ENDTERM PAPER SOLUTION 2024 - ADV JAVA SIMPLIFIED

Q1. All Question are compulsory

(a) Differentiate between core java and advanced java. (5)

Feature	Core Java (J2SE)	Advanced Java (JEE)
Definition & Purpose	Foundation; basic language features, OOP, I/O, collections. For desktop/CLI apps.	Specs/APIs on Core Java; for large-scale, distributed, multi-tiered enterprise apps.
Scope	Single-tier, desktop-focused.	Multi-tier, web/network-focused, server-side.
Technologies Included	Basic syntax, `java.lang`, `java.io`, `java.util`, `java.net`, Swing, AWT, JavaBeans.	Core Java + Servlets, JSP, EJB (hist.), RMI, JPA, Web Services, JMS.
Application Type	Standalone apps, applets (hist.).	Web apps, enterprise apps, distributed systems.
Complexity Management	Fundamental programming constructs.	Simplifies managing many users, network security, system integration, n-tier architecture.
Key Idea	Language basics.	Tools/frameworks for complex scenarios (websites, online services).

(b) State and explain the types of cookies in servlets. (5)

1. **Non-Persistent Cookie (Session Cookie):**
   - Default if `setMaxAge()` not called or set to `-1`.
   - Stored in browser memory; valid for current session; discarded on browser close.
   - Usage: Temporary session info.
2. **Persistent Cookie:**
   - Created with `setMaxAge(positive_seconds)`.
   - Stored on user's hard drive; valid until expiration.
   - Usage: Preferences, "remember me," tracking.
3. **Deleting a Cookie:**
   - Achieved by `setMaxAge(0)` on a cookie with the same name.
   - Browser immediately deletes it.

    - Usage: Explicit removal (e.g., logout).

(c) List out and explain the features of JSP. (5)

1. **Servlet Extension:** Compiled into Servlets, inherits Servlet power (platform independence, scalability).
2. **Simplified Dynamic Pages:** HTML-like structure with embedded Java; easier for layout.
3. **Improved Presentation/Logic Separation:** Especially with EL & JSTL.
4. **Reusable Components:** JavaBeans, Custom Tags (e.g., JSTL).
5. **Platform Independence:** Inherited from Java.
6. **Java API Access:** Full access (JDBC, RMI, etc.).
7. **Implicit Objects:** Predefined objects (`request`, `response`, `session`, `out`, etc.).
8. **Custom Tag Extensibility:** Developers create reusable tags.

(d) How JSP is more advantageous than Servlet. (5)

1. **Easier Page Layout:** HTML-like structure vs. programmatic HTML in Servlets.
2. **Better Presentation/Logic Separation:** Clearer distinction (especially with EL/JSTL).
3. **Reduced Java in View:** Declarative tags (EL/JSTL) replace scriptlets.
4. **Faster Presentation Development:** Modifying HTML-like structures is quicker.
5. **Component Reusability:** Standard actions & custom tags directly in page.

(e) Explain Hibernate framework and how it is related to ORM tool. (5)

Hibernate Framework: Open-source Java ORM framework. Automates mapping Java objects to relational database tables, reducing JDBC code. Implements JPA, offers HQL, caching, auto table creation.

Relation to ORM tool: Hibernate **is** an ORM tool. ORM is a technique mapping objects to relational data; Hibernate provides a concrete implementation of this technique.

UNIT-I

Q2. (a) Write a java program to demonstrate the concept of socket programming. (6.5)

```java
// SimpleServer.java
import java.io.*; import java.net.*;
public class SimpleServer {
    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(5000)) {
            System.out.println("Server waiting for client on port 5000...");
            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected.");
            try (PrintWriter out = new
PrintWriter(clientSocket.getOutputStream(), true);
                 BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()))) {
                String inputLine = in.readLine();
                System.out.println("Client says: " + inputLine);
                out.println("Server says: Hello Client, got - " +
inputLine);
            }
        } catch (IOException e) { System.err.println("Server exception: " +
e.getMessage()); }
    }
}

// SimpleClient.java
import java.io.*; import java.net.*;
public class SimpleClient {
    public static void main(String[] args) {
        try (Socket socket = new Socket("localhost", 5000);
             PrintWriter out = new PrintWriter(socket.getOutputStream(),
true);
             BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
             BufferedReader stdIn = new BufferedReader(new
InputStreamReader(System.in))) {
            System.out.println("Connected to server. Enter message:");
            String userInput = stdIn.readLine();
            out.println(userInput);
            System.out.println("Server response: " + in.readLine());
        } catch (IOException e) { System.err.println("Client exception: " +
e.getMessage()); }
    }
}
```

(b) Discuss the advantages, disadvantages and hierarchy of applets. (6)

Applets: Java programs embedded in HTML, run in browsers.
Advantages: Dynamic content, cross-platform (hist.), rich GUI, client-side processing.
Disadvantages: Requires plugin, security concerns, slow startup, deprecated, complex for simple tasks.
Hierarchy: `Object` -> `Component` -> `Container` -> `Panel` -> `java.applet.Applet`.

Q3. (a) Explain the basic steps of implementing a server with basic methods used in each step. (6.5)

1. **Create `ServerSocket`:** `new ServerSocket(port);` (Listens on a port).
2. **Accept Connection:** `serverSocket.accept();` (Waits for client, returns `Socket`).
3. **Get Streams:** `clientSocket.getInputStream();`, `clientSocket.getOutputStream();` (For I/O with client).
4. **Communicate:** Read from input stream, write to output stream.
5. **Close Client Socket:** `clientSocket.close();` (Release client resources).
6. **Close `ServerSocket`:** `serverSocket.close();` (On server shutdown).

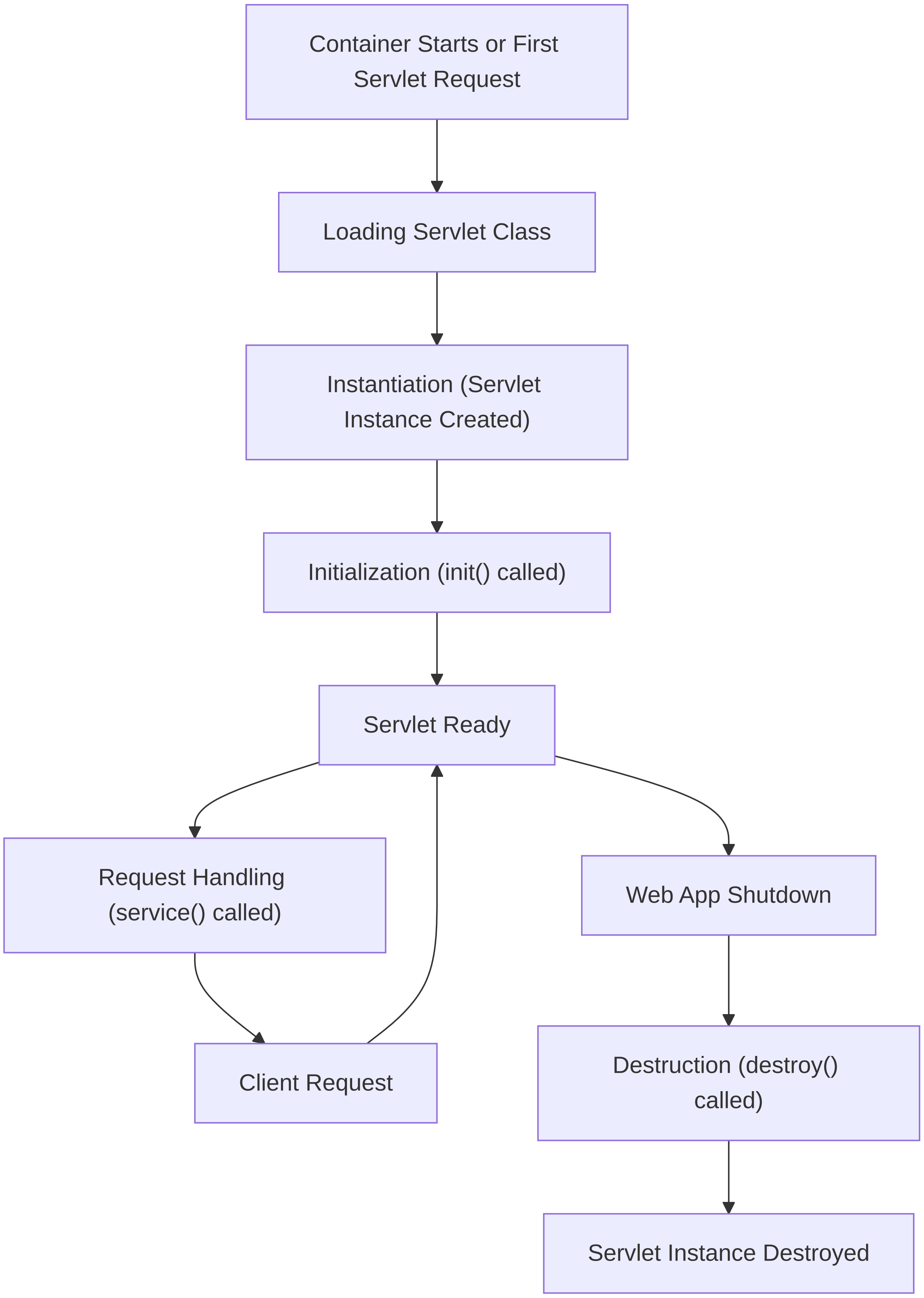
(b) Write a program in java to demonstrate the concept of applets. (6)

```java
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;
/* <applet code="MyApplet.class" width="200" height="100"></applet> */
public class MyApplet extends Applet {
    public void paint(Graphics g) {
        g.setColor(Color.blue);
        g.fillRect(20, 20, 150, 50);
        g.setColor(Color.white);
        g.drawString("Hello Applet!", 50, 50);
    }
}
```

UNIT-II

Q4. (a) Explain the lifecycle of a servlet with an example. (6.5)

1. **Loading:** Container loads servlet class.
2. **Instantiation:** Container creates servlet instance (once).
3. **Initialization (`init()`):** Called once for setup.
4. **Request Handling (`service()` -> `doGet`/`doPost`):** Called per request.
5. **Destruction (`destroy()`):** Called once for cleanup on shutdown.


Container Starts or First Servlet Request
Loading Servlet Class
Instantiation (Servlet Instance Created)
Initialization (init() called)
Servlet Ready
Request Handling (service() called)
Client Request
Web App Shutdown
Destruction (destroy() called)
Servlet Instance Destroyed

Example Code:

```
import javax.servlet.*; import javax.servlet.http.*; import java.io.*;
public class MyLifecycleServlet extends HttpServlet {
    public void init() throws ServletException { log("Servlet
Initializing"); }
```

```
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
throws IOException {
        res.getWriter().println("Hello from MyLifecycleServlet!");
log("doGet called");
    }
    public void destroy() { log("Servlet Destroying"); }
}
```

(b) Write a java program to demonstrate the use of Java Beans. (6)

JavaBean (`ProductBean.java`):

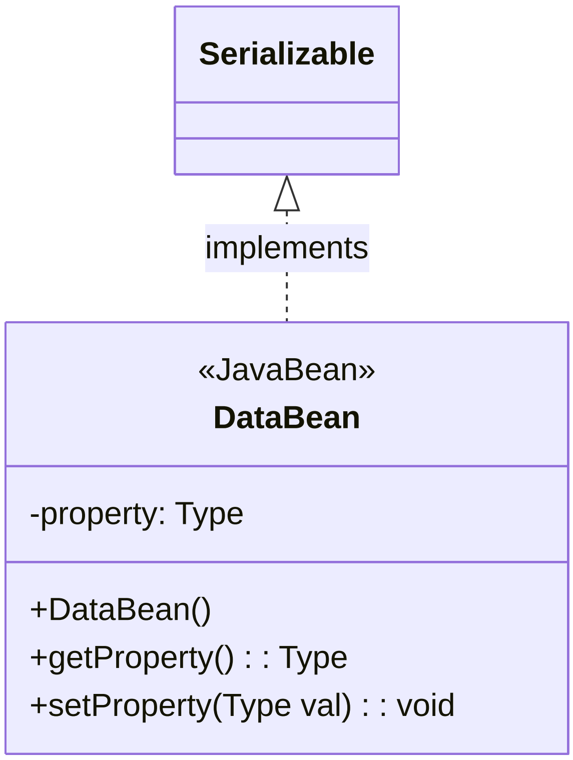
```
package com.example; import java.io.Serializable;
public class ProductBean implements Serializable {
    private String name; private double price;
    public ProductBean() {}
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public double getPrice() { return price; }
    public void setPrice(double price) { this.price = price; }
}
```

Using Class (`TestProductBean.java`):

```
package com.example;
public class TestProductBean {
    public static void main(String[] args) {
        ProductBean product = new ProductBean();
        product.setName("Laptop"); product.setPrice(1200.99);
        System.out.println("Product: " + product.getName() + ", Price: $" +
product.getPrice());
    }
}
```

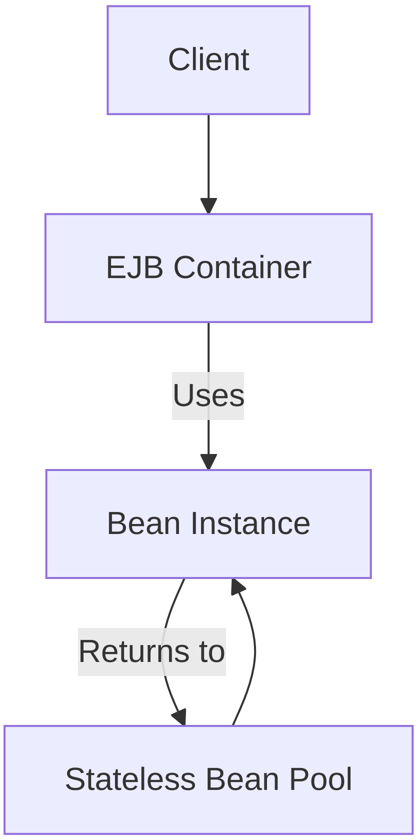
Q5. Discuss the types of Java Beans with a diagram of each type. (12.5)

I. Standard JavaBeans: Reusable components via conventions.
* **Role:** Often data carriers (DTOs/VOs).
* **Diagram (Data Bean):**

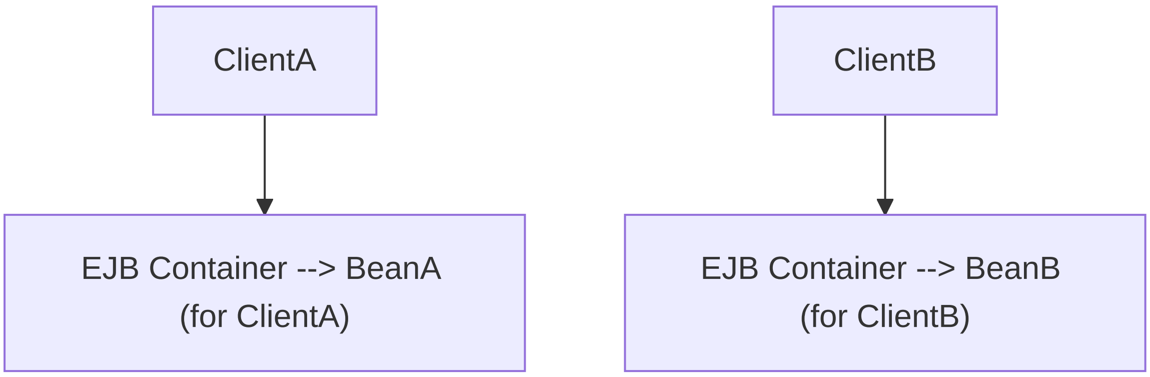


II. Enterprise JavaBeans (EJBs): Server-side components in EJB container.

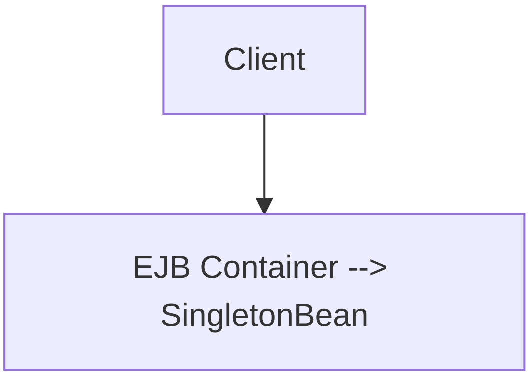
1. **Session Beans:** Business logic.
    - **Stateless:** No client state, pooled.



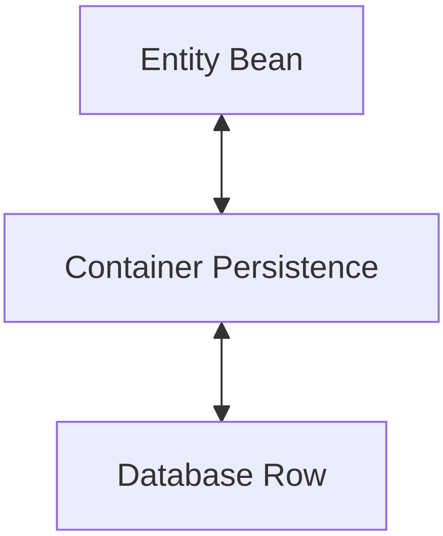
    - **Stateful:** Client-specific state, dedicated instance per session.


ClientA
ClientB
EJB Container --> BeanA (for ClientA)
EJB Container --> BeanB (for ClientB)

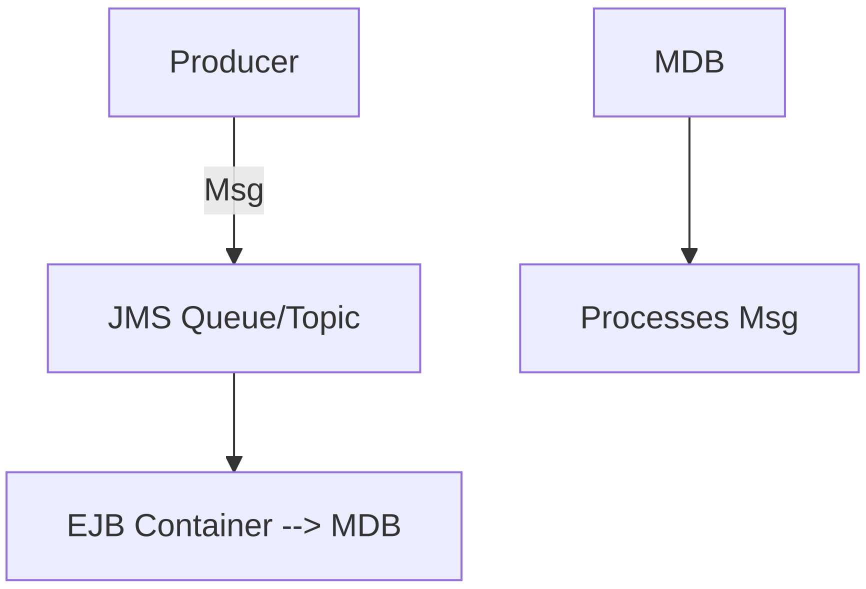
- **Singleton:** One instance for entire application.



2. **Entity Beans (Historical):** Represented persistent DB data. (Largely replaced by JPA).



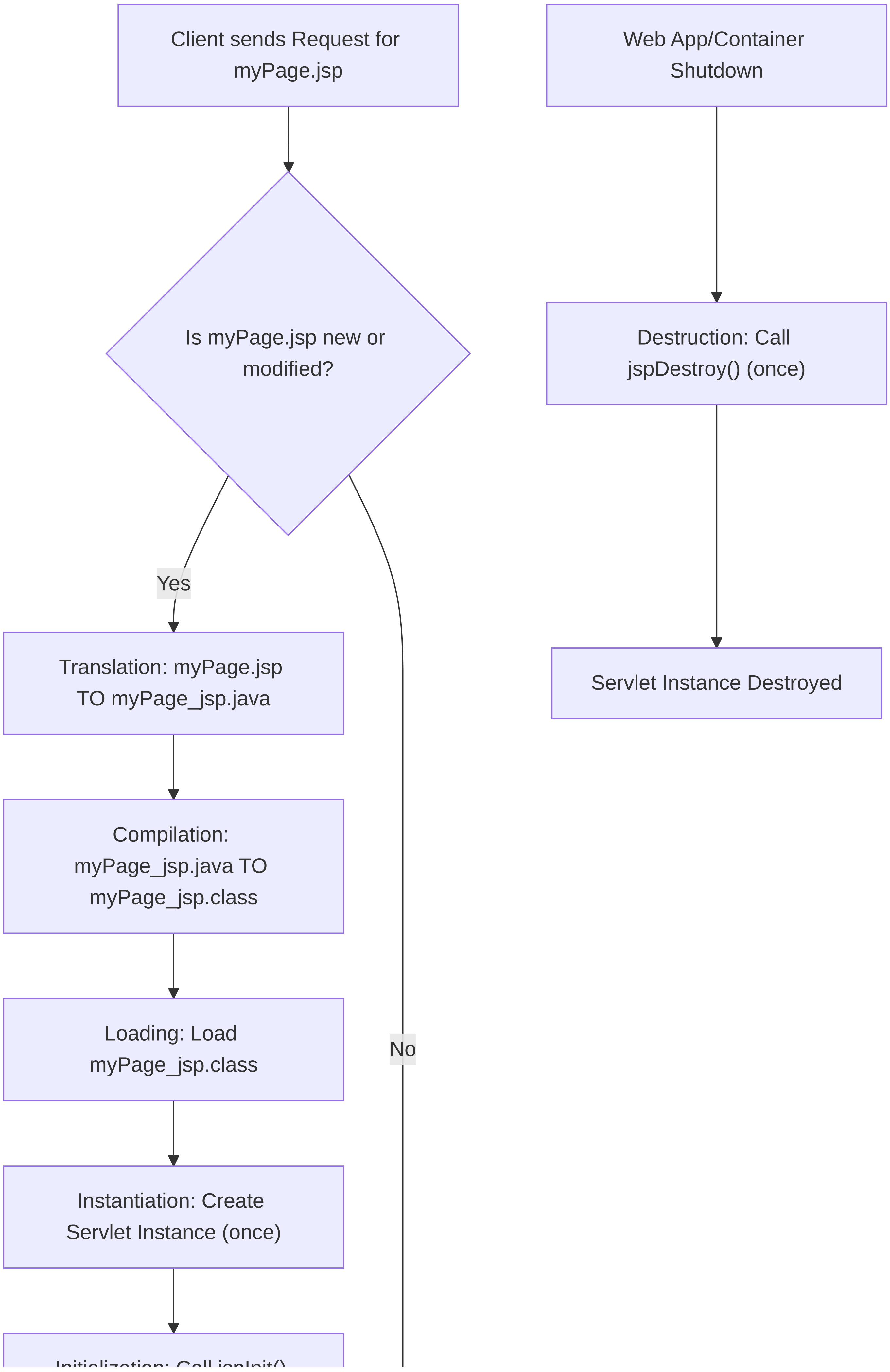
3. **Message-Driven Beans (MDBs):** Asynchronous message consumers (JMS).

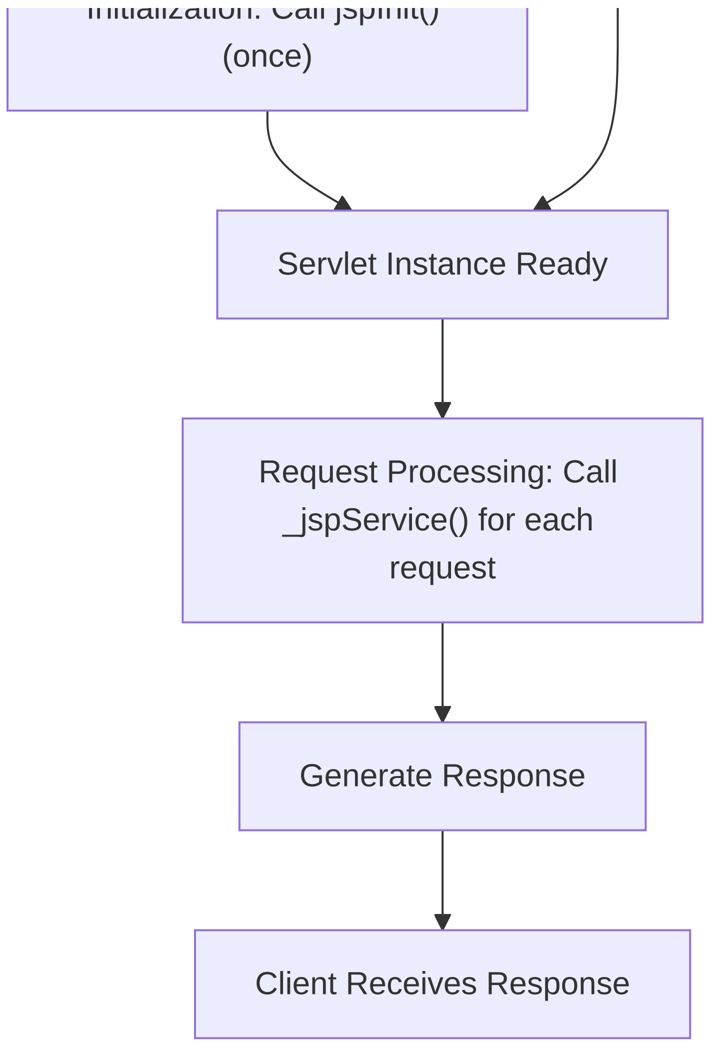


UNIT-III

Q6. Explain the lifecycle of a JSP page with a diagram. (12.5)

1. **Translation:** `.jsp` to `.java` servlet (first request/if modified).
2. **Compilation:** `.java` to `.class` (bytecode).
3. **Loading:** Servlet `.class` loaded.
4. **Instantiation:** Servlet instance created (once).
5. **Initialization (`jspInit()`):** Called once for setup.
6. **Request Processing (`_jspService()`):** Called per request; contains JSP logic.
7. **Destruction (`jspDestroy()`):** Called once on shutdown for cleanup.


Client sends Request for myPage.jsp
Is myPage.jsp new or modified?
Yes
No
Translation: myPage.jsp TO myPage_jsp.java
Compilation: myPage_jsp.java TO myPage_jsp.class
Loading: Load myPage_jsp.class
Instantiation: Create Servlet Instance (once)
Initialization: Call jspInit()
Web App/Container Shutdown
Destruction: Call jspDestroy() (once)
Servlet Instance Destroyed


Initialization: Call jspInit() (once)
Servlet Instance Ready
Request Processing: Call _jspService() for each request
Generate Response
Client Receives Response

Q7. Illustrate about any five implicit objects of JSP with example. (12.5)

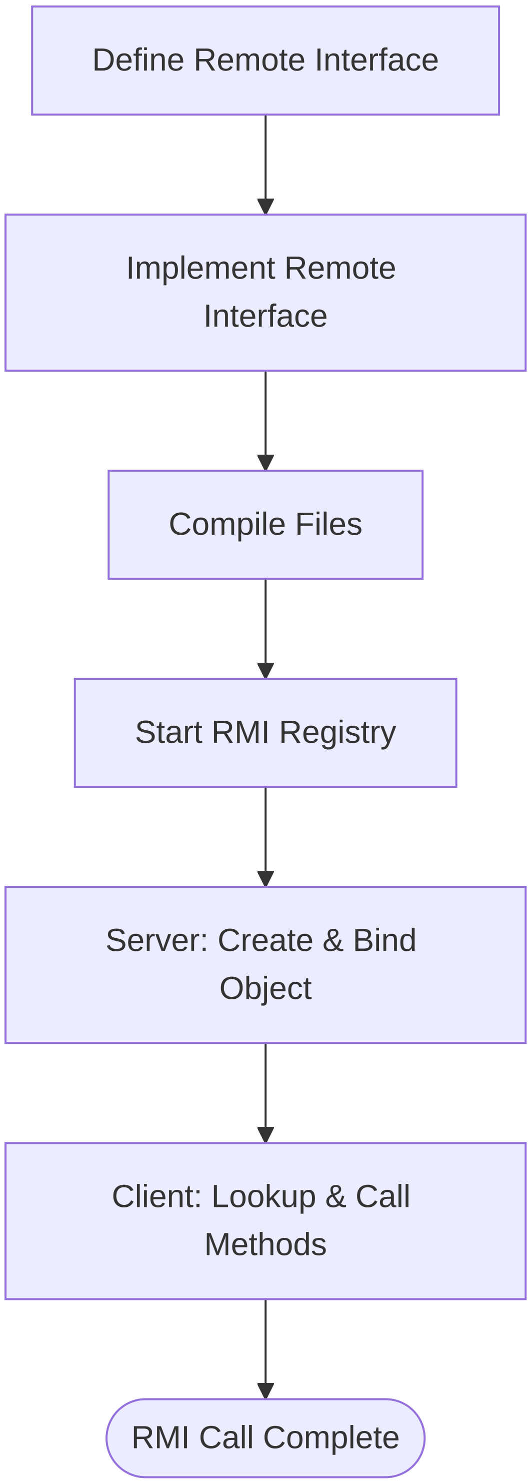
1. **`request`** (`HttpServletRequest`): Client's HTTP request data.
   `<p>Param: <%= request.getParameter("id") %></p>`
2. **`response`** (`HttpServletResponse`): Server's HTTP response.
   `<% response.setHeader("Cache-Control", "no-cache"); %>`
3. **`out`** (`JspWriter`): Writes to response stream.
   `<% out.print("Current time: " + new java.util.Date()); %>`
4. **`session`** (`HttpSession`): User-specific data across requests.
   `<% session.setAttribute("user", "john_doe"); %>`
5. **`application`** (`ServletContext`): Data shared by all users/app components.
   `<p>App Name: <%= application.getServletContextName() %></p>`

UNIT-IV

Q8. Discuss the steps to write a RMI program with an example of each step. (12.5)

1. **Define Remote Interface:** `public interface MyRemote extends java.rmi.Remote { String doWork() throws java.rmi.RemoteException; }`

2. **Implement Remote Interface:** `public class MyRemoteImpl extends java.rmi.server.UnicastRemoteObject implements MyRemote { public MyRemoteImpl() throws java.rmi.RemoteException {} public String doWork() { return "Work done"; } }`
3. **Compile:** `javac MyRemote.java MyRemoteImpl.java` (and `rmic` historically).
4. **Start RMI Registry:** `rmiregistry` or `java.rmi.registry.LocateRegistry.createRegistry(1099);`.
5. **Server: Create & Bind Object:** `MyRemoteImpl obj = new MyRemoteImpl(); java.rmi.Naming.rebind("MyService", obj);`
6. **Client: Lookup & Call:** `MyRemote stub = (MyRemote)java.rmi.Naming.lookup("rmi://host/MyService"); stub.doWork();`


Define Remote Interface
Implement Remote Interface
Compile Files
Start RMI Registry
Server: Create & Bind Object
Client: Lookup & Call Methods
RMI Call Complete

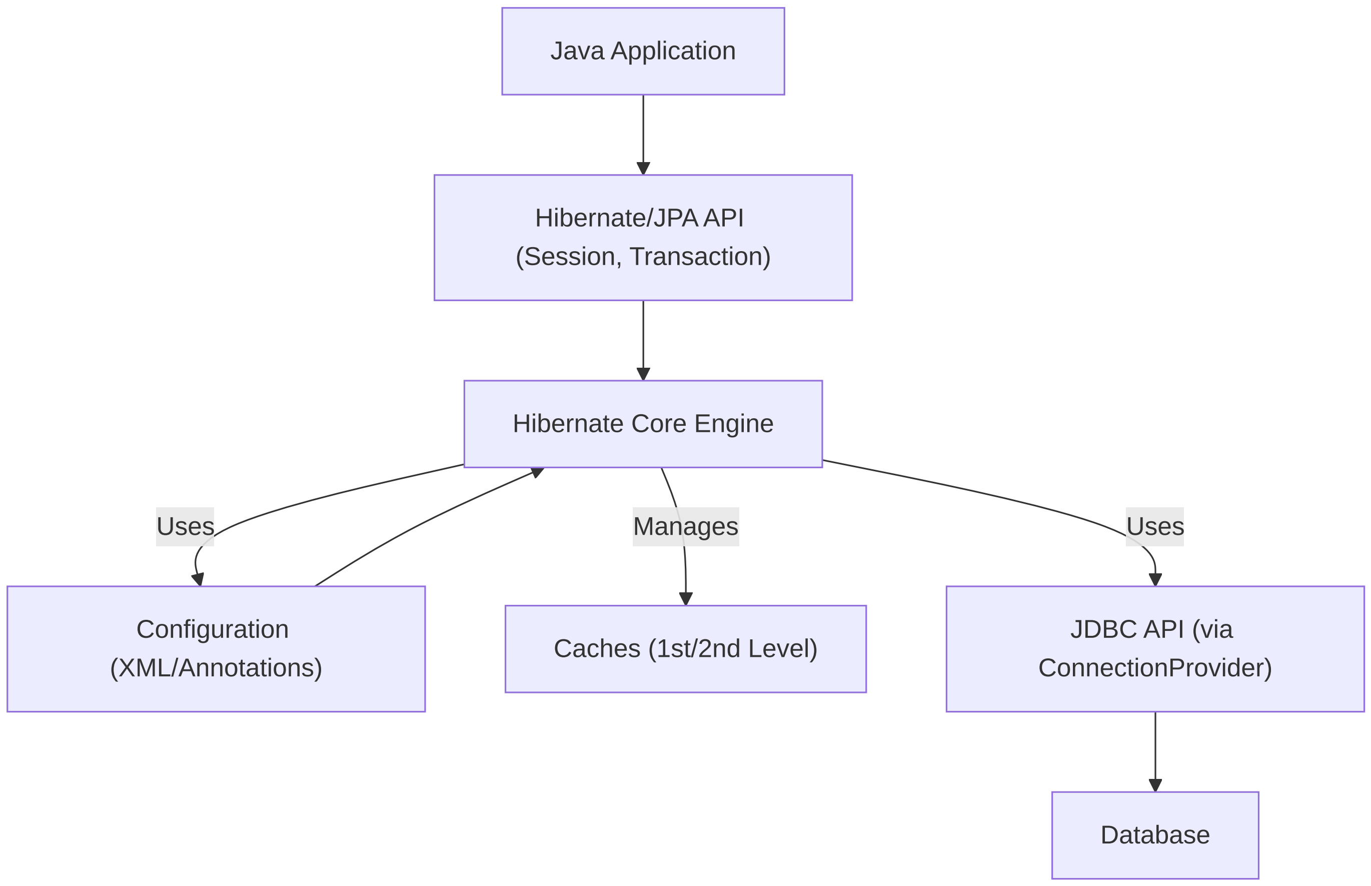
Q9. Draw the architecture diagram of Hibernate framework and also explain its elements. (12.5)

Elements:

1. **Configuration (`Configuration` / `persistence.xml`):** DB settings, mappings.
2. **SessionFactory (`SessionFactory` / `EntityManagerFactory`):** Factory for Sessions; created once.
3. **Session (`Session` / `EntityManager`):** Unit of work for DB interaction; short-lived, not thread-safe.
4. **Transaction (`Transaction` / `EntityTransaction`):** Atomic unit of work from Session.
5. **Persistent Objects (Entities):** Mapped Java POJOs.
6. **ConnectionProvider:** JDBC connection factory/pool.

7. **Query Objects:** For HQL/Criteria/SQL.

Layers: App -> Hibernate APIs -> Hibernate Core -> Backend APIs (JDBC) -> Database.


Java Application
Hibernate/JPA API (Session, Transaction)
Hibernate Core Engine
Uses
Manages
Uses
Configuration (XML/Annotations)
Caches (1st/2nd Level)
JDBC API (via ConnectionProvider)
Database