JSP allows for the creation of custom Java-based content that can be processed on the server and displayed in a web browser.

JSP Architecture



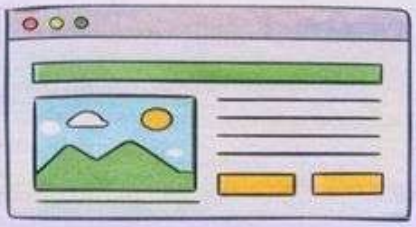
JSP Architecture Explanation

@ curious_programmer

1. Web Browser Sends Request:

- The user makes a request by accessing a JSP page through their web browser.

Request →



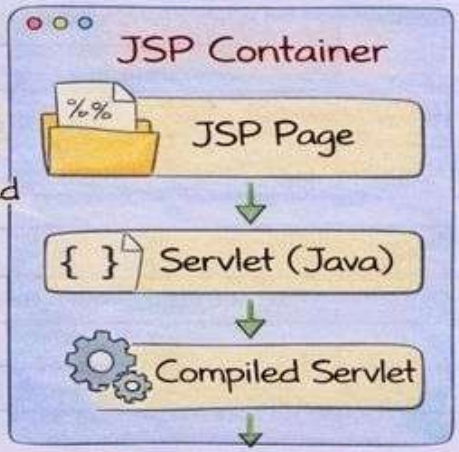
Web Browser

2. Web Server Processes Request:

- The web server receives the JSP request and forwards it to the JSP container for processing.

3. JSP Container Processes JSP Page:

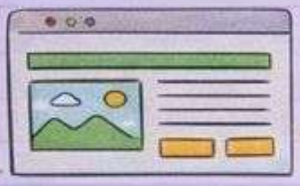
- JSP Page: Contains HTML with embedded Java code (<% %>) to generate dynamic content.
- Servlet (Java): Converts the JSP into a servlet, which is Java code that can be executed on the server.



4. HTML Response Sent Back to Browser:

- The server executes the servlet to generate the dynamic content, which is sent back to the browser as an HTML response.

Response → HTML Response



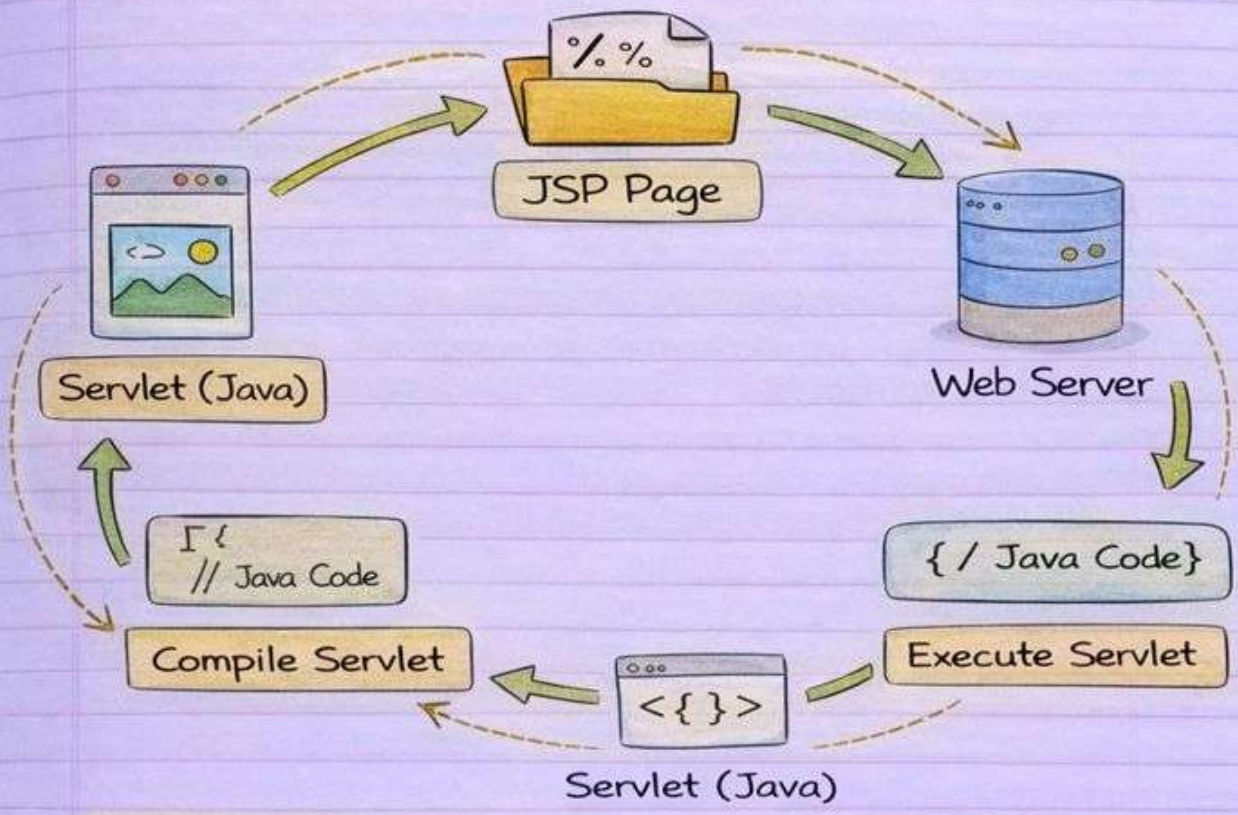
Web Browser

→ The web browser then displays the generated web page to the user.

Overall, JSP allows the creation of dynamic, Java-based content on the server-side that is embedded into HTML, which is then displayed as a web page to the user.

JSP Life Cycle

@curious_programmer



JSP Life Cycle Steps:

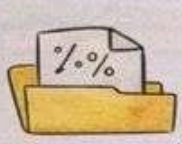
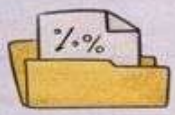
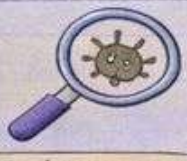
- ✔ **Compilation:** The JSP page is converted into a Java Servlet.
- ✔ **Loading and Initialization:** The servlet is loaded into the server memory and initialized.
- ✔ **Execution:** The servlet is executed to process client requests and generate a response.
- ✔ **Destruction:** When the server shuts down, the servlet is removed from memory.

Overall, the JSP life cycle involves converting a JSP page into a Java servlet, executing the servlet to handle client requests, and eventually removing the servlet from memory when no longer needed.

JSP vs Servlet

@curious_programmer

Difference Between JSP and Servlet:

Criteria	JSP	JSP	Servlet
 Purpose	✔ Used for web page development to embed Java in HTML.	✔ Used for handling HTTP requests and responses via pure Java code.	• Used for handling HTTP requests and responses via pure Java code
 Code Type	✔ Mix of HTML and Java code with JSP tags (<% %>).	✔ Pure Java code for logic; usually no HTML	• Pure Java code for logic; usually no HTML.
 Ease of Use	✔ Easier to write and maintain due to embedded Java in HTML.	✔ More complex, requires detailed knowledge of Java to write and manage	• More complex, requires detailed knowledge of Java to write.
 Debugging	✔ Harder to debug due to indirect translation to servlet code.	• Easier to debug directly in Java code as it is pure Java.	• Faster as it directly handles requests without translation.
 Performance	✔ Can be slightly slower due to translation into servlet first.	✔ Faster as it directly handles requests without translation.	• Faster as it directly handles requests without translation.

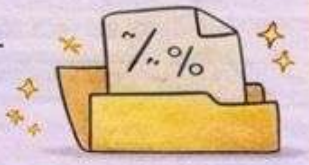
@curious_programmer

JSP Syntax

@ curious_programmer

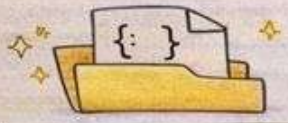
What is JSP Syntax?

JSP syntax allows embedding Java code into HTML pages. By using special JSP tags, you can insert Java code to handle dynamic content, making it easy to mix Java and HTML in a single page.



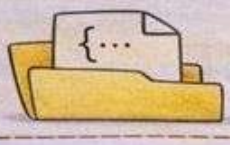
Elements of JSP Syntax

Scriptlet



<% Java code here %>

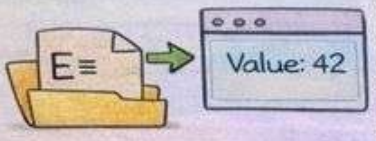
- ✔ Contains Java code.
- ✔ Executes code each time the page is requested



Expression

<%= expression %>

<%= expression %>



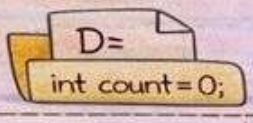
- ✔ Evaluates and outputs Java expressions
- ✔ Displays values in the HTML page.

Declaration

<%! dataType method() { code } %>

<%! dataType method) { code } %>

- ✔ Declares variables or methods.
- ✔ Initialized once, retained between requests.



Elements of JSP Syntax

Scriptlet

<% Java code here %>

- ✔ Contains Java code.
- ✔ Executes code each time the page is requested.

Expression

<%= expression %>

- ✔ Evaluates and outputs Java expressions.
- ✔ Displays values in the HTML page.

Declaration

<%! dataType method() { code } %>

- ✔ Initialized once, retained between requests.

@ curious_programmer

JSP Directives (page, include, taglib)

@curious_programmer

Page Directive

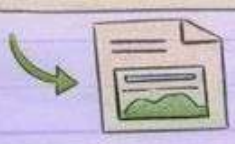
- ✓ Defines page-specific attributes and settings for the JSP file.
  - Can set content type, language, error page, etc.
  - Example:

```
<<% page contentType="text/html" language="java"
     errorPage="error.jsp" %>
```

Include Directive

- ✓ Includes a file during the translation phase.
  - Used to include static content like headers, footers, etc.
  - Example:

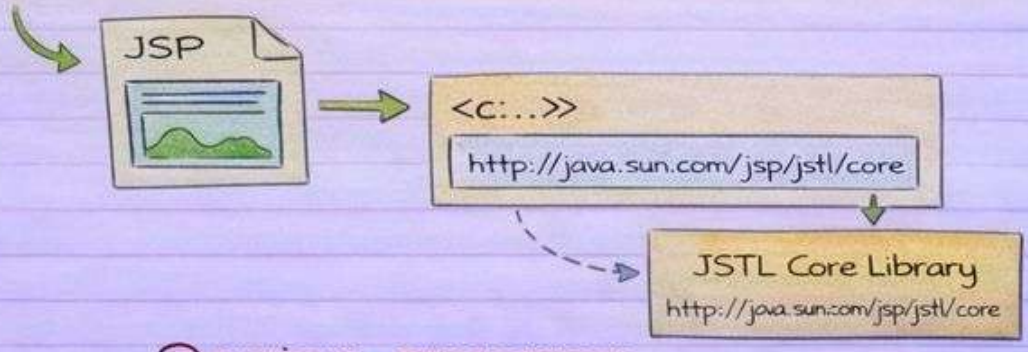
```
<<% include file="header.jsp" %>
```



Taglib Directive

- ✓ Declares a tag library for use in the JSP file.
  - Allows using custom tags by referencing a tag library.

```
<<% taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```



@curious_programmer

JSP Directives (page, include, taglib)

@curious_programmer

Page Directive

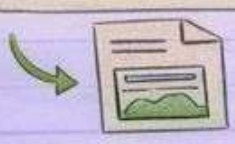
- ✓ Defines page-specific attributes and settings for the JSP file.
  - Can set content type, language, error page, etc.
  - Example:

```
<<% page contentType="text/html" language="java"
     errorPage="error.jsp" %>
```

Include Directive

- ✓ Includes a file during the translation phase.
  - Used to include static content like headers, footers, etc.
  - Example:

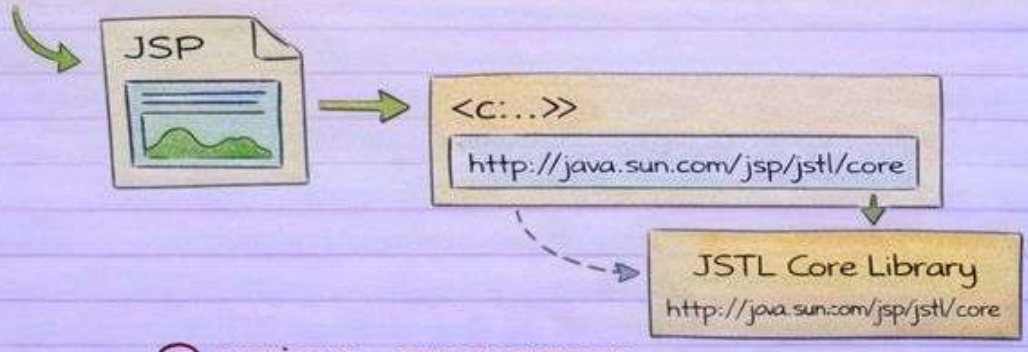
```
<<% include file="header.jsp" %>
```



Taglib Directive

- ✓ Declares a tag library for use in the JSP file.
  - Allows using custom tags by referencing a tag library.

```
<<% taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```



@curious_programmer

JSP Actions (jsp:useBean, jsp:setProperty, etc.)

@curious_programmer

jsp:useBean

- ✓ Creates or locates a JavaBean in the page scope.
- Specifies a JavaBean class and an identifier (id).
- Can create a new bean if it doesn't exist yet.

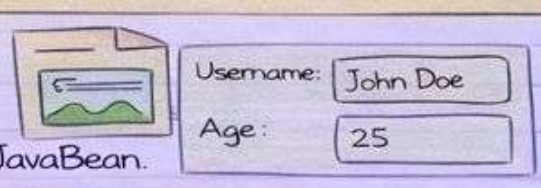
```
<<jsp:useBean id="myBean"
    class="com.example.MyBean" scope="page"/>
```

jsp:setProperty

- ✓ Sets a property of a JavaBean.
- Can set a property value with a static value or request parameter.
- Example:

```
<<jsp:setProperty name="myBean" property="username" value="John Doe"/>
<<jsp:setProperty name="myBean" property="age" param="userAge"/>
```

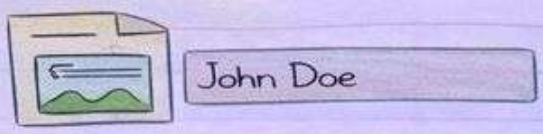
jsp:getProperty



- ✓ Retrieves a property value from a JavaBean.
- Displays the specified property of a JavaBean.

```
<<jsp:getProperty name="myBean" property="username" />
```

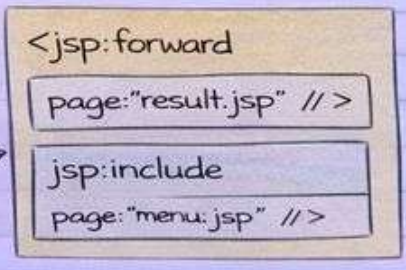
Other JSP Actions



- ✓ Provide functionalities to the page
- Examples: `<jsp:forward page="result.jsp" />`

ze4.-ec-zcs
tjsp:amam

- <jsp:forward> Forwards to another resource.
- <jsp:include> Includes a resource at runtime
- <jsp:param> Adds a parameter to a request.



@curious_programmer

What are Implicit Objects?

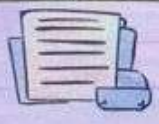
@ curious_programmer

Implicit Objects

- Predefined objects in JSP that provide access to various aspects of the web environment. They are created automatically and can be used directly in JSP without explicit declaration.

Common Implicit Objects

request

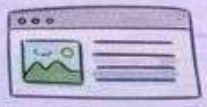


- Provides information about the request: Used to get parameters, headers.
- Example:

```
"Hello, <%= request.getParameter("name") %>
```

→ welcome.jsp

response



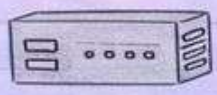
- Tracks user sessions.
- Used to store session data.

```
<% response.sendRedirect("welcome.jsp");
```

session

- Tracks user session.
- Used to store session data.

application



- Provides information about the web application.
- Used to share data between users.

```
<% application
    setAttribute("companyName"
                 "ABC Corp"
```

application

```
<% application.setAttribute("companyName", "ABC Corp") %>
```

@ curious_programmer

More Implicit Objects

@curious_programmer

In addition to the common implicit objects, JSP provides more implicit objects to facilitate advanced functionality.

Additional Implicit Objects

config

- Provides the Servlet configuration information.
- Used to get initialization parameters.

```
<<%-config.getInitParameter("setting") %>
```

→ DB URL: jdbc:mysql://localhost/db

pageContext

- Provides access to various page attrbutes.
- Used to access to request, session, application scopes.

```
<% pageContext.forward("anotherPage.jsp"); %>
```

→ currentPage.jsp

page

- Represents the current JSP page.
- Rarely used, as it points to current page itself.

```
<% String info = page.toString(); %>
```

→ myPage.jsp

Result: "myPage.jsp"

exception

- For handling errors in JSP with error pages.
- Used as an instance of JspWriter.

```
<% exception.printStackTrace(); %>
```

→ Output: Hello, JSP!

out

- Used for sending output to the client.
- Facilitates printing output to the response.

```
<% out.println("Hello, JSP!"); %>
```

→ Output: Hello, JSP!

@curious_programmer

JSP Standard Tag Library (JSTL)

@curious_programmer

JSTL

A powerful set of tags provided by Java Standard Tag Library to simplify JSP development by encapsulating common tasks.

- Provides tags for control flow, data manipulation, internationalization, etc.
- Helps write cleaner and more maintainable code.

```
<<% taglib uri="http://java.sun.com/jsp/jstl/core" prefix='c" %>
```

JSTL Tag Libraries

Core Tags

- Provides tags for control flow,
- Used for conditional logic, iteration.

```
<c:if test="$(param.option =="yes")>
    Yes!/< if
```

Formatting Tags

- Provides tags for text, numbers, and dates.
- Used for internationalization (i18n).

```
$12,345.00
```

SQL Tags

- Provides tags for working with databases.

```
<sql:query var="result"
    dataSource="jdbc/myDB>
SELECT + FROM users
  </sql:query>
```

XML Tags

- Provides tags for working with XML documents.
- Used for parsing, transforming XML data.

```
<x:parse var= data">
    <x:set select:/root/item'> ...
```

XML Tags

- Provides tags for working with XML documents.
- Used for parsing, transforming

```
$(fn:toUpperCase('hello"))
```

Functions Tags

- Provides tags for performing common functions like string manipulation.

```
$(fn:toUpperCase('hello"))
HELLO
```

@curious_programmer

Expression Language (EL)

@ curious_programmer

What is EL?

A simple language for accessing data in JavaServer Pages.

- Allows easy access to data stored in JavaBeans, request, session, application, etc.
- Simplifies JSP page development by embedding expressions directly into the JSP pages.

Why Use EL?

Simplifies the code

- Simplifies the code in JSP pages by reducing the need for Java code inside HTML.

- Improves readability and maintainability of JSP pages.
- Makes accessing data like request, session, and beans easier.

EL Syntax

${object.property}

${user.name}

Using EL Objects

- request
- session
- application
- JavaBeans

- Access request parameters: ${param.paramName}
- Access session attributes: ${session.attrName}
- Access JavaBeans properties: ${bean.property}

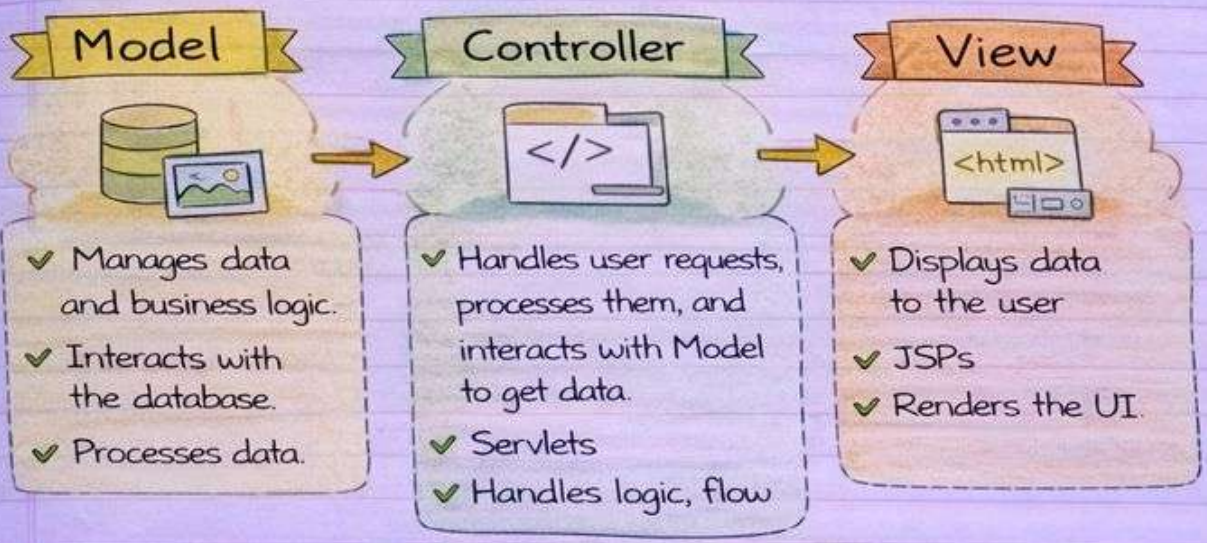
${object.property} → ${user.name}

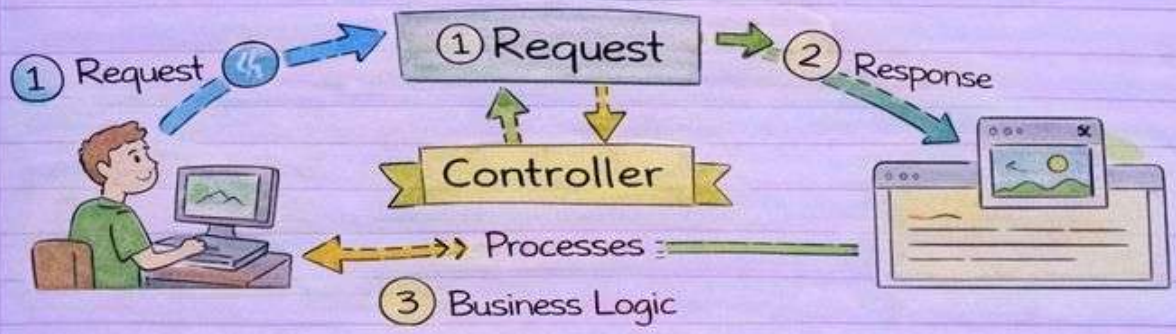
@ curious_programmer

MVC Architecture with JSP & Servlet

@curious_programmer

MVC (Model-View-Controller) is a design pattern used to separate an application into three interconnected components.





Steps in MVC Flow

1. User makes a request via a JSP page to the Servlet (Controller).
2. Servlet (Controller) processes request using Model to retrieve or store data.
3. Servlet forwards the response to a JSP page (View) to display data to the user.

① Request	② Business Logic	③ Response
① User makes a request via JSP page Servlet (Controller).	② Servlet (Controller) processes request using the Model to retrieve or store data.	③ Servlet forward response to a JSP page (View) to display data to the user.

@curious_programmer

Error Handling in JSP

@curious_programmer

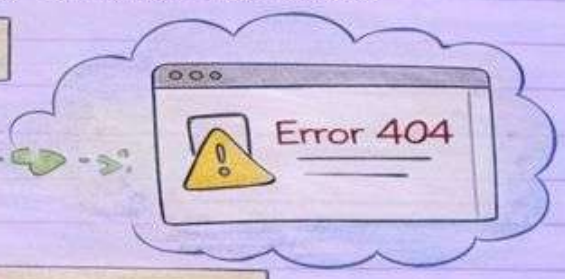
In JSP, errors can occur during the execution of a page. Proper error handling ensures that users receive clear and friendly error messages instead of displaying technical details.

Declaration for Error Page

- Declare an error page in your JSP that will handle the errors.

```
<%@ page isErrorPage="true" %>
```

Error Page Creation



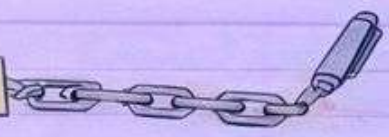
- Here is an example error page.

```
<%@ page isErrorPage="true" %>
<html>
<body>
  <h>Oops! An error occurred. <h1>
  <p>Sorry, an error has occurred. Please try again later.
  <br/>
  <p:Error details: <%= exception.getMessage() %></p>
 </body>
</html>
```

Page Directive

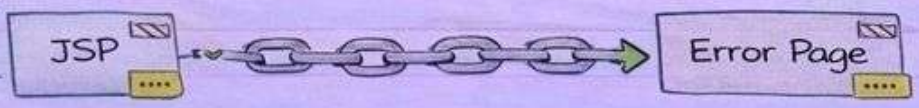
- In the main JSP file, use the "errorPage" attribute to direct to the error page.

```
<%@ page errorPage="errorPage.jsp" %>
```



Steps in Error Handling

- Create an error page with isErrorPage="true" to handle errors.
- Use the "errorPage" directive in the main JSP page to specify the error page.
- When an error occurs, the user is redirected to the specified error page.



@curious_programmer

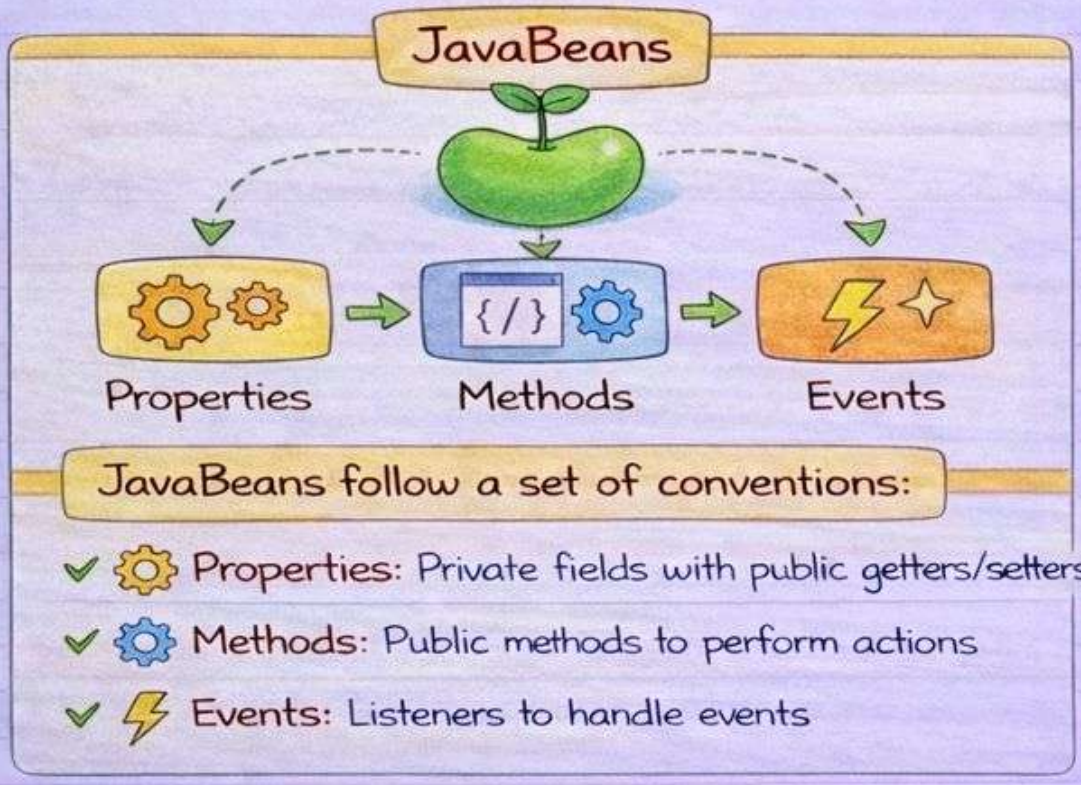
Chapter 4: Introduction to JavaBeans

@ curious_programmer

✓ What is JavaBeans?

JavaBeans are reusable software components for Java applications. They follow a set of conventions and can be easily integrated and manipulated in visual development tools.

JavaBeans encapsulate data into reusable and customizable Java components known as beans.



JavaBean Conventions

@curious_programmer

1 Properties

- Use private fields with public getters and setters to encapsulate data.

```
public class UserBean {
    private String name; // Private field
    public String getName() {
        return name;
    }.
    public void setName(String name){
        this.name = name;
    }
}
```

- Keeps data hidden from other classes
- Getter methods return the property value
- Setter methods allow setting the property value

2 Methods

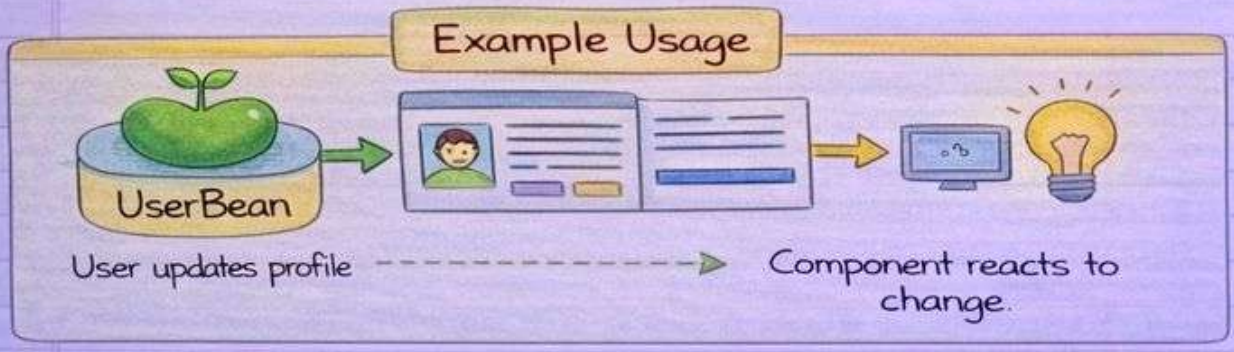
- Provide public methods to perform actions.

```
public class UserBean {
    public void updateProfile() {
        System.out.println("Updating
            user profile...");
    }
}
```

- Public methods provide functionality to other classes
- Perform actions like calculations, database access, etc.
- Usually, method names start with a verb.

3 Events

- Use event listeners to handle events and notify other components.



Properties

(Read-only, Write-only, Read-Write)

@ curious_programmer

JavaBean properties control access to data encapsulated in beans. Properties can be:

Read-only | Write-only | Read-Write

Read-only Properties

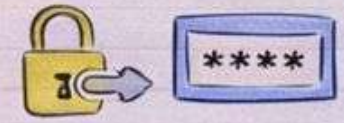
Getter method only for read access:

```
public class Product {
 private double price; //Private field
 public double getPrice() {
    return price;
 }
```

Write-only Properties

Setter method only for write access:

```
public class User {
 private String password; //Private field
 public void setPassword(String password) {
    this.password = password;
 }
```



Read-Write Properties

Both getter and setter methods for read and write access.

```
public class Person {
 private int age; //Private field
 public int getAge() {
    return age;
 }
  public void setAge(int age) {
    this.age = age;
 }
```

Getter & Setter Methods

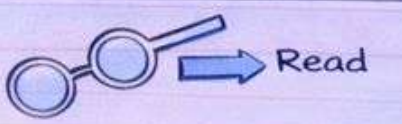
@ curious_programmer

What are Getter and Setter Methods?

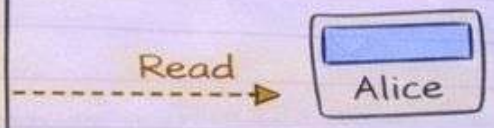
✔ Getter and setter methods are used to access and modify the value of private fields in JavaBeans.

Getter Method

Provides read access to a private field.

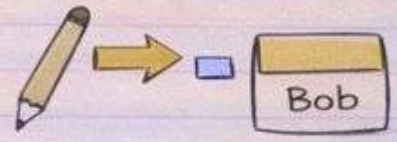



```
public class UserBean {
  private String name; // Private field
  public String getName() {
     return name;
  }
```

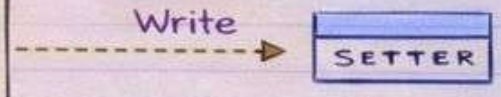


Setter Method

Provides write access to a private field.




```
public class UserBean {
  private String name; // Private field
  public void setName(String name)
     this. name = name;
  }
```



Getter and setter methods ensure data encapsulation, allowing control ous access and modification of private fields.

✔ Getter methods start with 'get' and return the field value.

✔ Setter methods start with 'set' and update the field value.

Using Beans in JSP

Advantages of JavaBeans

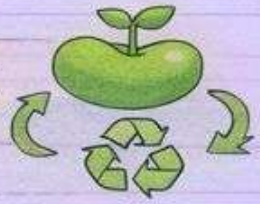
Why use JavaBeans in JSP?

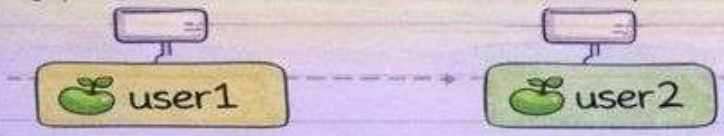
- JavaBeans in JSP (JavaServer Pages) encapsulate business logic and properties into reusable components called beans, providing several advantages:

1. Reusability

- Beans are reusable components that can be instantiated multiple times.

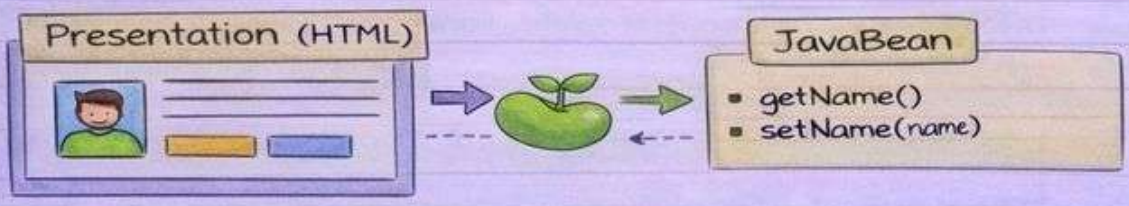
```
<jsp:useBean id="user1" class="com.example.UserBean" />
<jsp:useBean id="user2" class="com.example.UserBean" />
```





2. Separation of Logic

- JavaBeans separate business logic from presentation, making JSPs cleaner and easier to maintain.

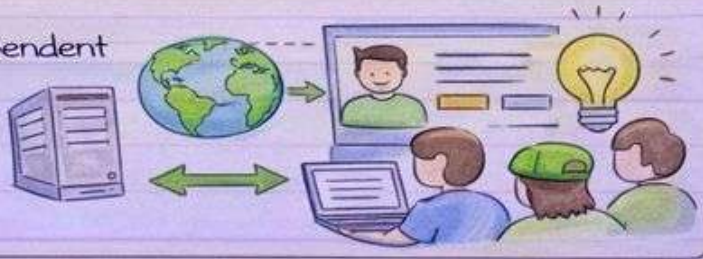


3. Maintainability

- Easier to update business logic by modifying beans without changing the presentation layers.

4 Portability

- Beans are platform-independent and can be moved across different environments.



Chapter 5: MVC Architecture

Understanding MVC Design Pattern

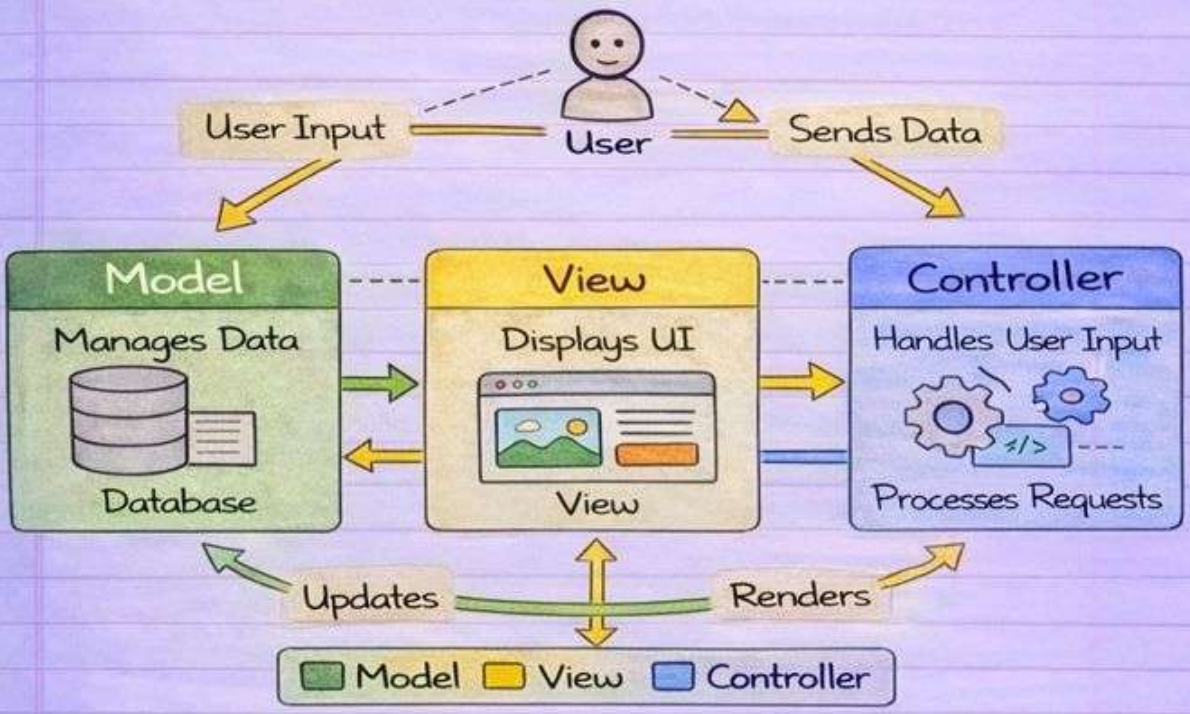
@curious_programmer

✔ What is MVC?

MVC stands for Model-View-Controller. It is a software architectural pattern that separates an application into three interconnected components: **Model**, **View**, and **Controller**.

The MVC design pattern helps organize code by separating the application's logic into three distinct parts. This separation of concerns makes the application easier to manage, develop, and test.

✔ Role of Model, View, Controller



✔ **Model**: Manages the data and business logic of the application.
- Interacts with the database to retrieve, save, and manipulate data.

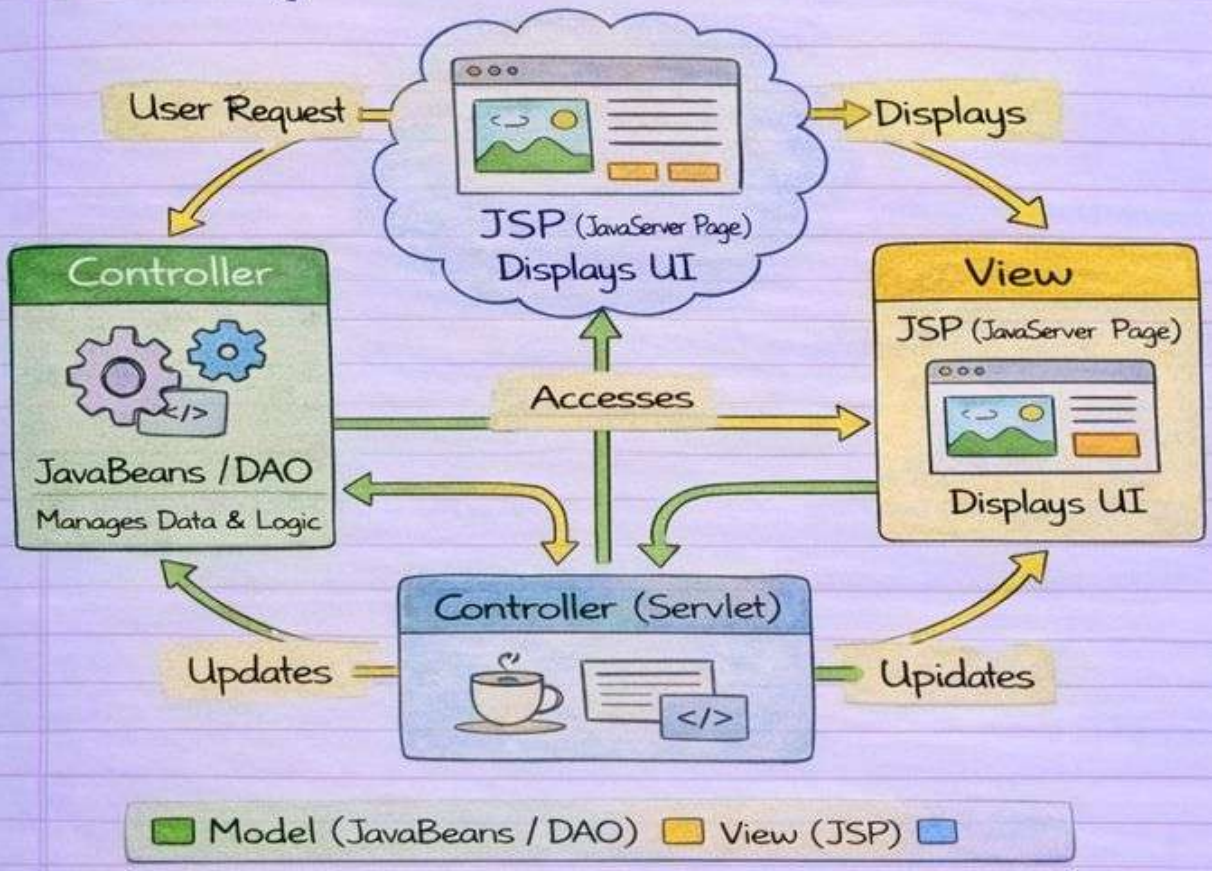
✔ **View**. Responsible for displaying the user interface (UI) and presenting data to the user. Receives data from the Model and renders it for the user.

✔ **Controller**: Acts as an intermediary between the Model and View
- Handles user input, processes requests, and updates the Model.

MVC using Servlets, JSP and JavaBeans/DAO

@curious_programmer

✓ MVC using Servlets as Controller, JSP as View



✓ Model (JavaBeans / DAO)

- JavaBeans hold the data and business logic.
- DAOs (Data Access Objects) handle database operations.

✓ View (JSP)

- JSP files display the user interface by embedding Java code within HTML

✓ Controller (Servlet)

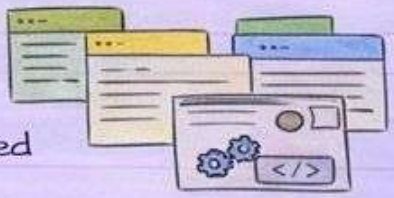
- Servlets handle client requests by acting as the Controller.
- They interact with JavaBeans/DAO to access data and forward results to JSP for display.

Benefits of MVC

@curious_programmer

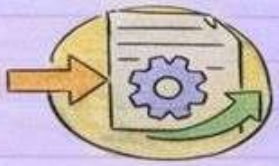
✔ Benefits of MVC

✔ Separation of Concerns



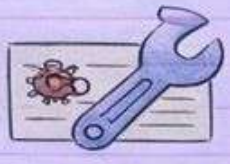
Divides the application into Model, View, and Controller, allowing each to be developed and maintained independentty.

✔ Reusability



Components like Models and Views can be reused across different parts of the application.

✔ Maintainability

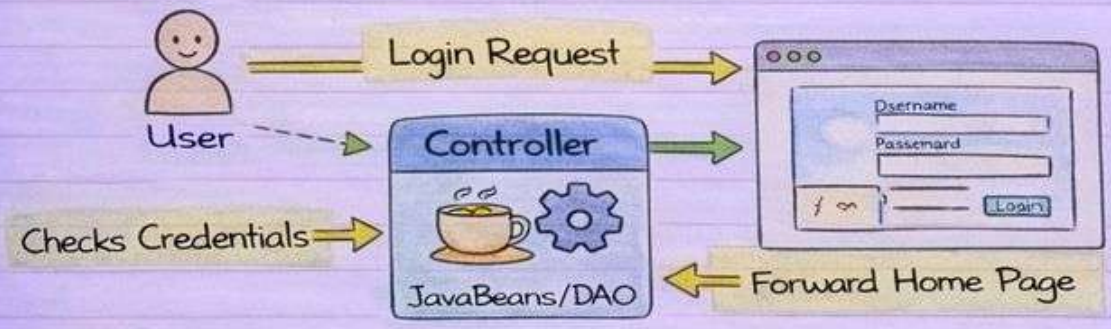


- Easier to update, debug, and test each part without affecting the others.

✔ Scalability

- More flexible as the application grows, making it easier to add new features.

✔ Real-world Web Application Flow



- ✔ User makes a login request via a web browser.
- ✔ Controller (Serviet) checks credentials by calling JavaBeans/DAO (Model).
- ✔ If credentials are correct, user is forwarded to the home page (View).

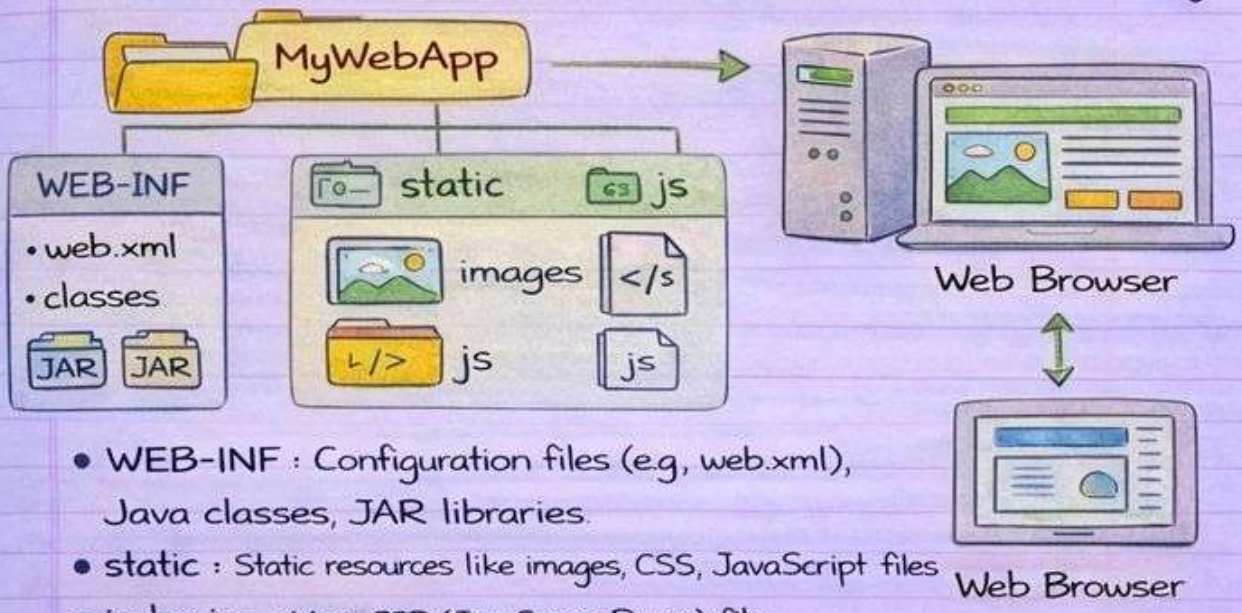
@curious_programmer

Chapter 6: Web Application Development

@ curious_programmer

Web Application Structure

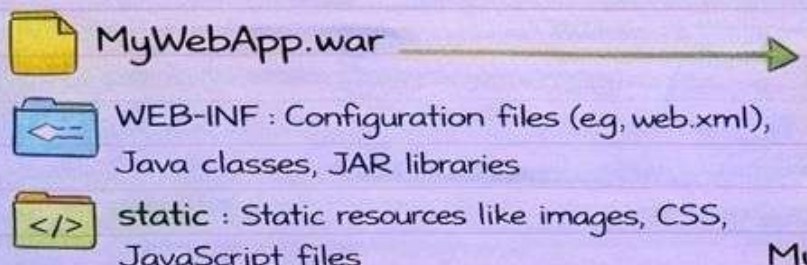
A web application is an application that runs on a server and is accessed through a web browser. Typically, a web application has a structured directory layout, containing various components including:



- **WEB-INF** : Configuration files (e.g, web.xml), Java classes, JAR libraries.
- **static** : Static resources like images, CSS, JavaScript files
- **index.jsp** : Main JSP (JavaServer Pages) file, the entry point of the application.

WAR Files

WAR stands for Web Application Archive. It is a file used to distribute a web application, containing all the contents of the application packaged into a .war file (Web application ARchive).



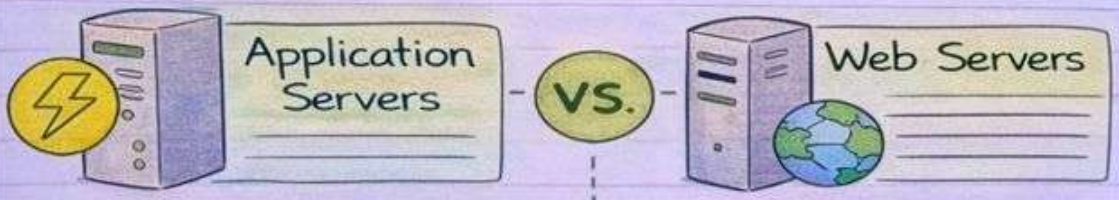
Deployment Descriptor

@ curious_programmer

Deployment Descriptor

- A deployment descriptor is an XML file used to describe how a web application should be deployed and configured in a server.
- ✔ Defines configuration settings for the application.
- ✔ Specifies servlets, URL patterns, initialization parameters, and security settings.

Application Servers vs Web Servers



- Runs web applications and enterprise-level applications.
- Supports Servlets, JSP (JavaServer Pages), EJBs, JNDI, and transactions.
- Provides services like database connectivity. session management, and messaging.

- Runs web applications that generate dynamic content
- Handles HTTP requests and responses.
- Supports Servlets and JSP, but not EJBs or advanced features

Overlapping Feature:

- Both can run Servlets and JSP to generate dynamic web content.

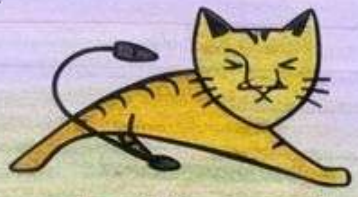
@ curious_programmer

Introduction to Apache Tomcat

@curious_programmer

Apache Tomcat

- Apache Tomcat is an open source web server and servlet container developed by the Apache Software Foundation.

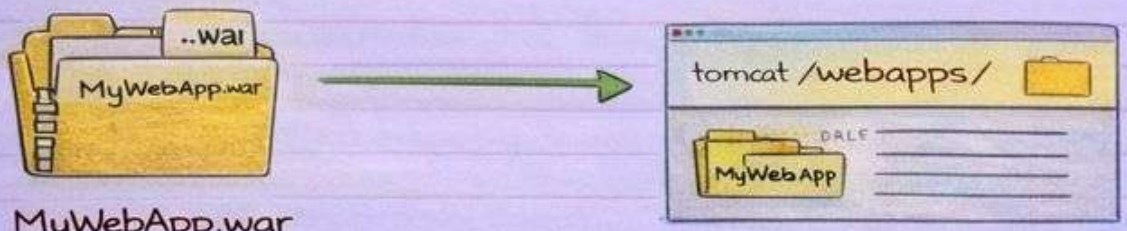


- ✔ It is used to deploy and run Java-based web applications in a lightweight, fast, and efficient environment.
- ✔ Supports Java Servlets, JSP (JavaServer Pages), and WebSocket.
- ✔ Cross-platform, runs on Windows, macOS, Linux, etc.

Deploying Web Applications

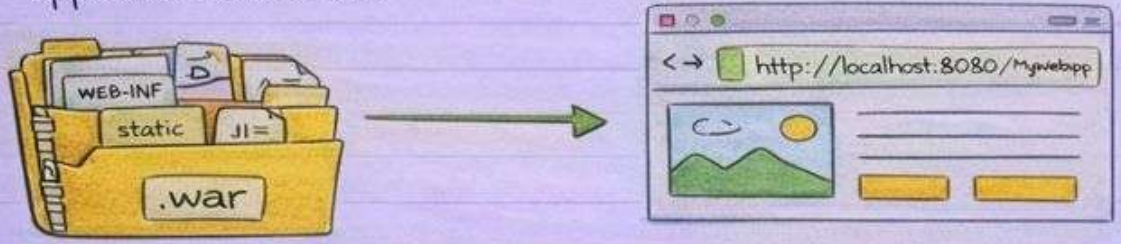
To deploy a web application in Apache Tomcat, follow these steps:

- ✔ Package the web application into a **WAR file** (Web Application Archive).



MyWebApp.war

- ✔ Copy the WAR file to the "webapps" directory in the Tomcat installation directory.
- ✔ Tomcat automatically deploys the WAR file and makes the web application accessible.



Done! The web application is now accessible via http://localhost:8080/MyWebApp

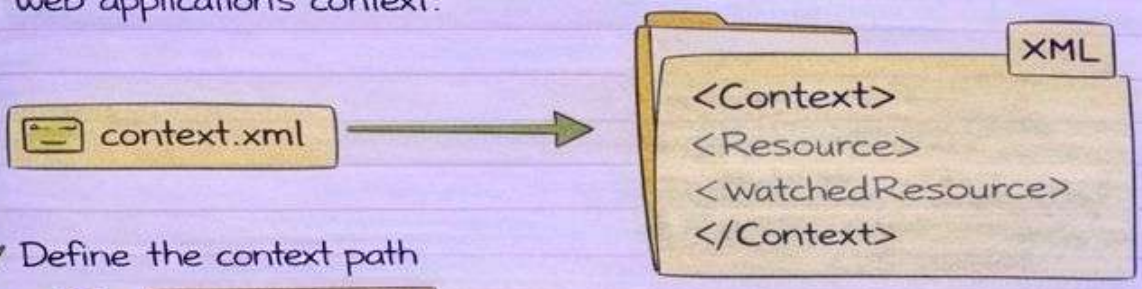
@curious_programmer

Context & Configuration Files

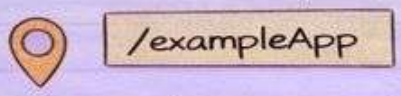
@curious_programmer

Context Files

- Context files are XML files that define settings specific to a web application's context.



- ✔ Define the context path



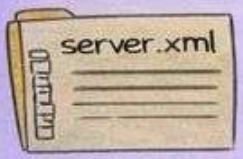
- ✔ Configure data sources (e.g, database connection pools)
- ✔ Set up security constraints and session settings.

Configuration Files

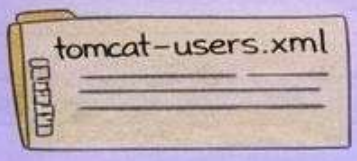
Configuration files are XML files that define server-wide settings, including how web applications are deployed and managed.

- ✔ Most common configuration files are the most common:

Deployment Descriptor
— Defines servlets, servlet-mapping, welcome files, error pages, etc.



Server.xml Main server configuration file
— Configures server level components, ports, thread pools, etc.



Users and roles
— Defines user accounts, roles, and admin permissions.

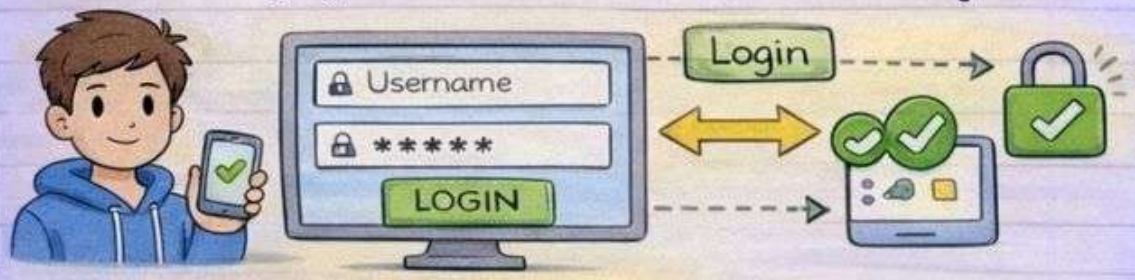
@curious_programmer

Chapter 7: Session Management & Security

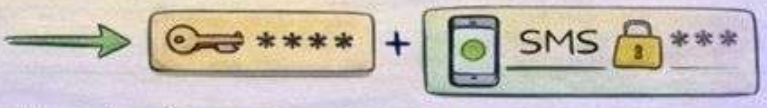
Authentication & Authorization

Authentication

Authentication is the process of verifying who a user is.
It ensures only legitimate users can access an account or system.

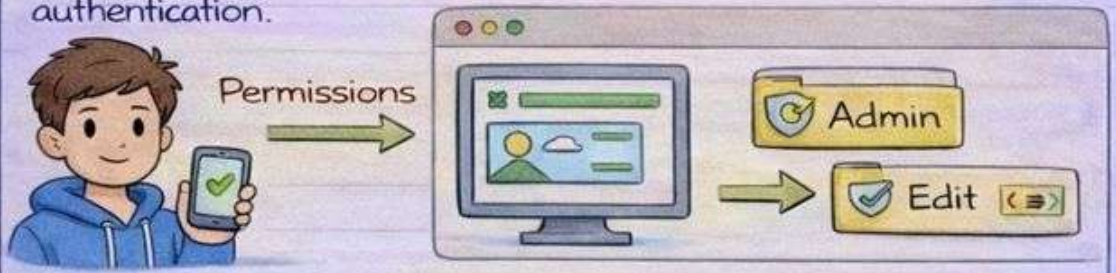


Examples:
→ Passwords
→ Two-Factor Authentication

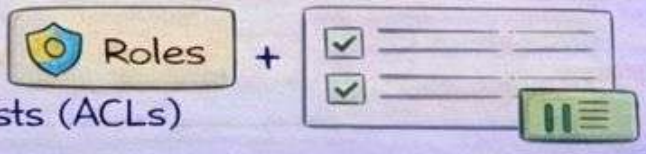


Authorization

Authorization is the process of verifying what a user is allowed to do. It controls access levels and permissions after authentication.



Examples:
→ Roles
→ Access Control Lists (ACLs)



Form-based Authentication & Programmatic Security

@ curious_programmer

✔ Form-based Authentication

Form-based Authentication is the process of verifying a user's identity using an HTML form.

Example:

→ Username

→ Password

✔ Programmatic Security

Programmatic Security involves handling authentication and authorization through code in a secure, programmable way.

- Developers use server-side code to check a user's credentials and enforce access controls based on roles or permissions.

✔ Example:

```
if (authenticate(username, password)
 { if (user.hasRole("admin')) {
    // grant admin access
 } else
    // grant regular access
 }
```

@ curious_programmer

Session Timeout & Preventing Session Hijacking

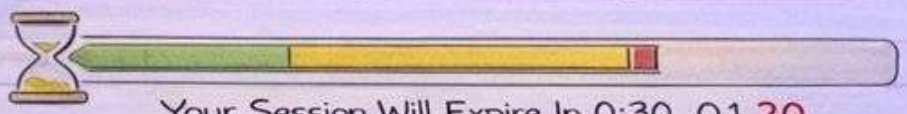
@ curious_programmer

✔ Session Timeout

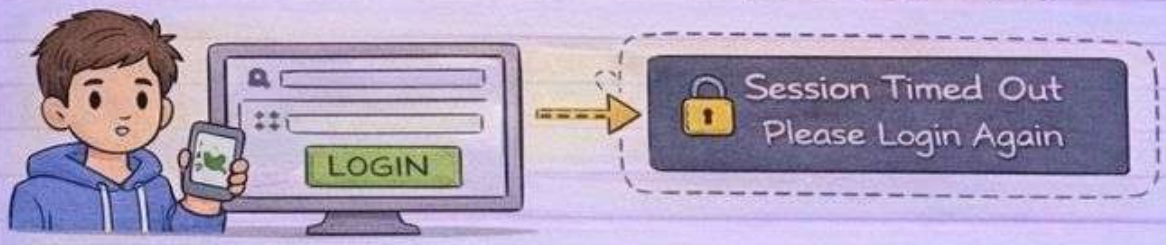
Session Timeout is a security feature that automatically logs out a user after a period of inactivity.

- Prevents unauthorized access if a user leaves their session unattended.
- Reduces the risk of sessions being exploited through idle time.

✔ Example:



Your Session Will Expire In 0:30 01 20



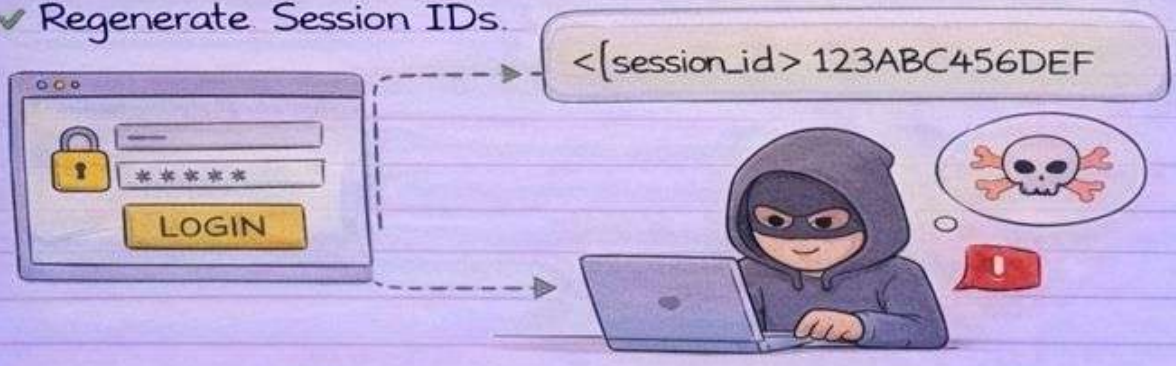
Preventing Session Hijacking

Session Hijacking is an attack where an attacker takes over a user's session.

✔ Use Secure Cookies

→ Implement HTTPS

✔ Regenerate Session IDs.



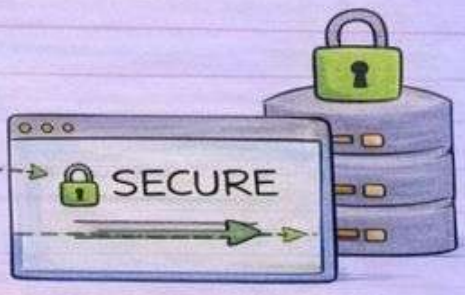
@ curious_programmer

HTTPS Basics & CSRF & XSS Overview

@ curious_programmer

HTTPS Basics

- HTTPS stands for Hypertext Transfer Protocol Secure, It encrypts data transmitted between a user's browser.
- HTTPS is important:
  - Encrypts data to keep it secure from eavesdroppers.
  - Secure communication between a user's browser and server.



- Example: https://www.example.com

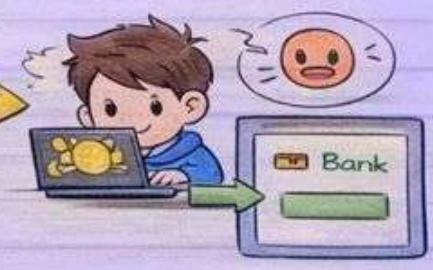
CSRF & XSS Overview

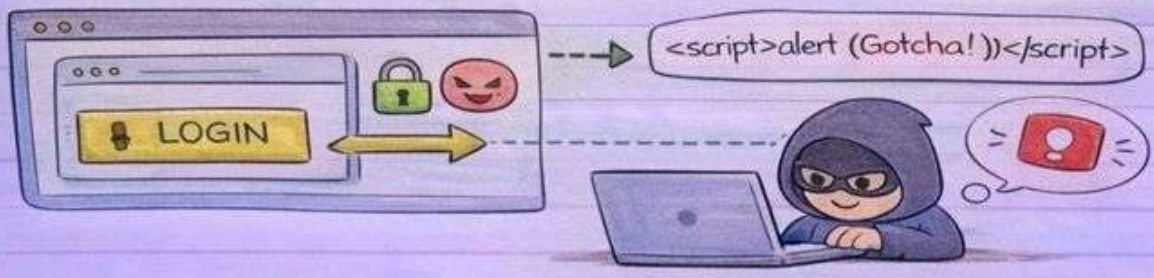
CSRF (Cross-Site Request Forgery)

- CSRF tricks a user into performing actions they didn't intend to by exploiting their authenticated session.

XSS (Cross-Site Scripting)

- XSS involves injecting malicious scripts into web pages viewed by other users.





@ curious_programmer

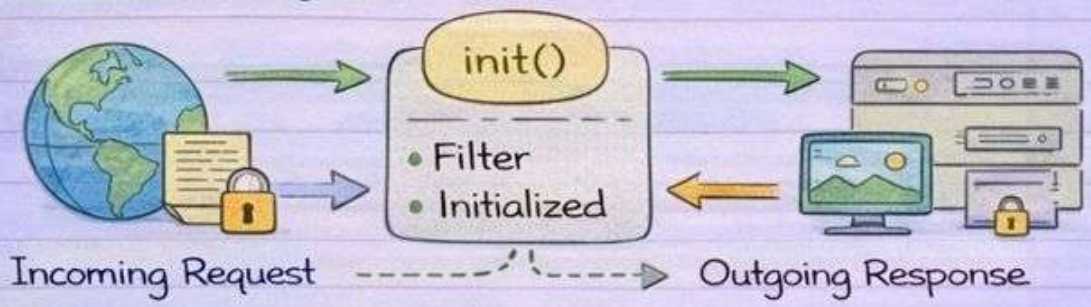
Chapter 8: Filters & Listeners

@ curious_programmer

✓ Filters

Filters are used to process and modify incoming requests or outgoing responses before they reach the target resource.

✓ Filter Life Cycle

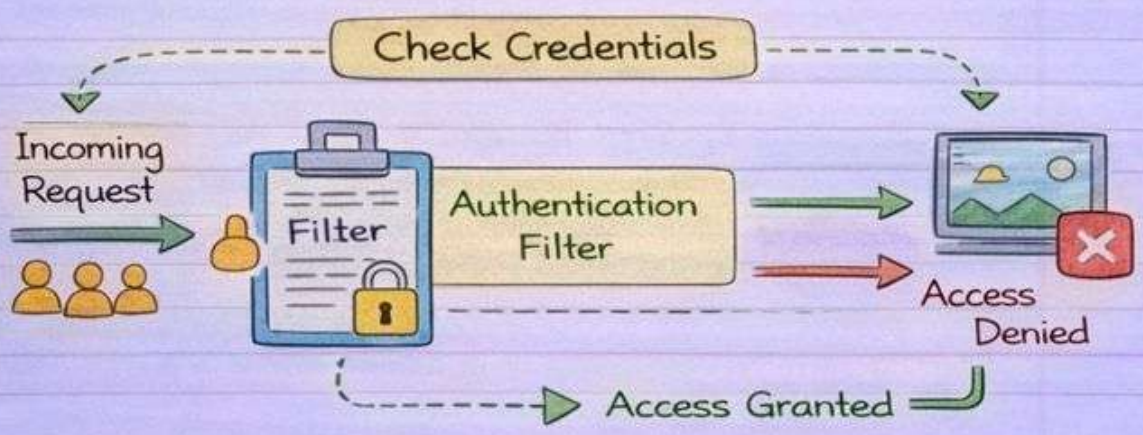


– A filter has three key stages:

1. Initialized: init() called once at startup.
2. Filtering: doFilter() called for every request or response.
3. Destroyed: destroy() called once when filter is removed.

✓ Authentication Filters

Authentication filters are used to verify and authenticate users before allowing them access.



Logging Filters & Compression Filters

@curious_programmer

✓ Logging Filters

- Capture and log information about requests and responses.
  - ✓ Logs details of each request, including URL, parameters, headers, timestamp, etc.
  - ✓ Logs details of each response, including status codes, content type, size, etc.
  - ✓ Helps in monitoring, debugging, and auditing web applications.

✓ Logging Filters

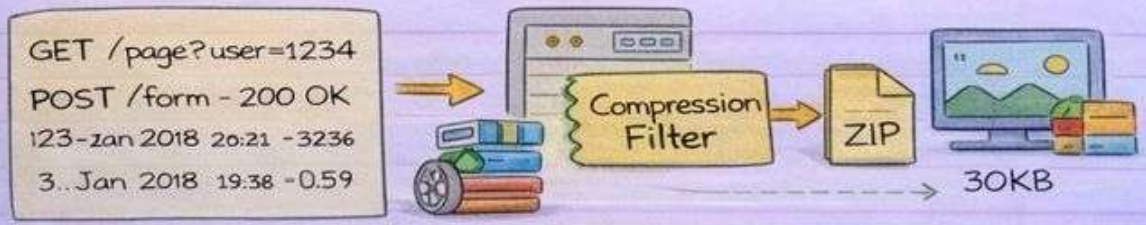
Logs requests & responses

✓ Compression Filters

- Compression Filters reduce the size of web content before sending it to the client.
  - ✓ It uses GZIP compression to make responses smaller in size.

Benefits include:

- ✓ Compresses responses, such. HTML, CSS, JS, JSON, reducing their size.
- ✓ Faster lood times by reducing the amount of data sent over the network.
- ✓ Saves bandwidth, especially beneficial for users with slow connections.



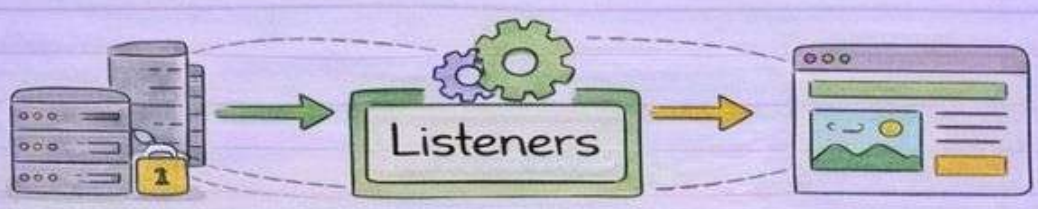
Compresses & reduces response size

Listeners

@ curious_programmer

✓ Listeners

- Listeners are classes that listen for specific events in a web application and respond when those events occur.



Execute custom code when specific events happen

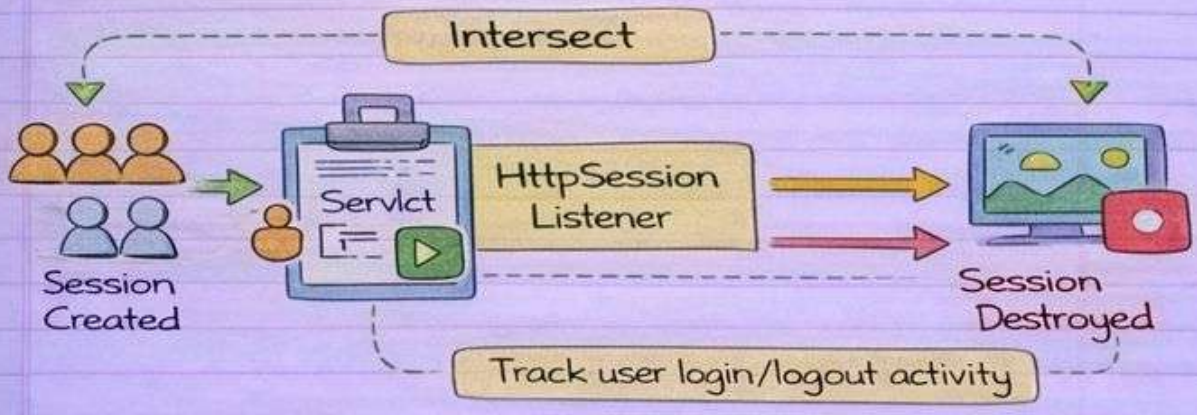
✓ ServletContextListener

- Listens for changes in the servlet context (application-scope).
- ✓ Context Initialized: Called when the web application starts.
- ✓ Context Destroyed: Called when the web application shuts down.

Set up resources or perform cleanup tasks

✓ HttpSessionListener

- Listens for events related to user sessions.
- ✓ Session Created: Called when a new user session is created.
- ✓ Session Destroyed: Called when a user session is invalidated or times out.



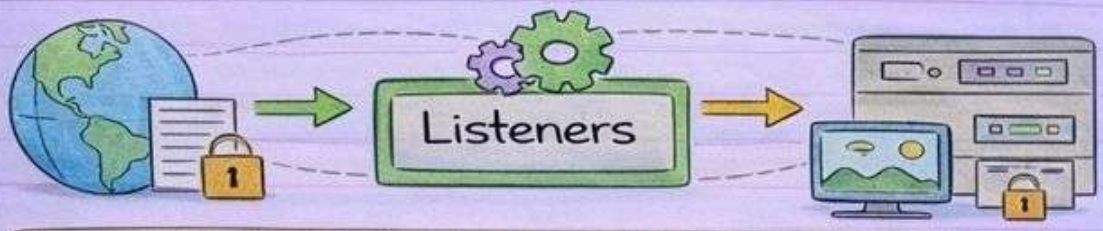
@ curious_programmer

Listeners

@ curious_programmer

✓ Listeners

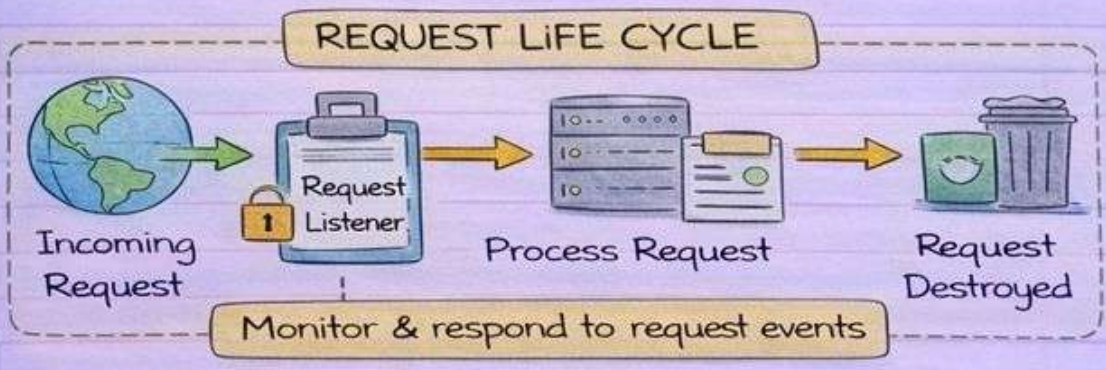
- Listeners are classes that listen for specific events in a web application and respond when those events occur.



Execute custom code when specific events happen

✓ Request Listeners

- Request listeners respond to events in the request life cycle.
  - ✓ Request Initialized: Called when a new request is received.
  - ✓ Request Destroyed: Called when a request is completed.



✓ Use Cases

- ✓ Log requests and responses for monitoring and auditing.
- ✓ Check authentication before processing a request.
- ✓ Track performance metrics like request processing time.
- ✓ Implement custom logic like setting attributes in requests.

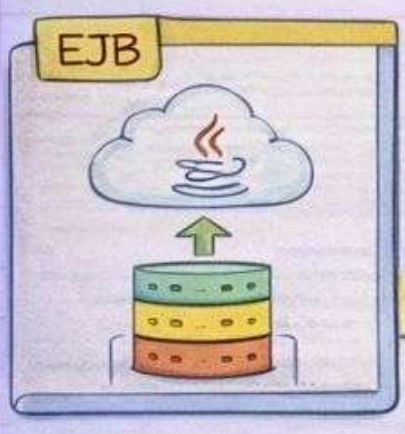
@ curious_programmer

Chapter 9: Enterprise JavaBeans (EJB – Overview)

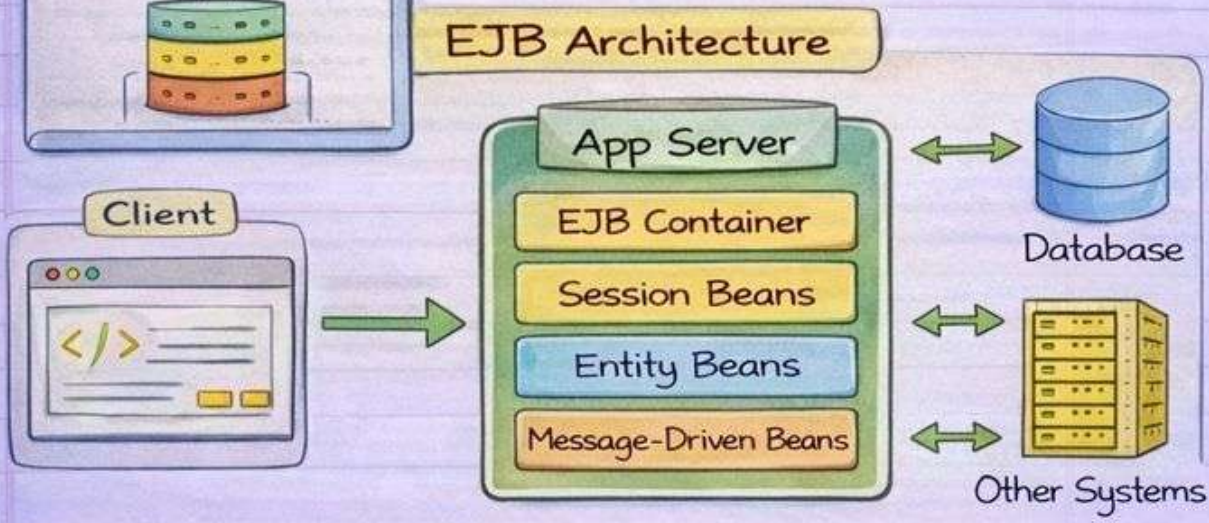
@ curious_programmer

✓ Introduction to EJB

✓✓ What is EJB?



EJB stands for Enterprise JavaBeans. It is a server-side component architecture used for building scalable, robust, and transactional business applications in Java.



- ✓ **Client**: The external application that interacts with the EJBs.
- ✓ **EJB Container**: The runtime environment within the app server that manages EJBs.
- ✓ **Session Beans**: Handle business logic.
- ✓ **Entity Beans**: Represent persistent data stored in a database.
- ✓ **Message-Driven Beans**: Process asynchronous messages.

- ☑ **Client**: The external application that interacts with the EJBs.
- ☑ **EJB Container**: The runtime environment within the app server that manages EJBs.
- ☑ **Session Beans**: Handle business logic.
- ☑ **Entity Beans**: Represent persistent data stored in a database.
- ☑ **Message-Driven Beans**: Process asynchronous messages.

Types of EJB:

@curious_programmer

Session Beans

Session beans are EJB components that implement business logic in an application. They run within the EJB container in an application server.

Stateless

✓ Stateless

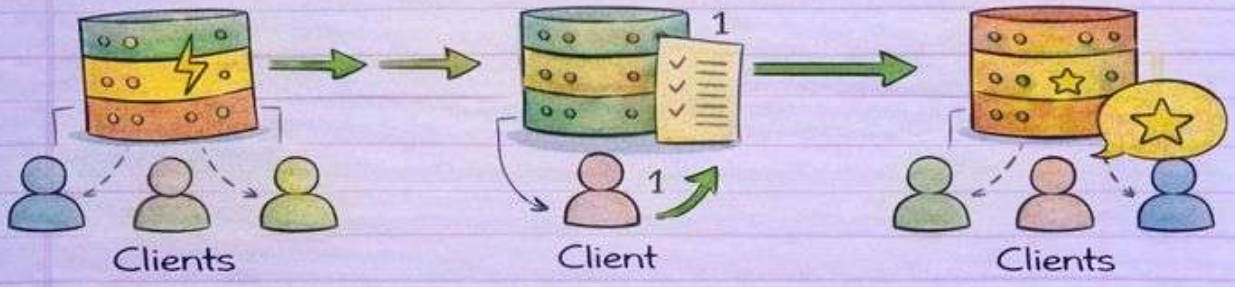
- ✓ Does not maintain any state between client calls.
- ✓ Each call is independent.

✓ Stateful

- ✓ Maintains state between client calls for a specific user.
- ✓ Remembers user interactions.

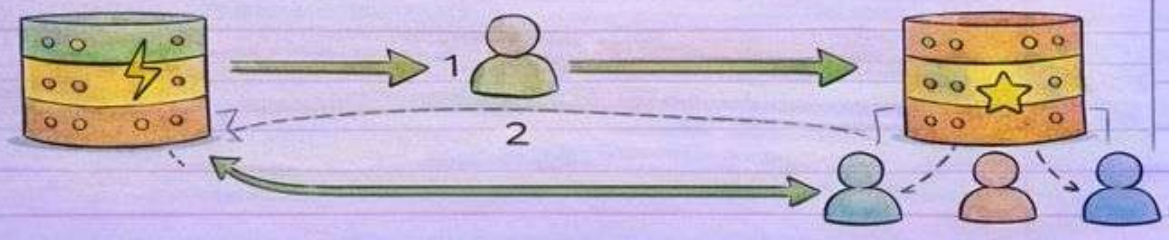
✓ Singleton

- ✓ A single, shared instance used by all clients.
- ✓ Maintains global state across users.



Stateful

- ✓ Maintains state between client calls for a specific user.
- ✓ Remembers user interactions.



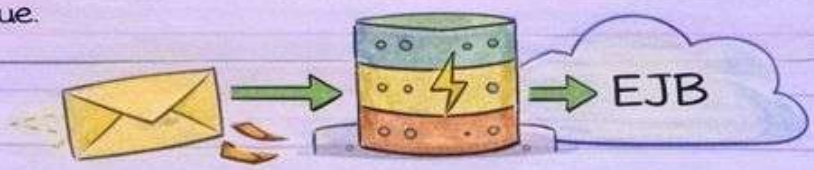
@curious_programmer

Message-Driven Beans & EJB Life Cycle

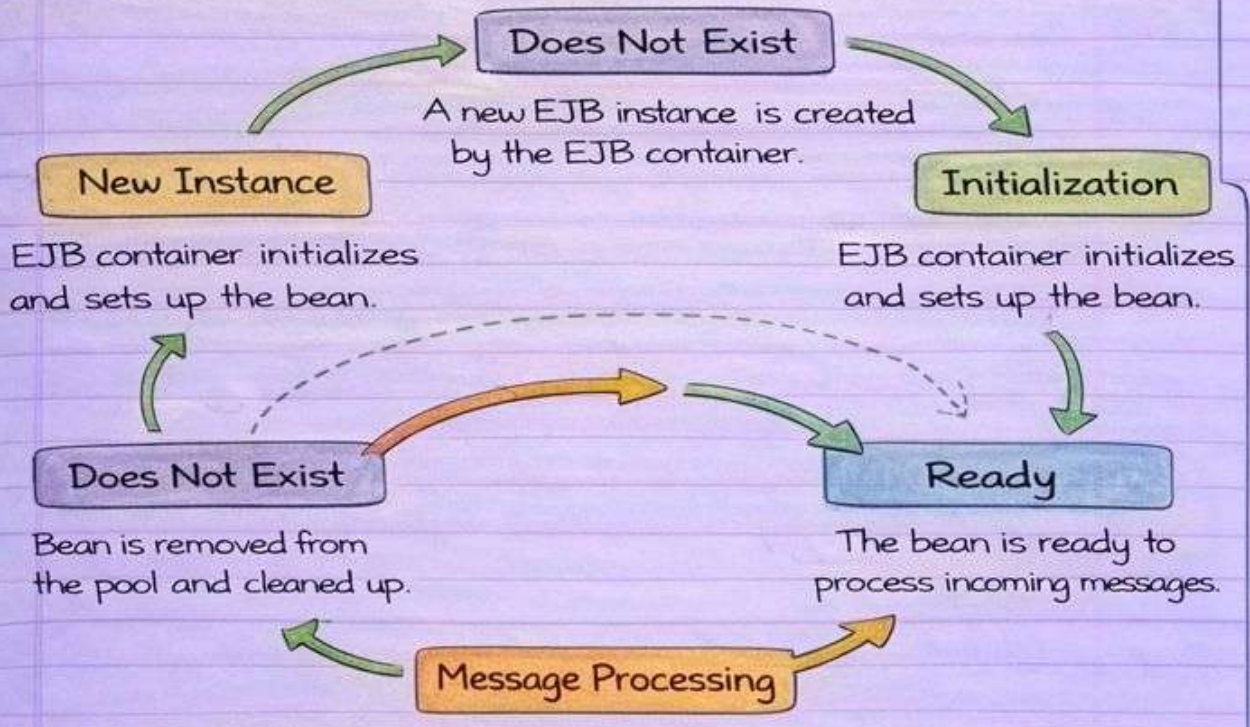
@ curious_programmer

Message-Driven Beans

Message-driven beans are EJB components that process asynchronous messages sent to the application from a message-oriented middleware. (MOM) or message queue.



EJB Life Cycle

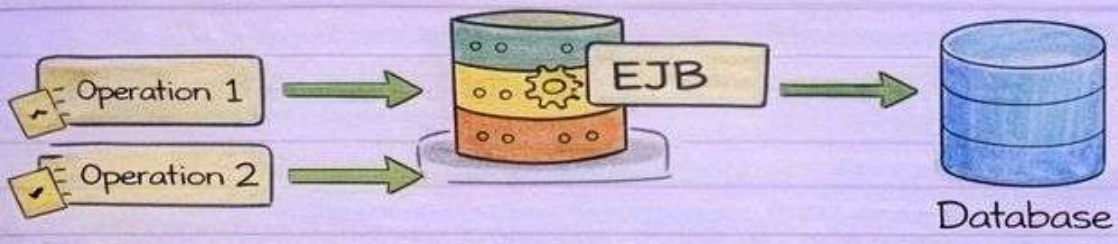


- ✓ **Does Not Exist**: Bean is removed from the pool and cleaned up.
- ✓ **New Instance**: A new EJB instance is created by the EJB container.
- ✓ **Initialization**: EJB container initializes and sets up the bean.
- ✓ **Ready**: The bean is ready to process incoming messages.
- ✓ **Message Processing**: The bean processes incoming messages from the message queue.

@ curious_programmer

Transactions in EJB

✔ Enterprise JavaBeans (EJB) use transactions to ensure data integrity and consistency across business operations.



✔ EJB transactions are automatically managed by the EJB container, ensuring ACID properties:

A Atomic	C Consistent	I Isolated	D Durable
All operations are treated as a single unit	Transactions leave data in a consistent state.	Intermediate states are not visible to other transactions	Once committed, changes are permanent.

EJB vs Spring (Overview)

	EJB	VS.	Spring
Framework Type	✔ Java EE server-side framework		✔ Lightweight, modular framework
Transactional Management	✔ Built-in container-managed transaction support		✔ Flexible transaction management with @Transactional annotation
Configuration	✔ XML-based configuration within application server		✔ Annotation-based configuration and XML support
Deployment	✔ Requires deployment in Java EE application server		✔ Can run in any Java environment, standalone or with a server
Learning Curve	✔ Moderate to steep		✔ Friendlier learning curve ☺

@ curious_programmer

Chapter 10: Java Persistence API (JPA) / Hibernate (Basics)

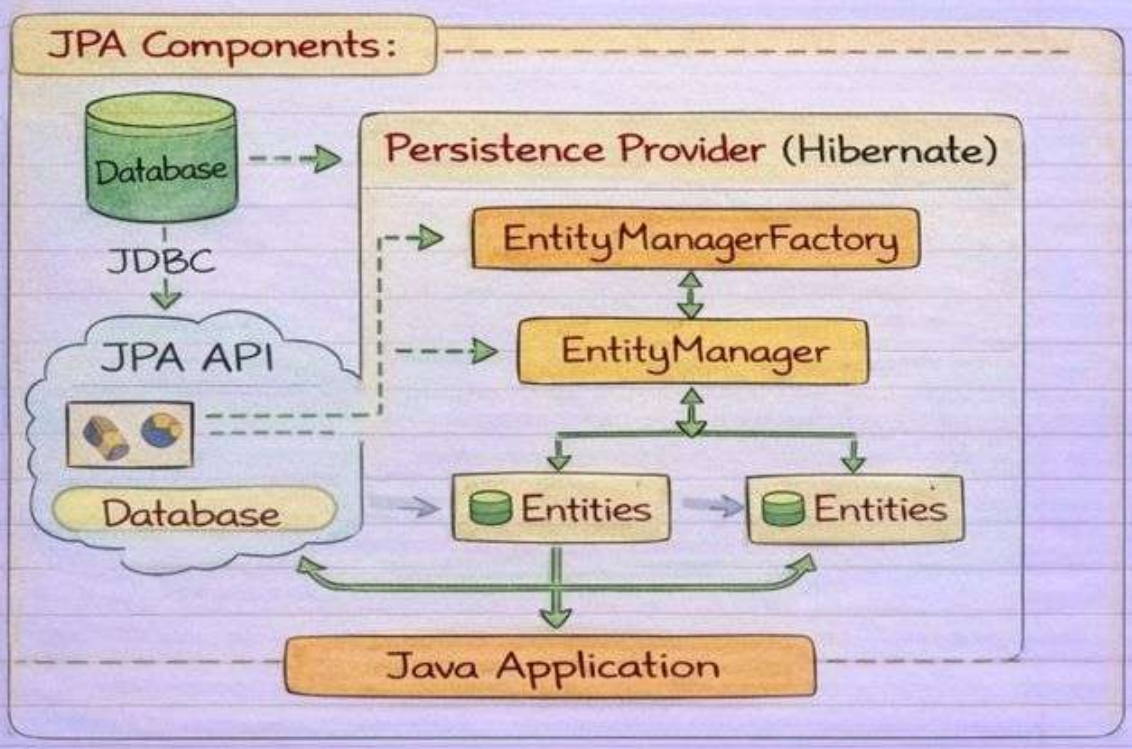
@ curious_programmer

✓ ORM Concepts

ORM stands for Object-Relational Mapping. It is a technique used to map Java objects to database tables.

ORM allows developers to interact with a database using Java objects instead of SQL queries.

✓ JPA Architecture



Entity Mapping

@ curious_programmer

✓ Annotations:

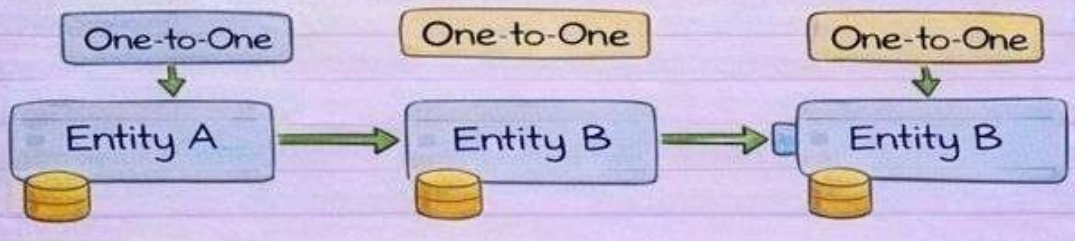
Annotations are used to define mappings between Java objects and database tables in JPA.

Entity + Table Mapping:

- @Entity: marks a class as a JPA entity.
- @Id: marks the primary key of the entity.
- @Table: specifies the table in the database.

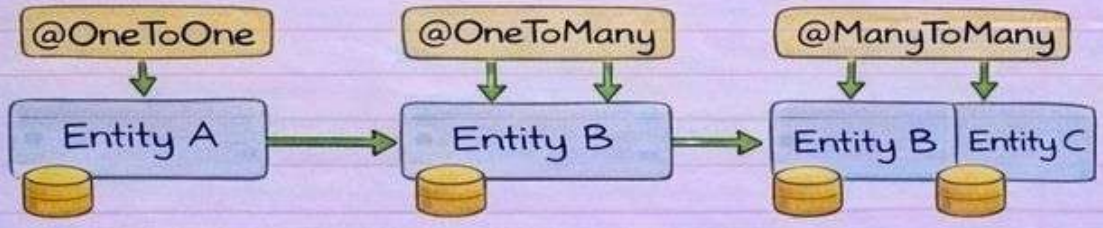
✓ Relationship Mappings:

✓ @OneToOne: defines a one-to-one relationship between two entities.



✓ @OneToMany defines a one-to-many relationship where an entity is related to many other entities.

✓ @OneToMany defines a many-to-many relationship between two entities.



✓ @ManyToMany defines a many-to-many relationship between two entities.

JPQL Hibernate Configuration

@curious_programmer

JPQL

- JPQL (Java Persistence Query Language) is an object-oriented query language for querying data from the database using entity objects.
- Similar to SQL but works with entity objects instead of tables.
- Enables retrieval of data based on entity attributes.
- Supports complex queries like joins and filtering using WHERE clause.

JPQL Query Example:

```
SELECT e FROM Employee e
WHERE e.salary > 50000
```

---> List<Employee>

Hibernate Configuration:

- Hibernate Configuration involves setting up the ORM framework to interact with the database.

Key configuration settings:

- Database Connection: hibernate.dialect (specifies the SQL dialect), database URL)
- Hibernate Properties: hibernate.hbm2ddl.auto
- Session Management: hibernate.show_sql (Logs SQL queries)
- hibernate.format_sql (Formats SQL output)

Caching Lazy vs Eager Loading

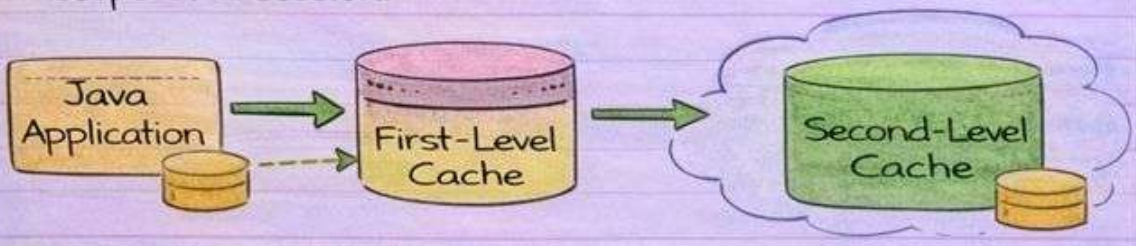
@curious_programmer

Caching

- Caching is used to improve performance by storing frequently accessed data in memory for quicker access.

First-Level Cache

- Associated with the Session and stores data within the scope of a session.



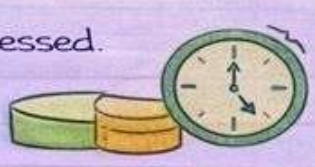
- Second-Level Cache: Shared across multiple sessions and stores data for the entire application.

Lazy vs Eager Loading

- Lazy and Eager loading are used to determine when related entities are loaded from the database.

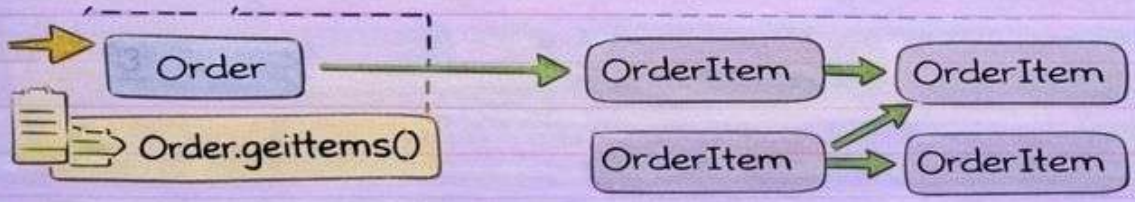
Lazy Loading

- Delays loading data until it is explicitly accessed.
  - Tip: More efficient for large data sets.



Eager Loading

- Loads all related entities immediately.



- May lead to performance overhead if not managed properly.

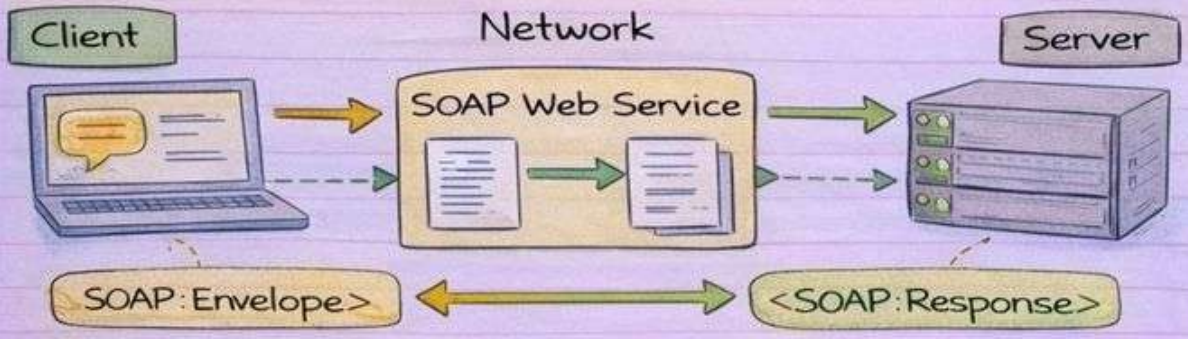
Chapter 11: Web Services

@curious_programmer

SOAP Web Services

- SOAP (Simple Object Access Protocol) is a protocol for exchanging structured information in distributed applications, based on XML.
- Uses XML for message format.
- Works over protocols like HTTP, SMTP, etc.
- Ensures platform and language independence.

SOAP Architecture



- Client sends a SOAP request (XML message).
- Server responds with a SOAP response (XML message).
- Works over protocols like HTTP, SMTP, etc.

WSDL

- WSDL (Web Services Description Language) is an XML-based language for describing the functionalities of a SOAP web service.
- Defines the service interface (methods, data types).
- Specifies the location of the service (URL).
- Describes the format of SOAP request and response messages.

@curious_programmer

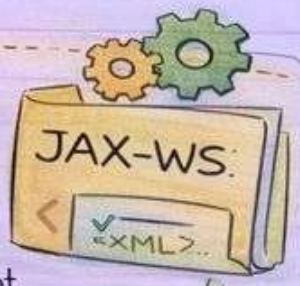
JAX-WS

Creating & Consuming SOAP Services

@ curious_programmer

JAX-WS

- JAX-WS (Java API for XML Web Services) is a Java API for building and consuming SOAP web ser-
- Standard part of Java EE (Enterprise Edition).
- Uses annotations to simplify web service development.
- Supports SOAP 1.1 and SOAP 1.2 specifications.



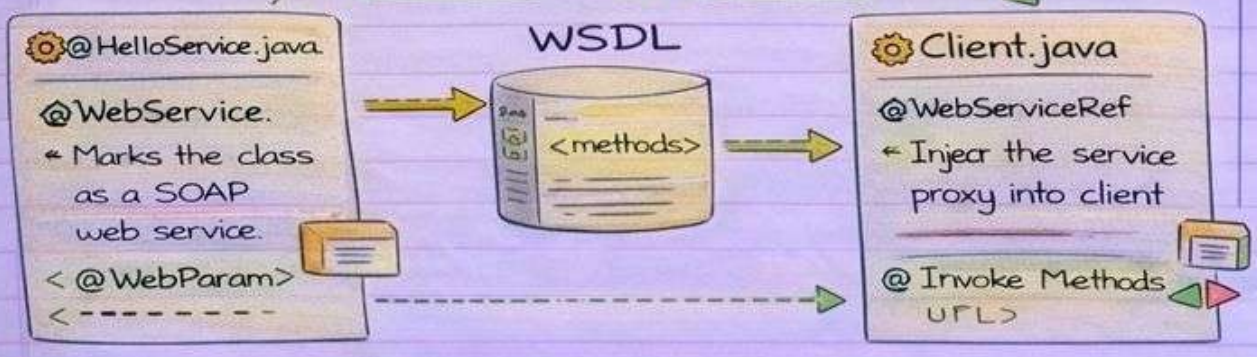
Creating & Consuming SOAP Services

Service (Creating) → **Client (Consuming)**

- **@WebService.**
  - Marks the class as a SOAP web service.
- **@WebMethod**
  - Defines a method as a web service operation.
- **@WebParam**
  - Specifies parameters for the web service.

- **@WebServiceRef**
  - Injects the service proxy into the client code.
- **Create Service**
  - Generates a service proxy class.
- **Invoke Methods**
  - Calls the web service methods using the proxy.

Server → **Client**



@ curious_programmer

RESTful Web Services

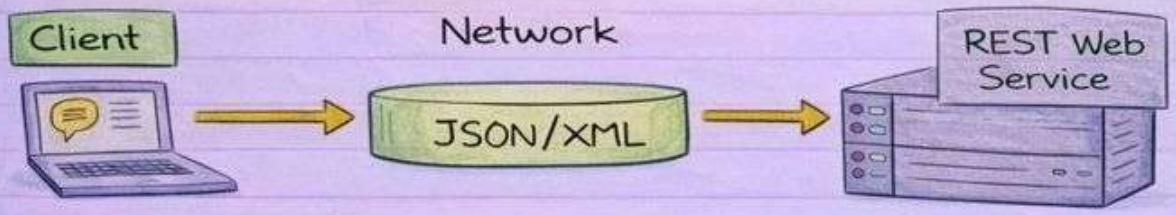
@ curious_programmer

RESTful Web Services

- ✓ REST (Representational State Transfer) is an architectural style that uses HTTP to build web services.
- ✓ Based on CRUD operations (Create, Read, Update, Delete).
- ✓ Uses stateless communication.

e.g, https://api.example.com/resource

REST Architecture



- ✓ Client sends HTTP requests to interact with resources.
- ✓ Server responds with data in formats like JSON or XML.

HTTP Methods

- ✓ RESTful web services use HTTP methods to perform CRUD operations.

GET →	READ: Retrieve data
POST →	CREATE: Add new data
PUT →	UPDATE: Modify existing data
DELETE →	DELETE: Remove data

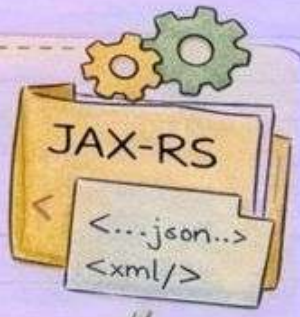
@ curious_programmer

JAX-RS

JSON & XML Handling

@ curious_programmer

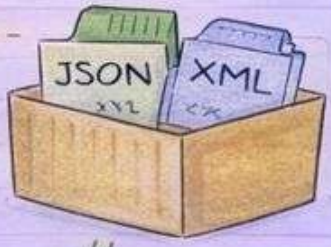
JAX-RS



- ✓ JAX-RS (Java API for RESTful Web Services) is a Java API for building RESTful web services in Java
- ✓ Part of Java EE (Enterprise Edition).
- ✓ Simplifies creation of RESTful web services.
- ✓ Uses annotations like @Path, @GET, @POST, etc.
- ✓ Supports JSON and XML data formats.

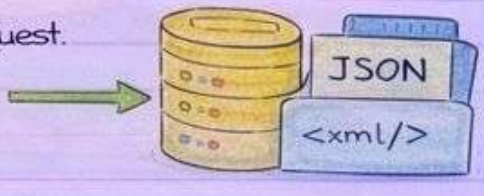
JSON & XML Handling

@Produces



- ✓ Specifies the response data format.
- ✓ Common values:
  - MediaType.APPLICATION_JSON
  - MediaType.APPLICATION_XML

@Consumes



- ✓ Specifies accepted data format in the request.
- ✓ Common values:
  - MediaType.APPLICATION_JSON
  - MediaType.APPLICATION_XML

- ✓ Using annotations like @Produces and Consumes makes handling JSON and XML data:
  - ✓ @Produces – Sets the output data type.
  - ✓ @Consumes – Sets the input data type.

@ curious_programmer

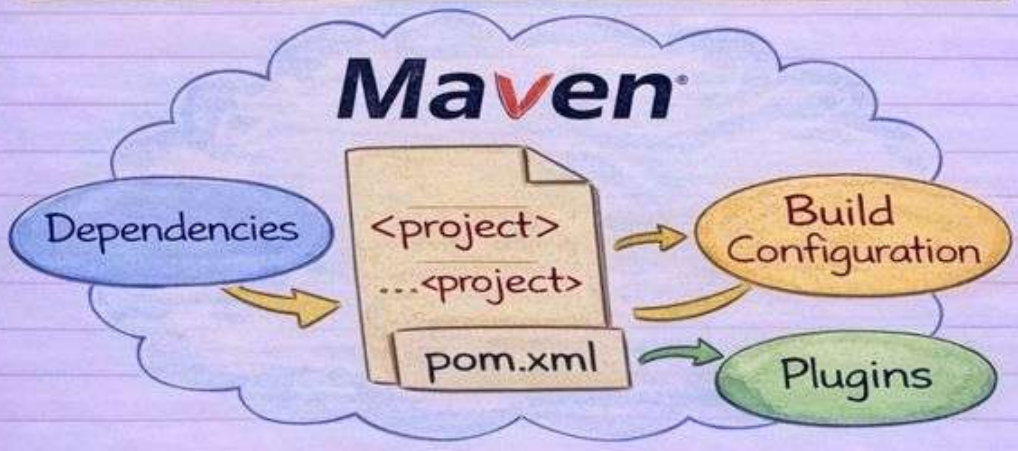
Chapter 12

Build & Dependency Management

Introduction to Maven

@curious_programmer

Project Object Model (POM)



- ✔ The POM is an XML file:
  - Defines the project structure and dependencies.
- ✔ Manages dependencies:
  - Specifies libraries and dependencies needed for the project.
- ✔ Configures build settings:
  - Describes how the project is built, including compiler settings.
- ✔ Uses plugins:
  - Includes plugins to perform tasks like compiling, testing, and packaging.

@curious_programmer

✓ Dependencies & Build Life Cycle

@ curious_programmer

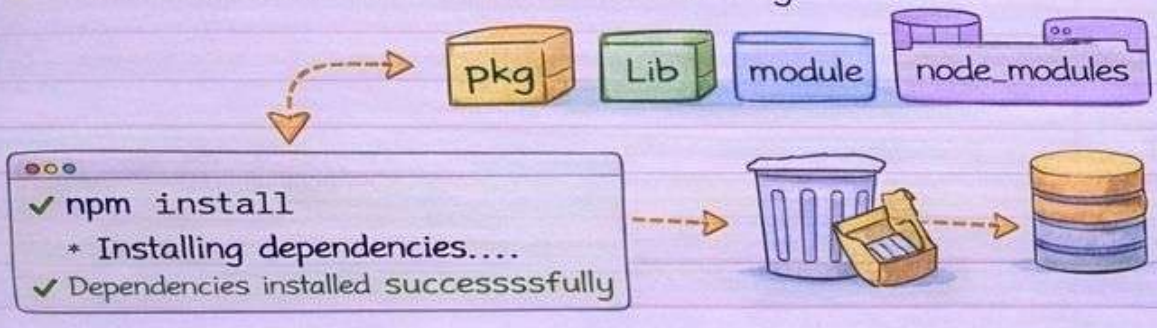
Dependencies

✓ Dependencies are external libraries or packages that a project needs in order to function properly.

✓ Help you leverage existing solutions

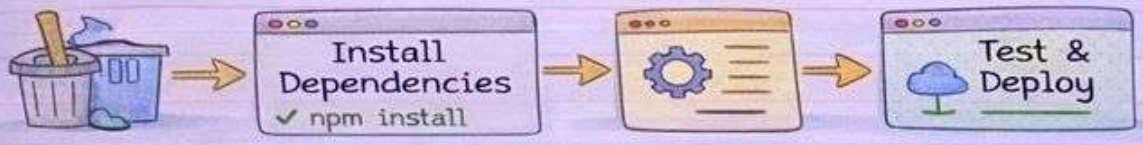
✓ Save development time.

✓ Ensure consistent and reliable functionality



Build Life Cycle

✓ The build life cycle is a process used to automate the tasks involved in building, testing, and deploying a project.



✓ Clean: Removes old files and prepares the project.

✓ Install Dependencies: Fetches and installs all required libraries.

✓ Build: Compiles the source code.

✓ Test & Deploy: Runs tests and deploy the project.

✓ Clean: Removes old files and prepares the project.

✓ Install Dependencies: Fetches and installs all required libraries.

✓ Build: Compiles the source code.

✓ Test & Deploy: Runs tests and deploys the project.

@ curious_programmer

✓ Plugins & Introduction to Gradle (Overview)

@ curious_programmer

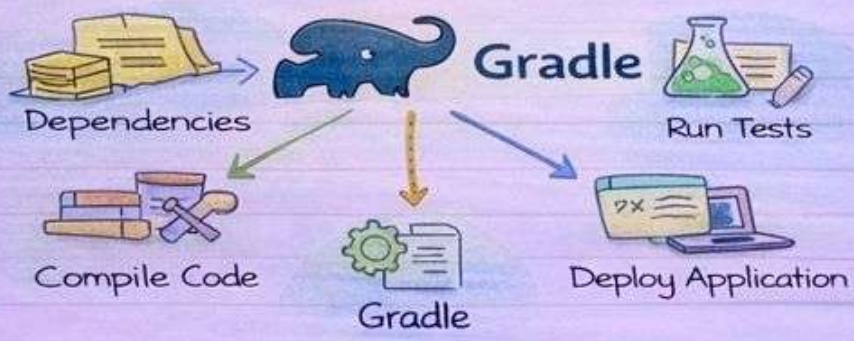
Plugins

- ✓ Plugins are additional pieces of code that extend and enhance the functionality of a build tool or application.
- ✓ Add new features or tasks (e.g, code linting, testing, packaging)
- ✓ Automate repetitive tasks.
- ✓ Available for many tools (e.g, Gradle, Maven, VS Code)


```
✓ gradle build deployTest
  - Generates build, deployment, and testing tasks.
```

Introduction to Gradle (Overview)

- ✓ Gradle is a powerful build automation tool used to define and manage the build process for software projects.
- ✓ Uses a Groovy- or Kotlin-based DSL (Domain Specific Language) to define build scripts (e.g, build.gradle, build.gradle.kts)
- ✓ Manages dependencies, compiles, tests, and deploys code.
- ✓ Highly flexible and customizable.




```
✓ gradle build
```

@ curious_programmer

✓Chapter13: Logging & Monitoring

@ curious_programmer

Logging Frameworks

Log4j

- Popular Java-based logging library.
- Highly configurable, allows, different logging outputs (e.g, console, files, database).
- Features built-in log levels to categorize logging messages.

SLF4J

- Simple Logging Facade for Java.
- Provides a unified logging API that works with various logging frameworks (e.g, Log4j, Logback).
- Allows you to switch between different logging frameworks without changing your application's main code.

Log Levels

✓ Log levels allow you to categorize and filter log messages based on their importance.

DEBUG	INFO	WARN	ERROR	FATAL
• Detailed information for debugging purposes.	• General information about the application's process.	• Potential problems that need attention	• Error events that prevent the application from fundioning correctly	• Severe errors that cause the application to crash.

```
✓gradle build
```

DEBUG	INFO	WARN	ERROR
• Detailed information for debugging purposes. ⚠			

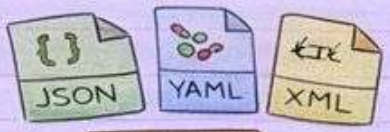
@ curious_programmer

✓ Configuration Files & Best Practices

@ curious_programmer

Configuration Files

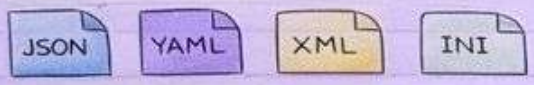
✓ Configuration files are used to manage settings and preferences for an application or system.



✓ Keep Settings Separate.

- Keep settings separate for code.

✓ Use Readable Formats (e.g, json, yaml)



✓ Document Settings

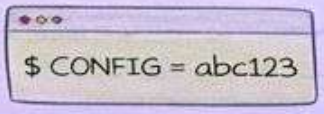
- Every setting should be well-documented.

✓ Secure Sensitive Data (e.g, API keys).

Best Practices

✓ Use Environment Variables

- Manage configuration settings via.

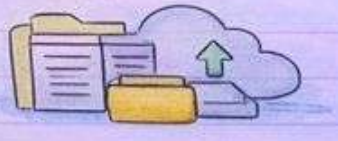


✓ Validate Configurations

- Validate configurations before use.

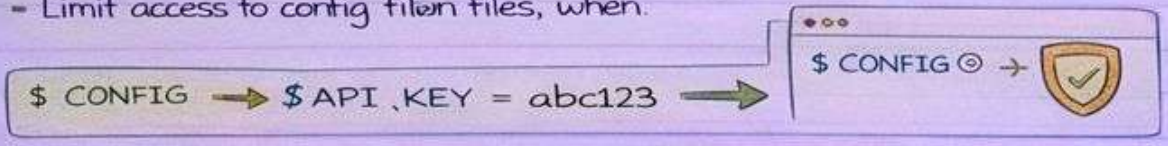
✓ Back Up Configuration Files

- Regularty back up configuration files.



✓ Apply the Principle of Least Privilege.

- Limit access to config filon files, when.



@ curious_programmer

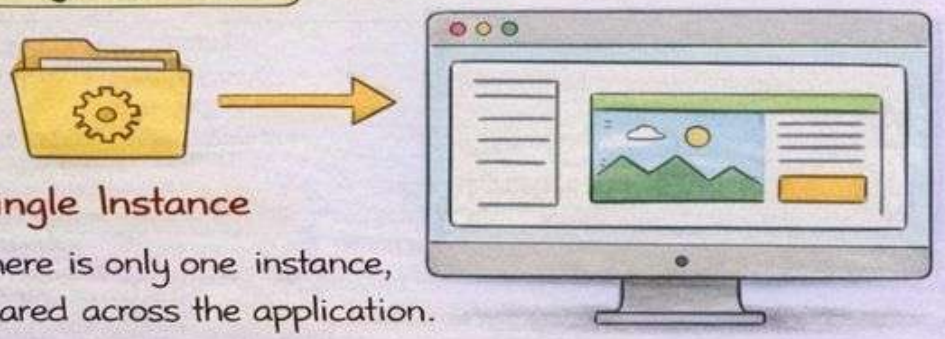
Chapter 14: Design Patterns (Web Focused)

@ curious_programmer

✓ Singleton

- Ensures that a class has only one instance and provides a way to access that instance.

✓ Singleton Class



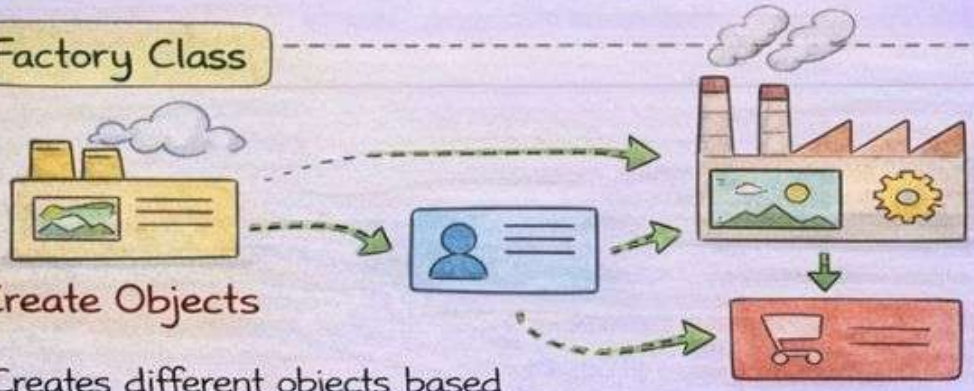
Single Instance

There is only one instance, shared across the application.

✓ Factory

- Provides an interface for creating instances of related or dependent objects without specifying the exact class to create.

✓ Factory Class



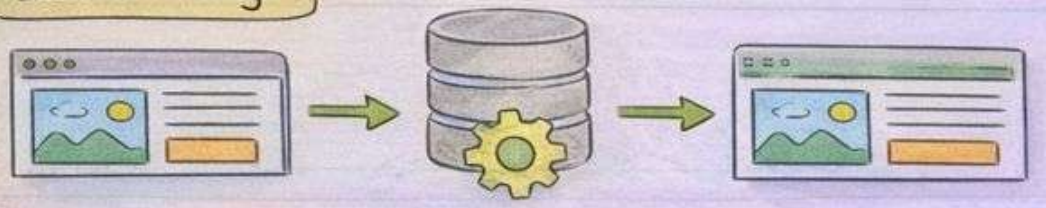
Create Objects

- Creates different objects based on the input parameters.

DAO

- Stands for Data Access Object. A design pattern that separates the data access logic from business logic.

Business Logic

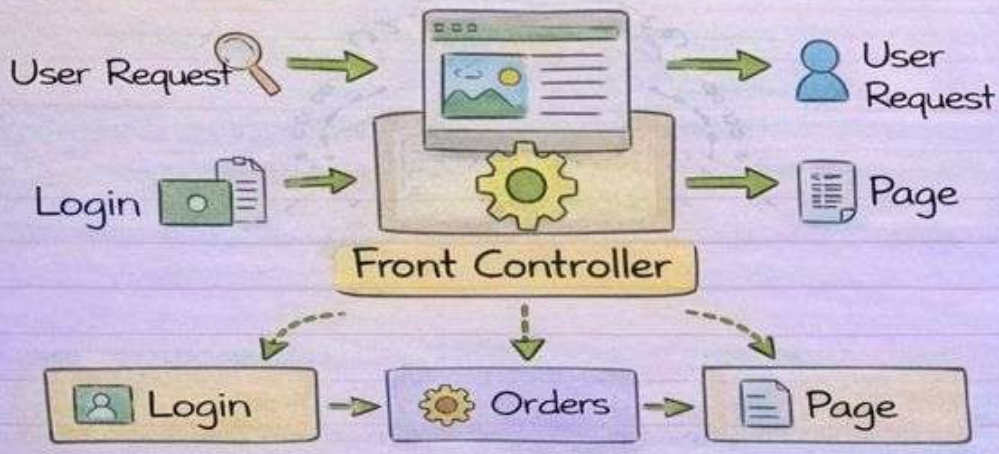


Manages and simplifies access to databases.

Front Controller

- Is a design pattern that provides a centralized entry point to handle all requests in a web application.

Front Controller

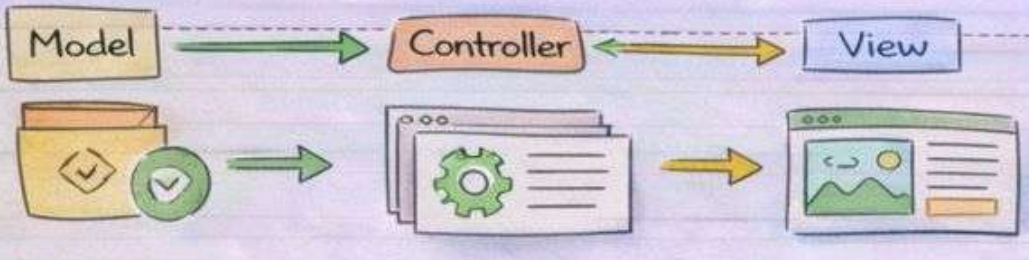


- Directs requests to appropriate handlers or views.

@ curious_programmer

MVC

- Stands for ModelView-Controller. A design pattern that separates an application into three interconnected components:



Model handles data and logic

Controller

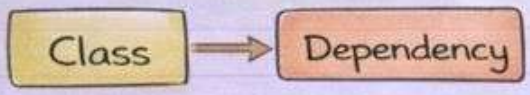
View displays data and Controller

Model handles data and logic, View displays data, and Controller proceses user input and updates the Model and View.

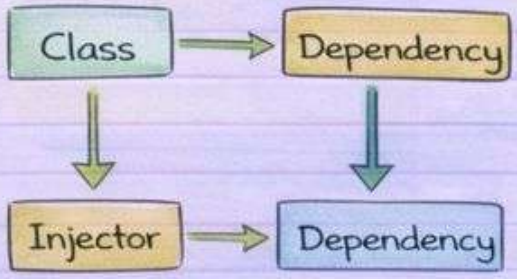
Dependency Injection (Concept)

- Is a concept where dependencies are injected into objects instead of the objects creating them.

Without Dependency Injection



With Dependency Injection



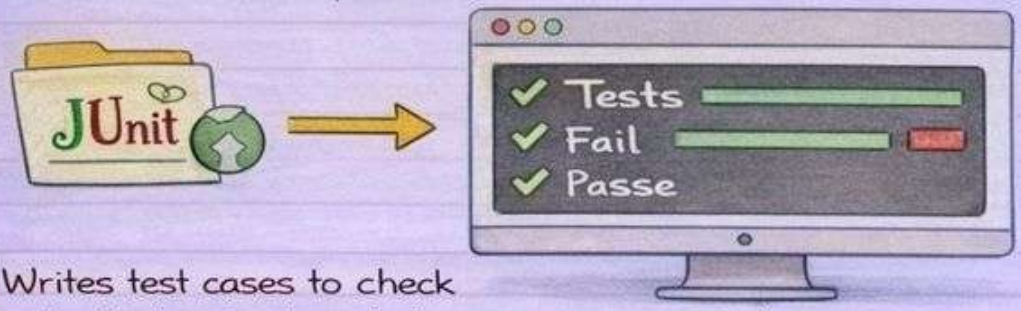
@ curious_programmer

Chapter 15: Testing

@ curious_programmer

✓ Unit Testing with JUnit

- A popular framework for unit testing in Java. It is used to write and run automated tests to ensure individual units of code work as expected.

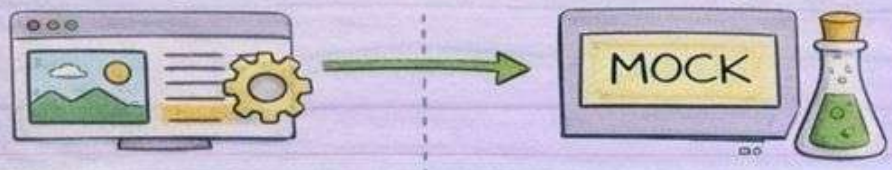


- Writes test cases to check individual methods and classes.
- Shows results with test passes and failures.

Tests

✓ Mockito (Basics)

- A popular framework for creating mock objects in Java. It is used to simulate the behavior of real objects.



Real Object

- Interact with actual database or services.

Mock Object

- Uses mocks to simulate database or services.
- Allows isolation of the unit being tested.

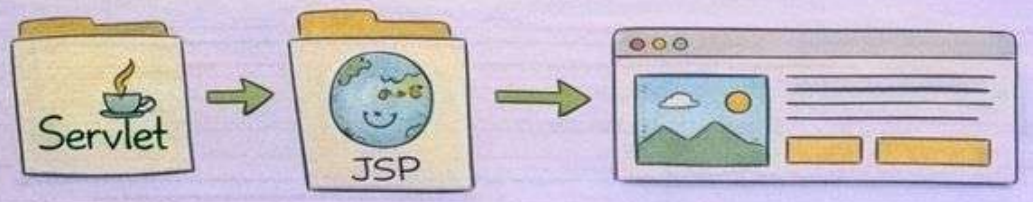
@ curious_programmer

Testing Servlets & JSP

@ curious_programmer

✓ Testing Servlets & JSP

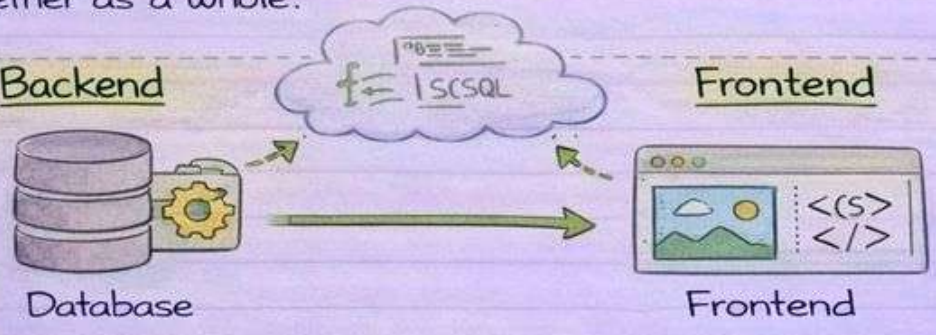
- Testing Java Servlets and JSPs (JavaServer Pages) involves testing the dynamic web components in a Java EE (Enterprise Edition) application.



- ✓ Testing request handling
- ✓ Testing session management
- ✓ Validating generated HTML content

✓ Integration Testing

- Involves testing how different parts of an application work together as a whole.



- ✓ Testing interactions between Backend & Frontend
- ✓ Verifying communication between components.
- ✓ Identifying issues at the integration points.

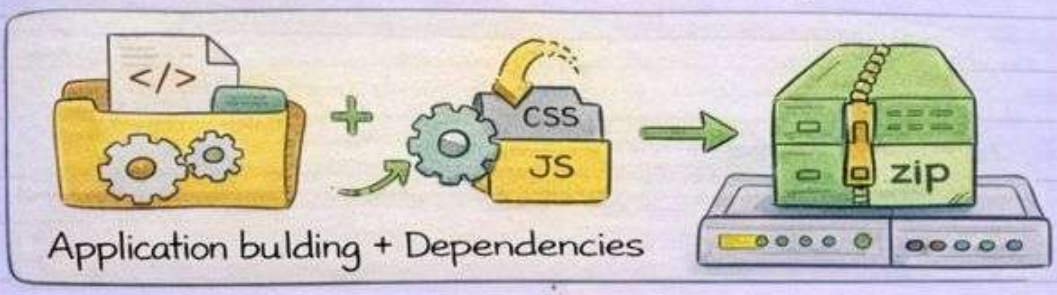
@ curious_programmer

Chapter 16: Deployment & Performance

@ curious_programmer

✓ Application Packaging

Application packaging involves bundling an application along with its dependencies into a single deployable file.



✓ Server Configuration

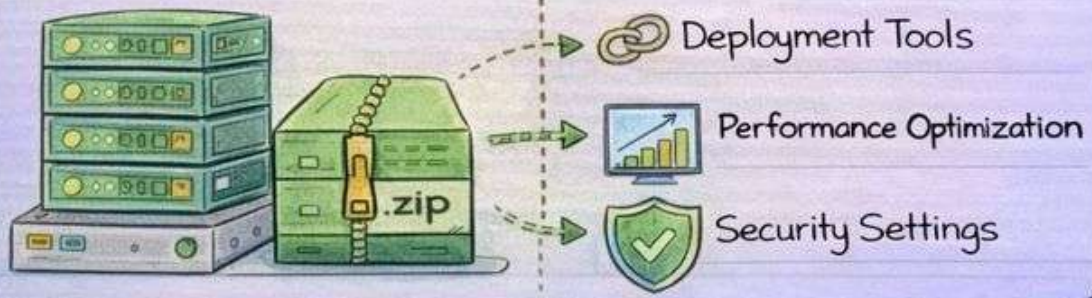
Server configuration involves setting up the server environment to host the web application efficiently.

✓ Application Packaging

Application packaging involves bundling an application along with its dependencies into a single deployable file.

✓ Server Configuration

Server configuration involves setting up the server environment to host the web application efficiently.



Load Testing Basics & Caching Strategies

@ curious_programmer

✔ Load Testing Basics

✔ Caching Strategies

Load testing involves simulating copies of frequently accessed data in a cache to speed up response times.

✔ Load Testing Basics

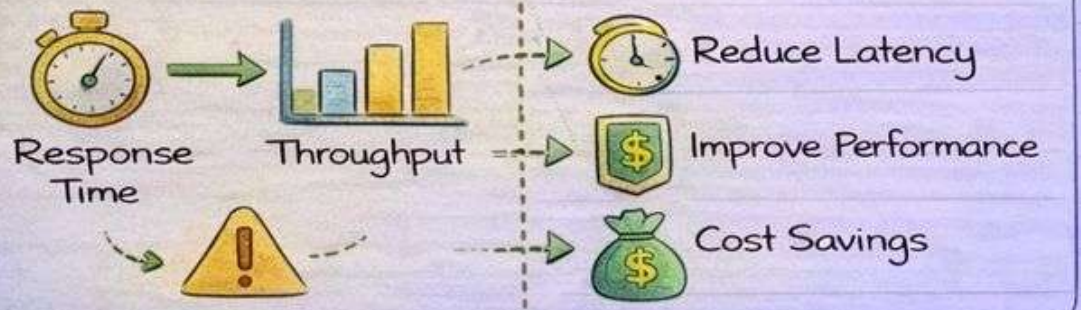
Load testing involves simulating multiple users accessing an application to measure its performance

Measure response times, throughput, and stability under heavy load.

✔ Caching Strategies

Caching strategies involve storing copies of frequently accessed data in a cache to speed up response times.

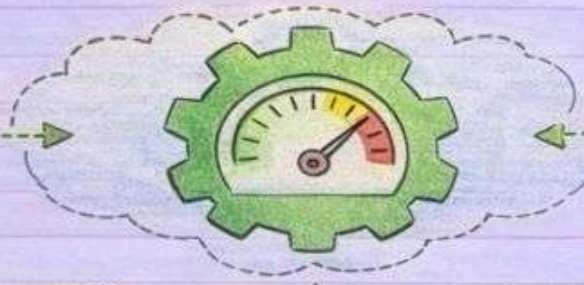
Cache stores static resources to reduce database load and latency.



Performance Optimization

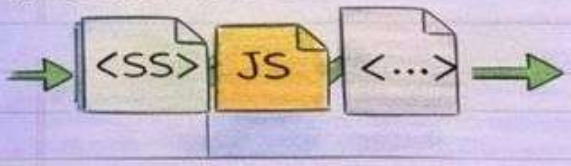
@ curious_programmer

✓ Performance optimization involves using techniques to improve the speed and efficiency of a web application.



✓ Code Minification

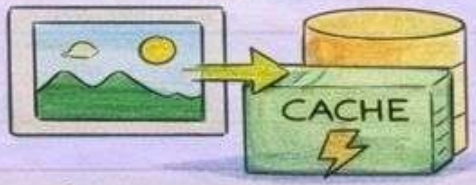
– Remove unnecessary whitespace, comments, and characters to reduce file size.



Remove unnecessary whitespace, comments, and characters

✓ Image Optimization

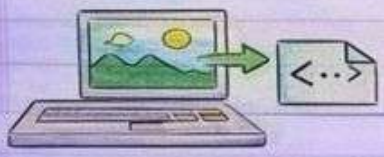
Compress and properly size images to speed up load times.



Store frequently accessed data in a cache to serve it faster.

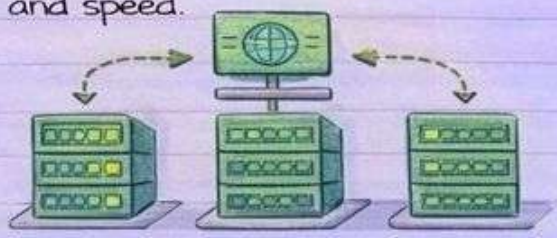
✓ Caching

– Store frequently accessed data in a cache to serve it faster.



✓ Load Balancing

– Distribute traffic across multiple servers to improve reliability and speed.



@ curious_programmer

Chapter 17: Mini Project / Capstone Project

@ curious_programmer

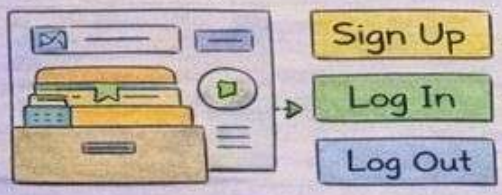
✓ CRUD Web Application

Build a web application that allows users to Create, Read, Update, and Delete (CRUD) records.

- ✚ Create
- 🔍 Read
- ✎ Update
- 🗑 Delete

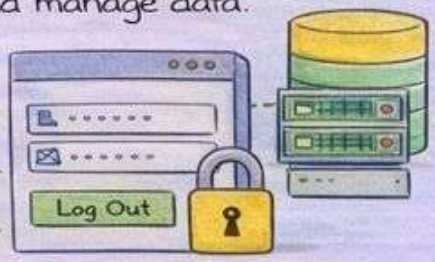
✓ Authentication System

Implement user authentication to secure the web application by allowing users to sign up, log in, and log out.



✓ Database Integration

Integrate the web application with a database to store and manage data.

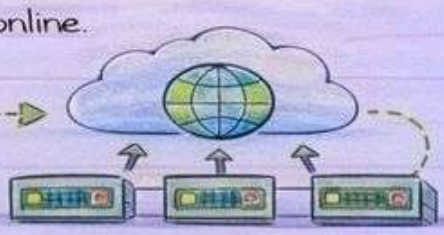


✓ MVC Architecture

Follow the Model-View-Controller (MVC) architecture for organizing your code.

✓ Deployment on Server

- Deploy the application to a live server so it is accessible online.



@ curious_programmer