

CSC584 Enterprise Programming

CHAPTER 7 – DEVELOPMENT OF ENTERPRISE APPLICATION

Chapter 7

Outline

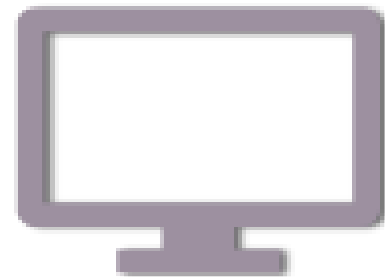
Development of Enterprise Application

- Choose the Java EE Architecture and Java EE pattern
- Design the Web components - HTML and JSP
- Develop Java Beans and Servlets
- Construct the JDBC connectivity with the enterprise application

Recap..



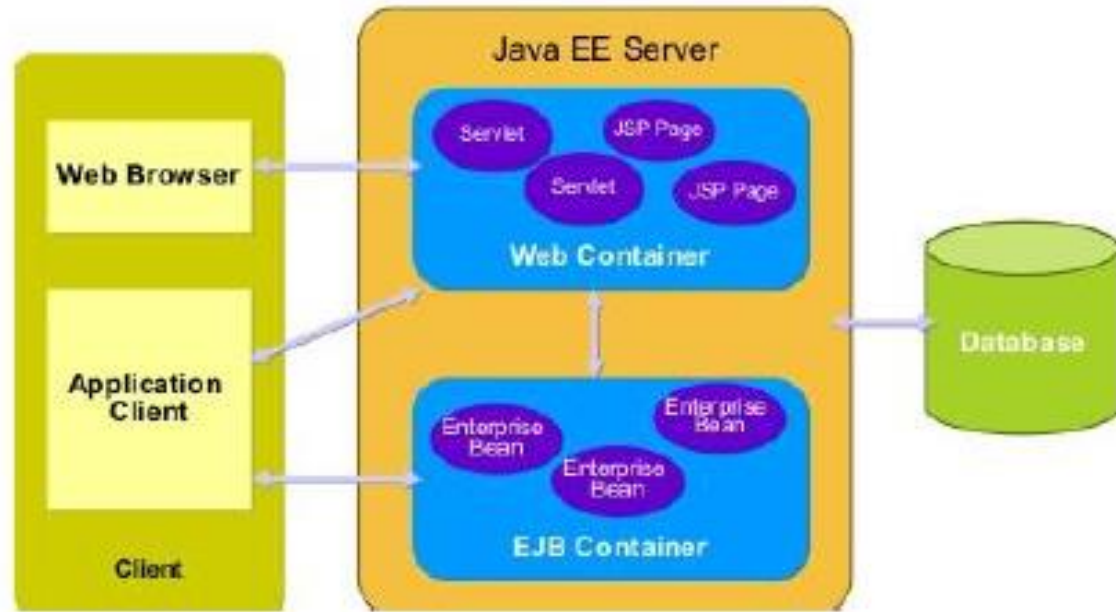
- Java EE architecture ?



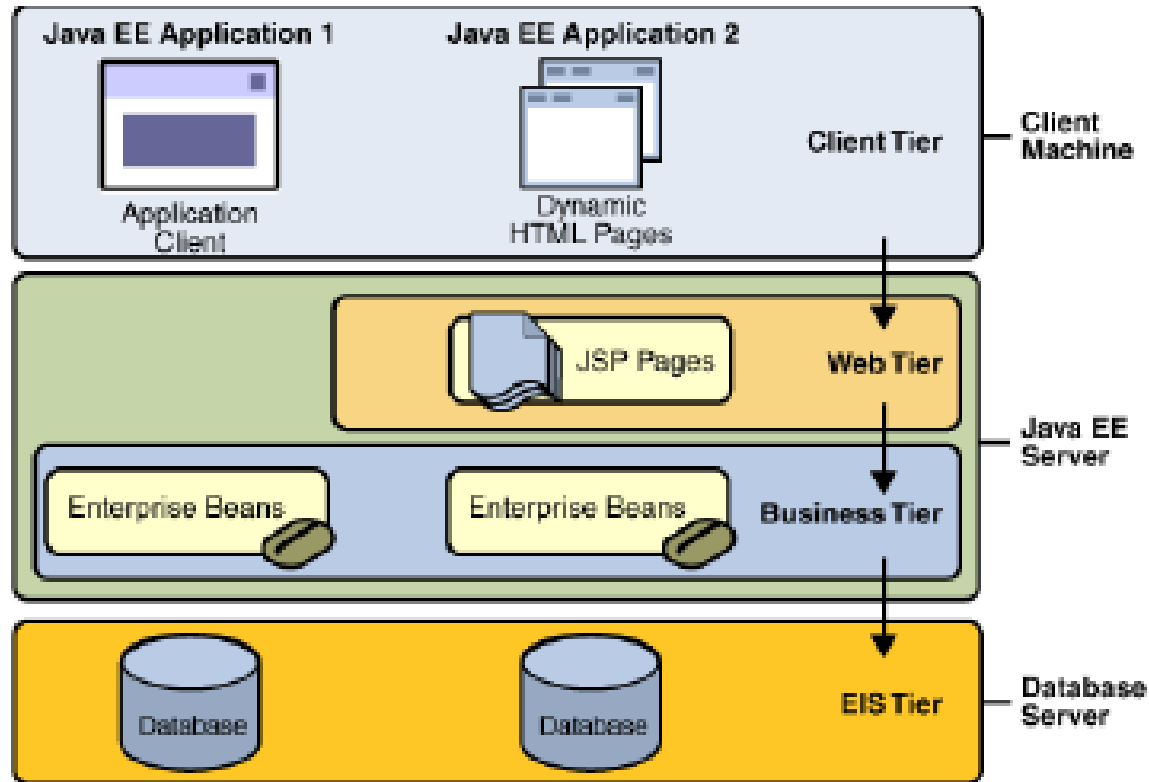
Web components?

Java EE Architecture - General

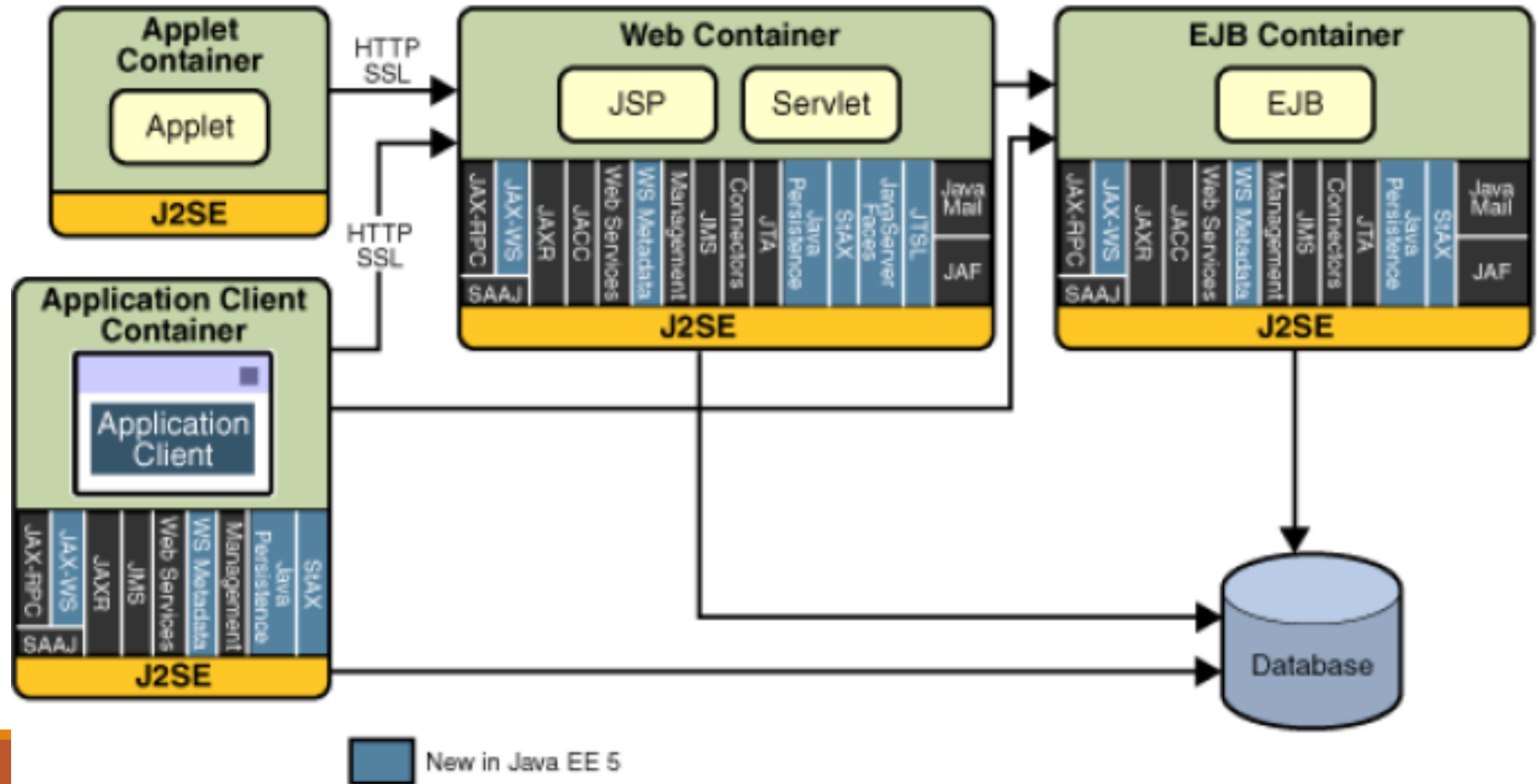
- Presentation Tier
- Business Logic Tier
- Data (EIS) Tier



Java EE Distributed Multi-tiered Application



Java EE Architecture



Java EE Server and Containers



Java EE server: The runtime portion of a Java EE product. A Java EE server provides EJB and web containers.



Enterprise JavaBeans (EJB) container: Manages the execution of enterprise beans for Java EE applications.

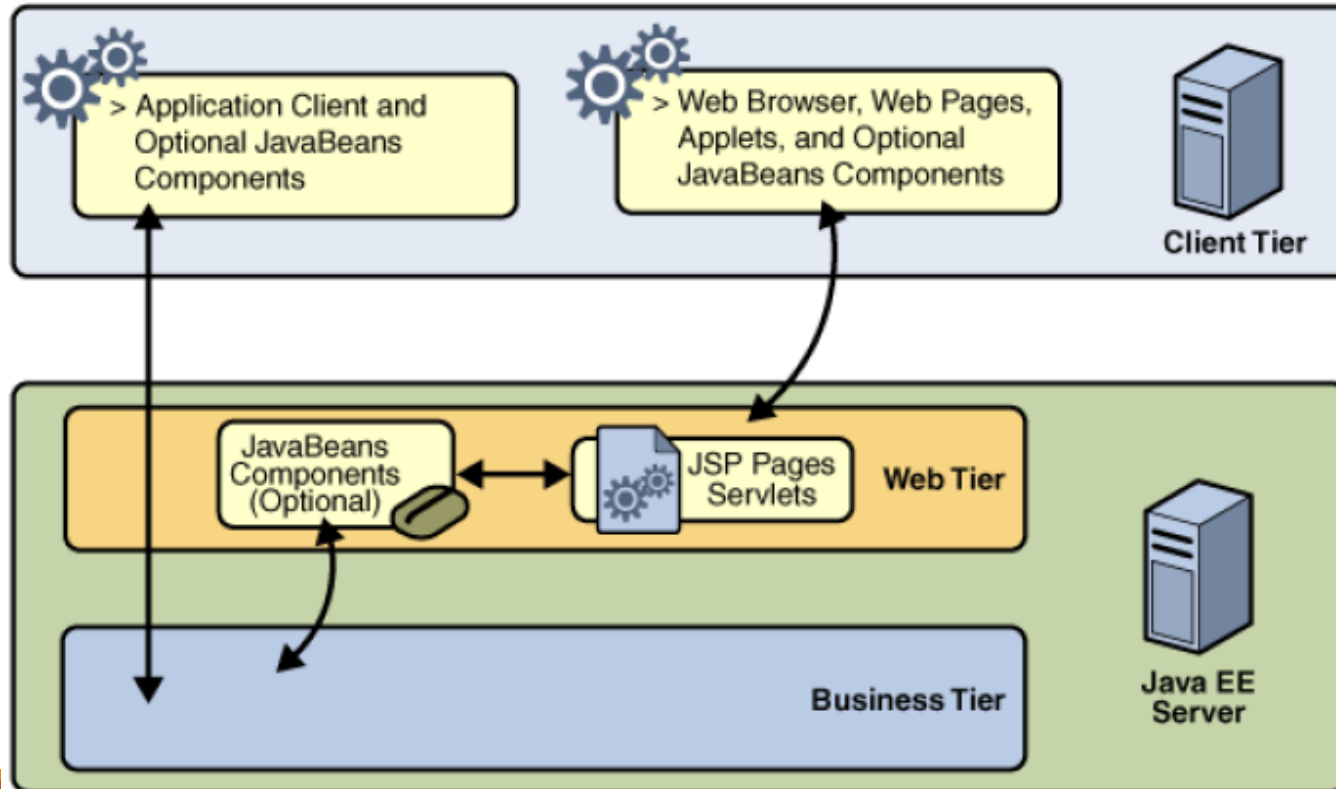


Web container: Manages the execution of JSP, Servlet, and Java Server Faces.

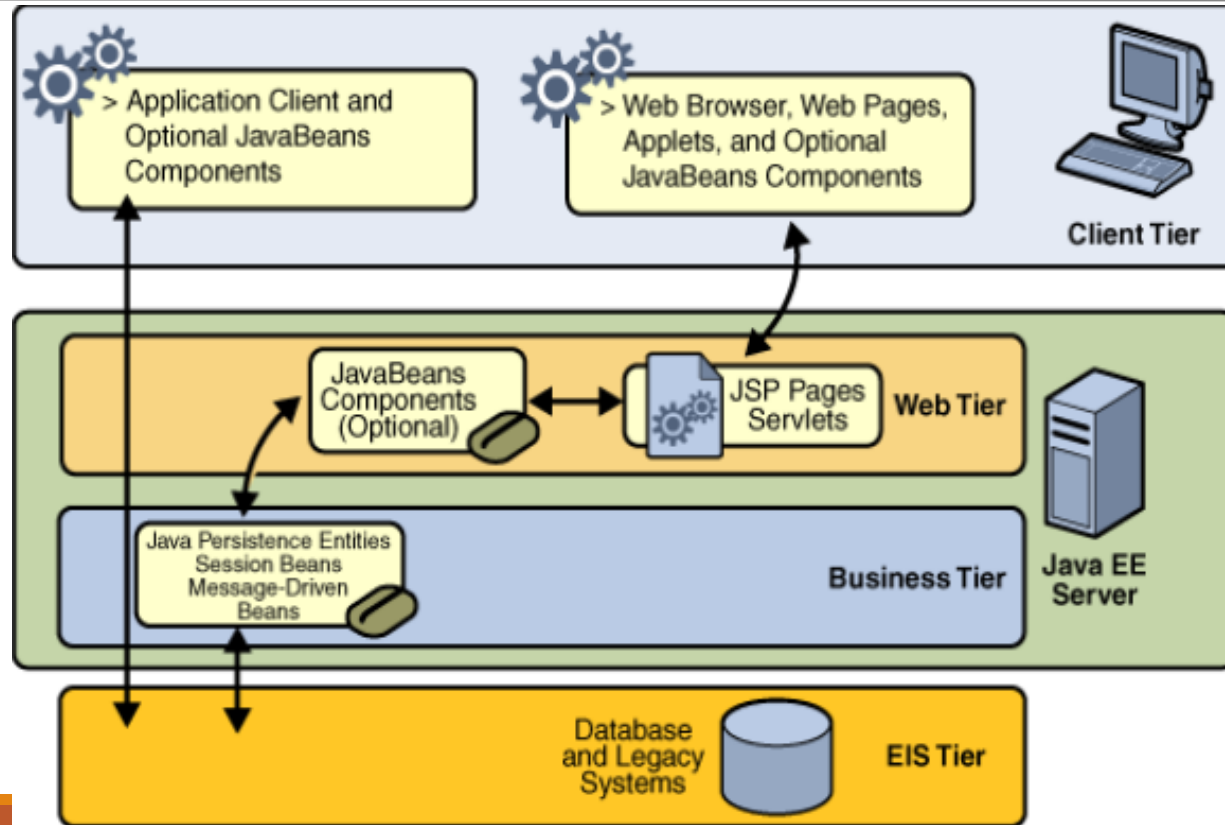


Application client container: Manages the execution of application client components.

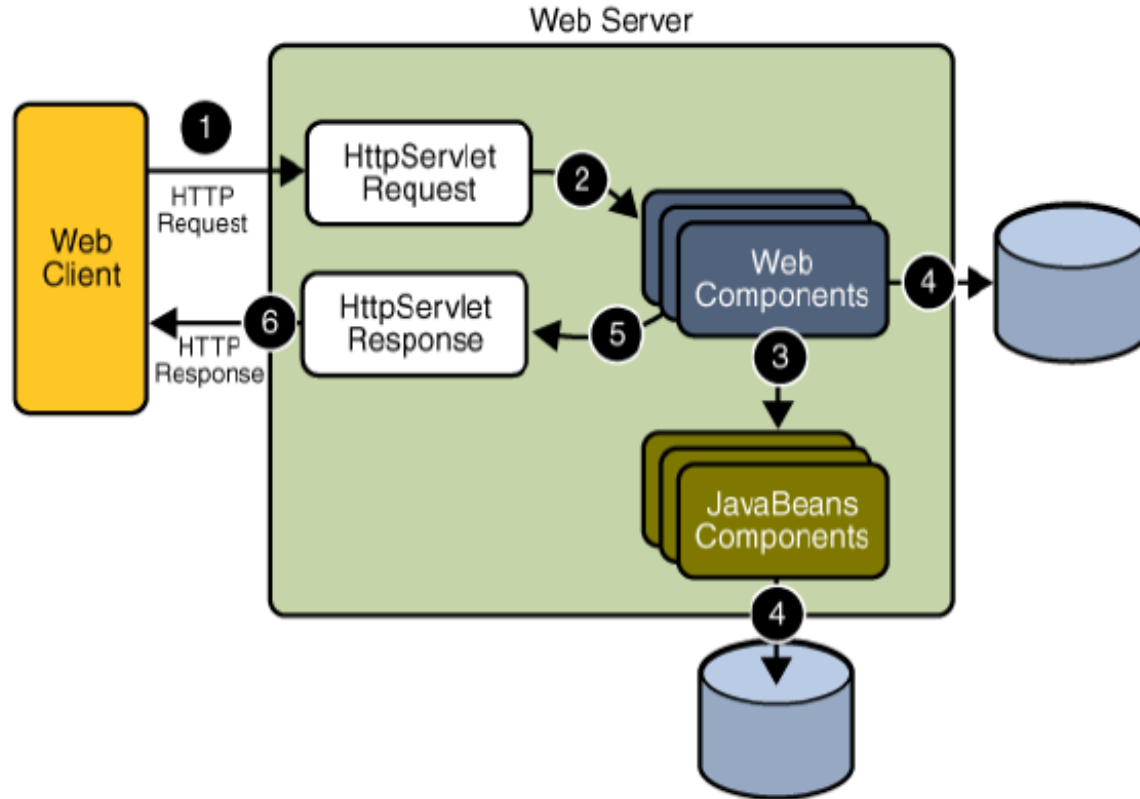
Web-Tier overview



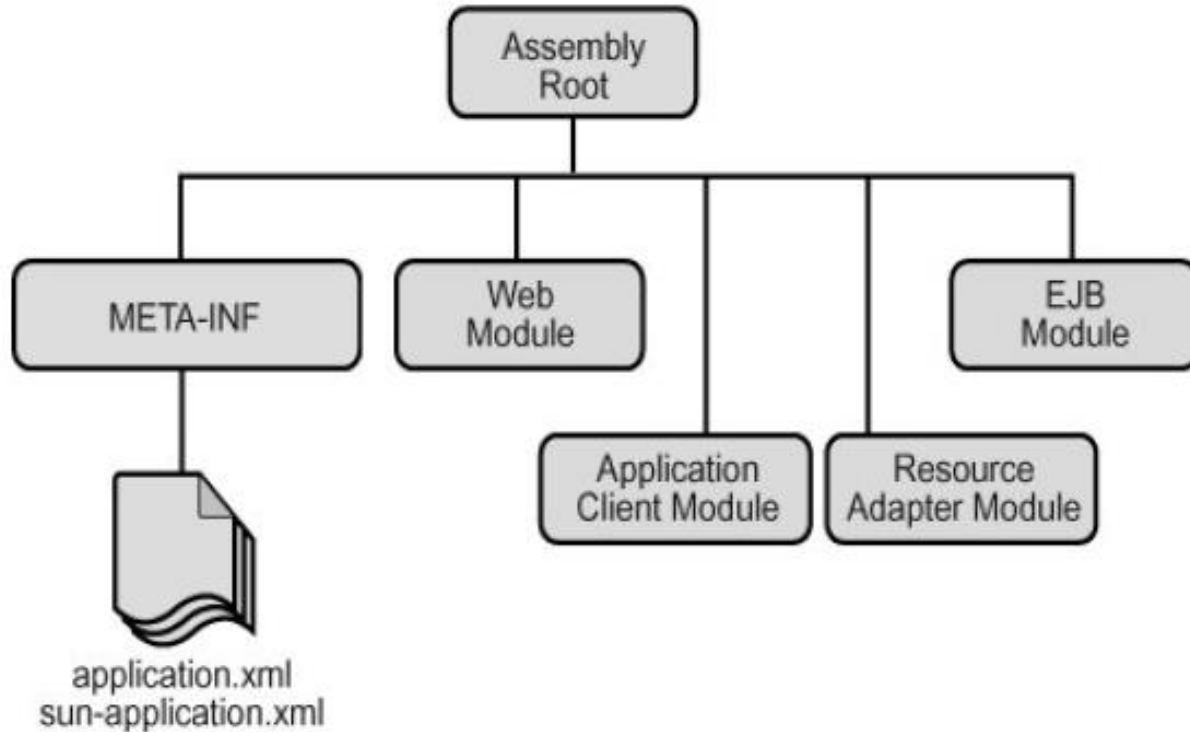
Business-Tier overview



Java Web Application Request Handling



Packaging applications



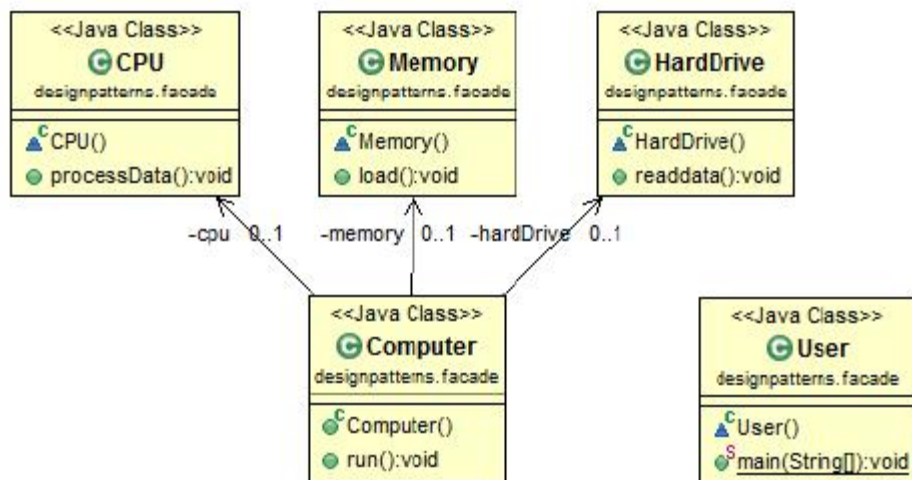
Packaging applications

- **A Java EE module**

- One or more Java EE components for the same container type
- One component deployment descriptor of that type

- **Java EE modules**

- **EJB modules**, which contain class files for enterprise beans and an EJB deployment descriptor. EJB modules are packaged as JAR files with a .jar extension.
- **Web modules**, which contain servlet class files, JSP files, supporting class files, GIF and HTML files, and a web application deployment descriptor. Web modules are packaged as JAR files with a .war (Web ARchive) extension.
- **Application client modules**, which contain class files and an application client deployment descriptor. Application client modules are packaged as JAR files with a .jar extension.



WHAT IS DESIGN PATTERN?

Template to solve
a problem

Obtained by trial
and error

Solution to
general problems

Best Practices

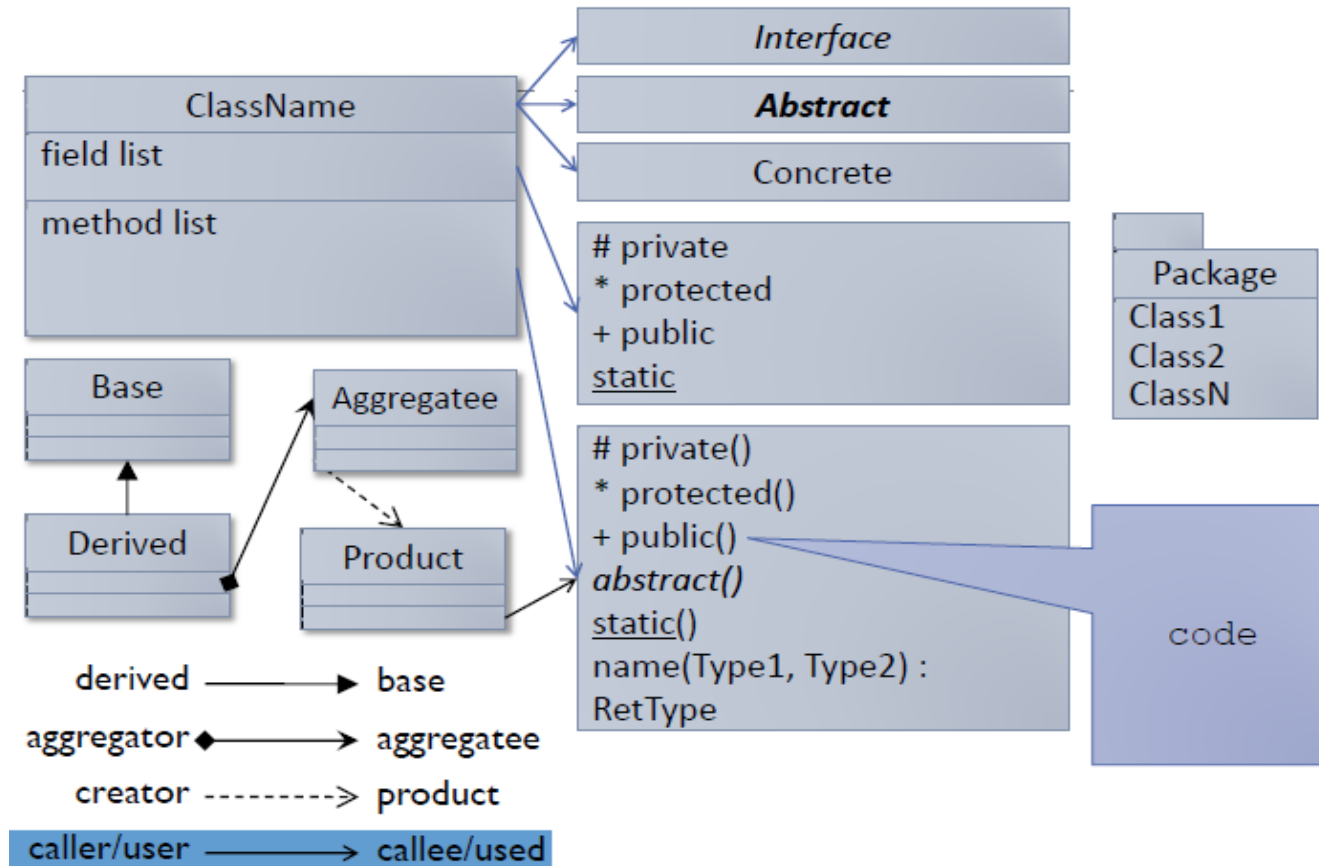
Common Language
among
Developers/Architects

Development process is fast
because of tested & proven
development paradigms

Design pattern

- A **design pattern** is a **general reusable** solution to a commonly occurring problem in software design. A design pattern is **not a finished design** that can be transformed directly into code.
- It is a **description** or **template** for how to solve a problem that can be used in many different situations.
- Object-oriented design patterns typically show **relationships** and **interactions** between classes or objects, without specifying the final application classes or objects that are involved.

UML class diagram recall



Simple Design Patterns

Singleton

- Ensure a class has only one instance, and provide a global point of access to it.

```
public class Singleton {  
    private Singleton() {}  
    private static Singleton _instance = null;  
    public static Singleton getInstance () {  
        if (_instance == null) _instance = new Singleton();  
        return _instance;  
    }  
    ...  
}
```

Lazy instantiation

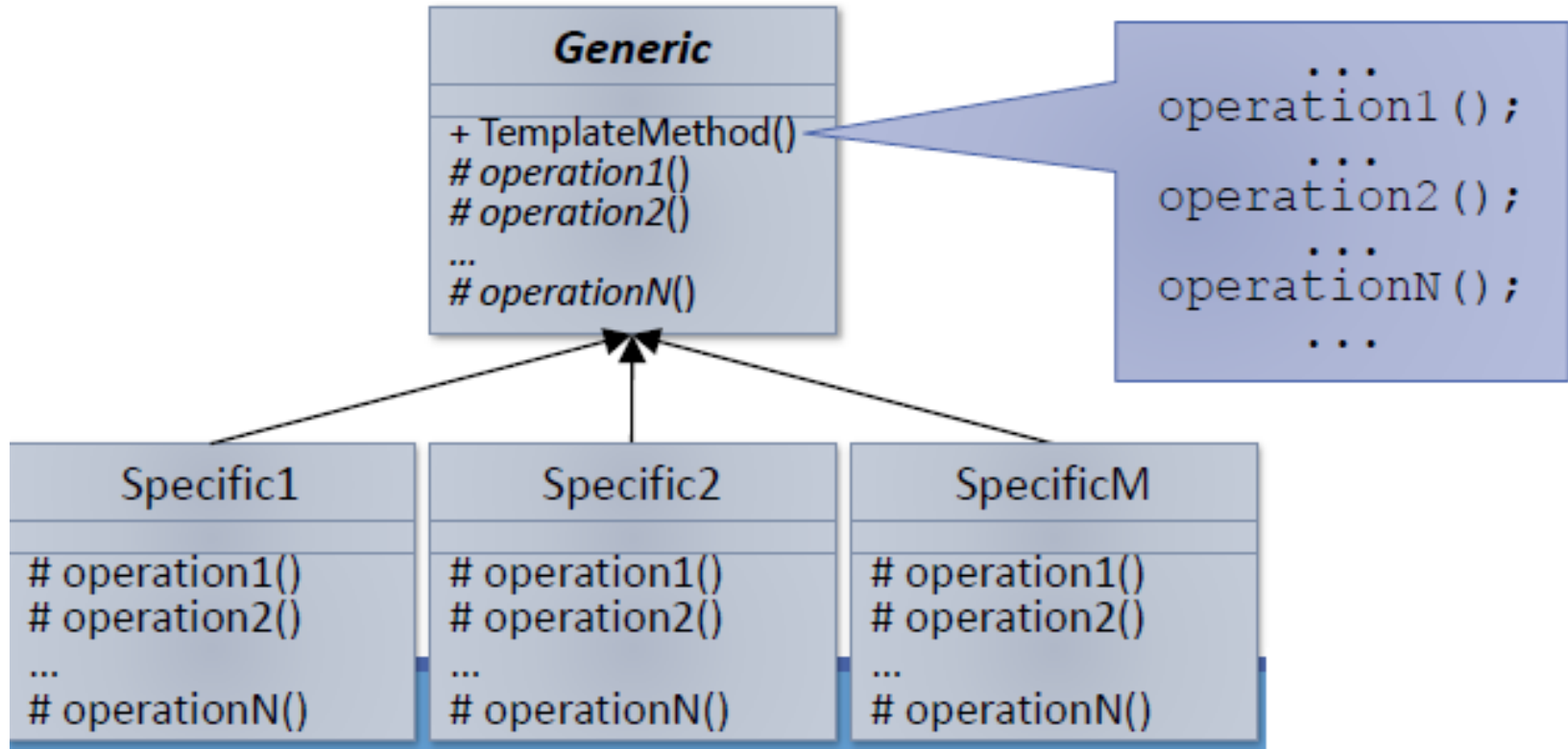
Tactic of delaying the creation of an object, the calculation of a value, or some other expensive process until the first time it is needed.

Singleton
_instance : Singleton
Singleton()
+ <u>getInstance()</u> : Singleton

return _instance;

Template Method

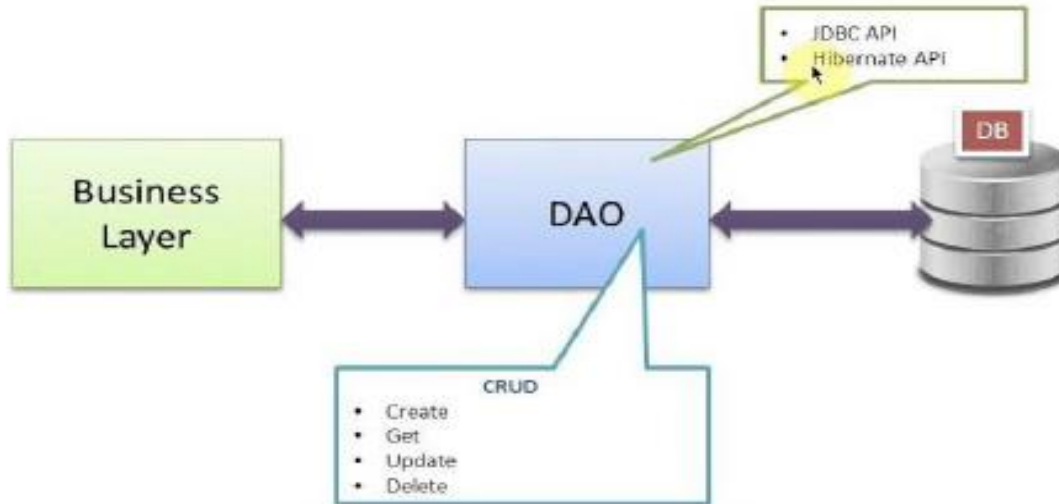
- Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.



Java EE pattern examples

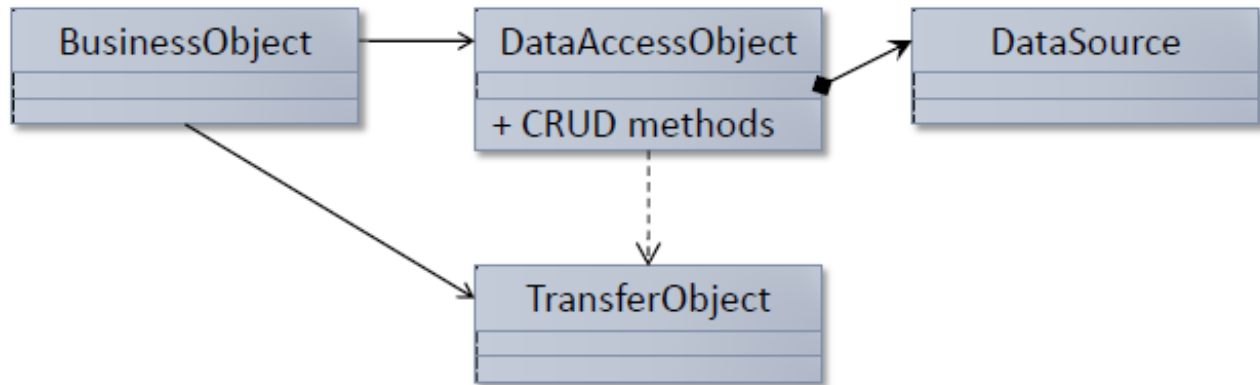
Data Access Objects (DAO)

- Code that depends on **specific features of data resources** ties together **business logic with data access logic**. This makes it difficult to replace or modify an application's data resources.
- This pattern
 - separates a data resource's client interface from its data access mechanisms;
 - adapts a specific data resource's access API to a generic client interface;
 - allows data access mechanisms to change independently of the code that uses the data.



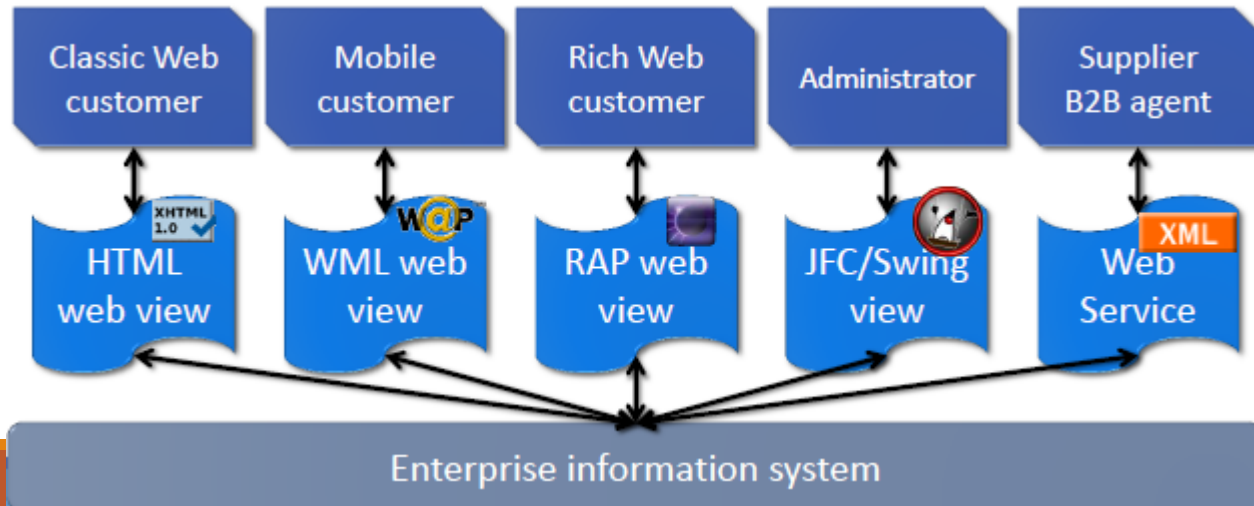
Data Access Objects (DAO) / 2

- **Business Object:** represents the data client.
- **Data Access Object:** abstracts the underlying data access implementation for the Business Object to enable transparent access to the data source.
- **Data Source:** represents a data source implementation.
- **Transfer Object:** the data carrier, DAO may use it to return data to the client.

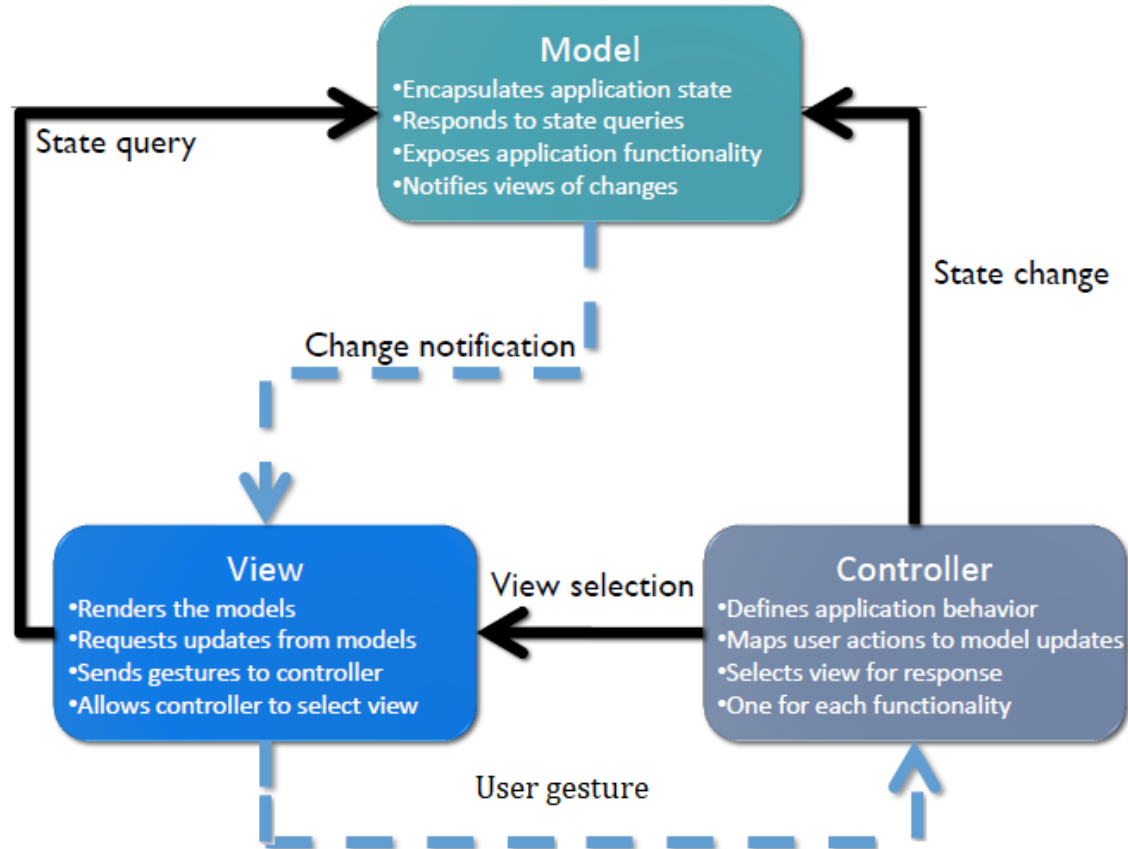


Model-View-Controller (MVC)

- Application presents content to users in numerous ways containing various data. The engineering team responsible for designing, implementing, and maintaining the application is composed of individuals with different skill sets.



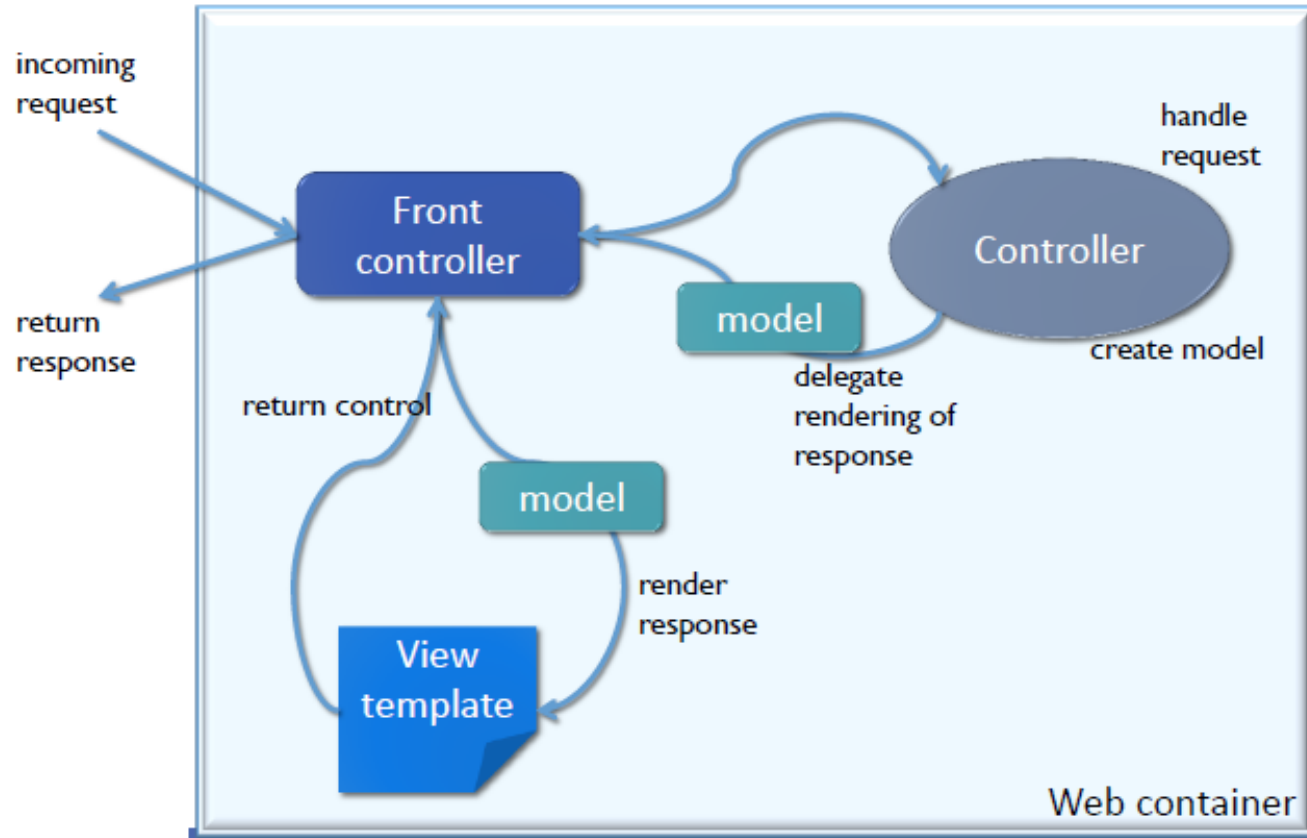
Model-View-Controller (MVC) / 2



Front Controller

- The presentation-tier request handling mechanism must control and coordinate processing of each user across multiple requests. Such control mechanisms may be managed in either a centralized or decentralized manner.
- Problems:
 - Each view is required to provide its own system services, often resulting in duplicate code.
 - View navigation is left to the views. This may result in commingled view content and view navigation.
 - Distributed control is more difficult to maintain: changes will need to be made in numerous places.

Front Controller / 2

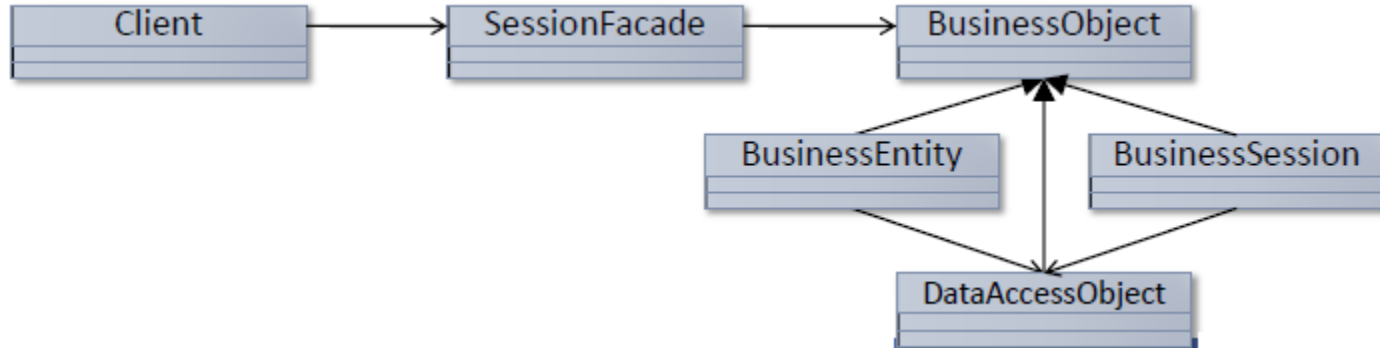


Session Facade

- Enterprise beans encapsulate business logic and business data and expose their interfaces, and thus the complexity of the distributed services, to the client tier.
 - Tight coupling, which leads to direct dependence between clients and business objects;
 - Too many method invocations between client and server, leading to network performance problems;
 - Lack of a uniform client access strategy, exposing business objects to misuse.

Session Facade / 2

- **Client:** the object which needs access to the business service. This client can be another session bean (Session Facade) in the same business tier or a business delegate in another tier.
- **Session Facade:** a session bean which manages the relationships between numerous Business Objects and provides a higher level abstraction to the client.
- **Business Object:** a role object that facilitates applying different strategies, such as session beans, entity beans and a DAO. A Business Object provides data and/or some services.



Design the Web components - HTML and JSP

JavaServerPages are similar to HTML files, but provide the ability to display dynamic content with Web pages. A JavaServer Page (JSP) is an HTML web page that contains embedded Java code.

The HTML displays the static portion of the page (text and graphics).

The Java code is used to generate dynamic content for the page (JDBC, RMI).

sample.jsp

HTML

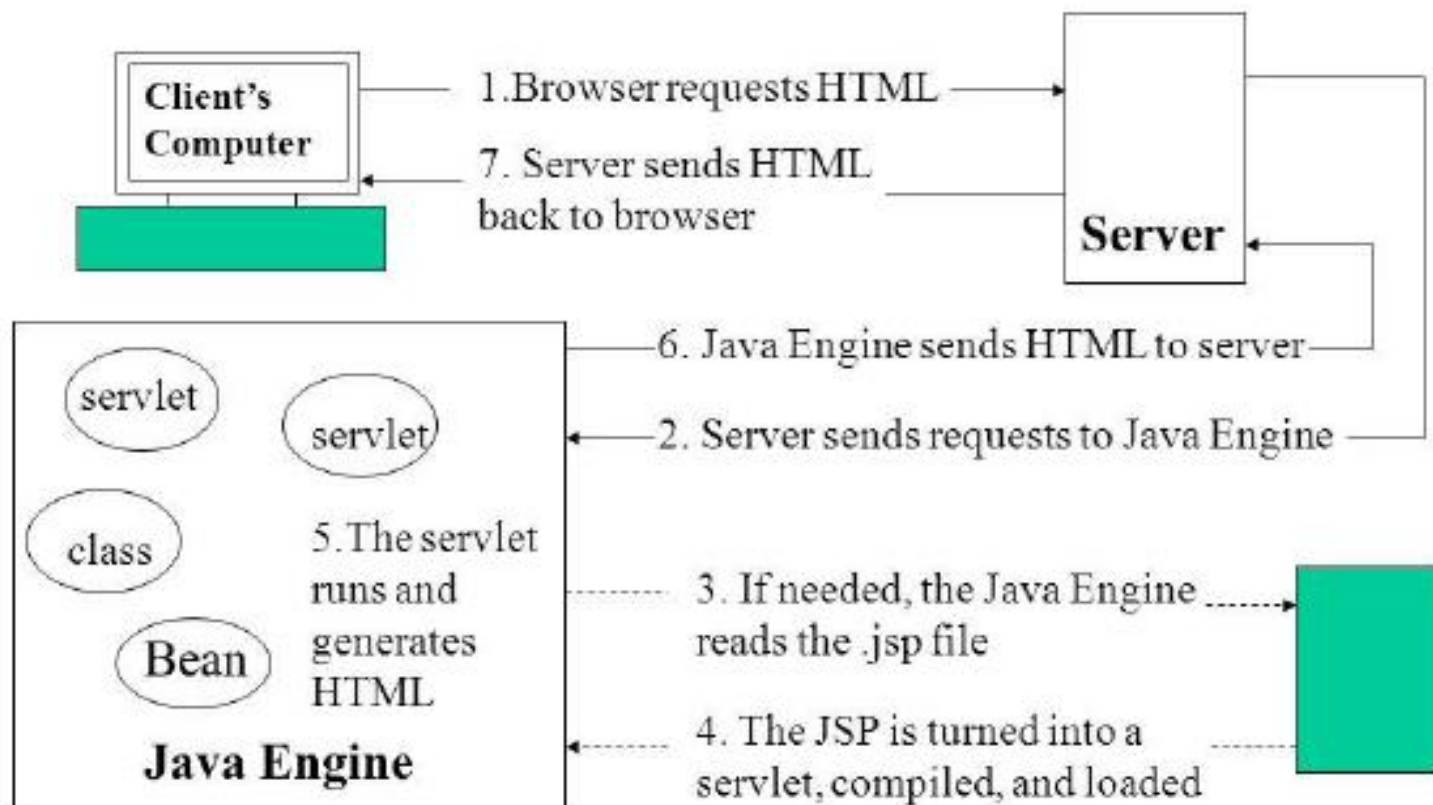
Java Code

HTML

HTML

A JSP separates user interfaces from content generation. This allows the separation of web page design from component development.

Java Server Pages (JSP)



HTML & JSP

```
<!-- ComputeLoan.jsp -->
<html>
<head>
  <title>ComputeLoan</title>
</head>
<body>
<% double loanAmount = Double.parseDouble(
    request.getParameter("loanAmount"));
double annualInterestRate = Double.parseDouble(
    request.getParameter("annualInterestRate"));
double numberOfYears = Integer.parseInt(
    request.getParameter("numberOfYears"));
double monthlyInterestRate = annualInterestRate / 1200;
double monthlyPayment = loanAmount * monthlyInterestRate /
    (1 - 1 / Math.pow(1 + monthlyInterestRate, numberOfYears * 12));
double totalPayment = monthlyPayment * numberOfYears * 12; %>
Loan Amount: <%= loanAmount %><br />
Annual Interest Rate: <%= annualInterestRate %><br />
Number of Years: <%= numberOfYears %><br />
<b>Monthly Payment: <%= monthlyPayment %><br />
Total Payment: <%= totalPayment %><br /></b>
</body>
</html>
```

Compute Loan Payment

Loan Amount

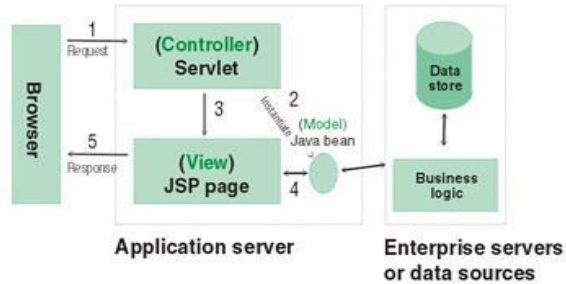
Annual Interest Rate

Number of Years

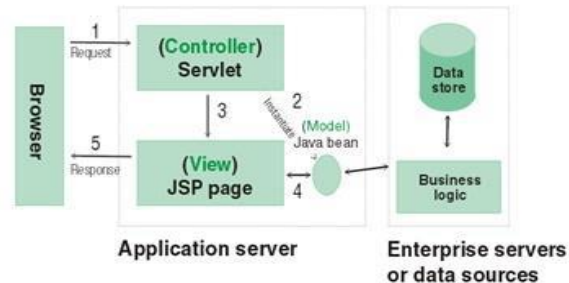
Compute Loan Payment

Reset

Develop Java Beans and Servlets



Java EE Web Development



Recap: Java Bean

- Java Beans are software component models.
- A java bean is a general purpose component model.
- A java bean is a reusable software component that can be visually manipulated in builder tools.
- Their primary goal of java bean is WORA (Write Once Run Anywhere)
- Java beans should adhere to portability, reusability and interoperability.

Java Bean: Example

```
public class Person {  
    private String name;  
    private String email;  
    private long phoneNo;  
  
    public Person() {  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setEmail(String email) {  
        this.email = email;  
    }  
  
    public String getEmail() {  
        return email;  
    }  
}
```

Recap: Servlet

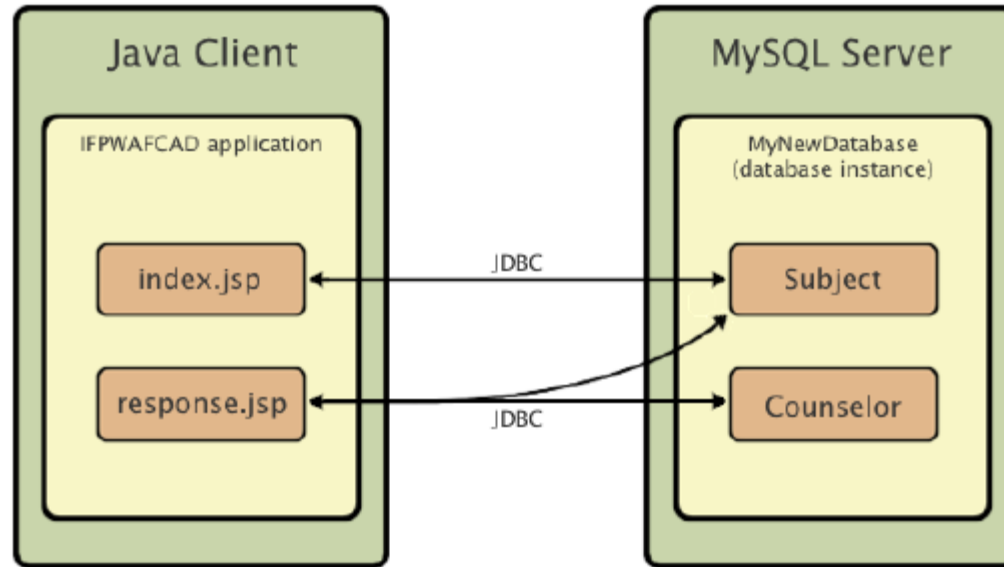
- A **servlet** is a Java class that can be loaded dynamically into and run by a special web server.
- This servlet-aware web server, is known as **servlet container**.
- Servlets interact with clients via a request-response model based on HTTP.
- Therefore, a servlet container must support HTTP as the protocol for client requests and server responses.
- A servlet container also can support similar protocols such as HTTPS (HTTP over SSL) for secure transactions.

Servlet: Example

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

public class BeanInServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        Person p = new Person();
        p.setName("Sam Dalton");
        p.setEmail("sam2222@gmail.com");
        p.setPhoneNo(1111111111);
        req.setAttribute("person", p);
        RequestDispatcher rd = req.getRequestDispatcher("/jsp/beandata.jsp");
        rd.forward(req, res);
    }
}
```

Construct the JDBC connectivity with the enterprise application



<https://netbeans.org/kb/docs/web/mysql-webapp.html>

MVC APPLICATION EXAMPLE

- In this example, servlet as a controller, JSP as a view component, java bean class as a model.
- Create 5 pages:
 - index.jsp: a page that gets input from the user.
 - ControllerServlet.java : a servlet that acts as a controller.
 - login-success.jsp and login-error.jsp files acts as view components.
 - web.xml file for mapping the servlet.

index.jsp

```
<form action="ControllerServlet" method="post">  
Name:<input type="text" name="name"><br>  
Password:<input type="password" name="password"><br>  
<input type="submit" value="login">  
</form>
```

```
package com.javatpoint;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class ControllerServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();

        String name=request.getParameter("name");
        String password=request.getParameter("password");

        LoginBean bean=new LoginBean();
        bean.setName(name);
        bean.setPassword(password);
        request.setAttribute("bean",bean);

        boolean status=bean.validate();
```

```
        if(status){
            RequestDispatcher rd=request.getRequestDispatcher("login-success.jsp");
            rd.forward(request, response);
        }
        else{
            RequestDispatcher rd=request.getRequestDispatcher("login-error.jsp");
            rd.forward(request, response);
        }
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        doPost(req, resp);
    }
}
```


loginBean.java

```
package com.javatpoint;
public class LoginBean {
    private String name,password;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public boolean validate(){
        if(password.equals("admin")){
            return true;
        }
        else{
            return false;
        }
    }
}
```

loginSuccess.jsp

```
<%@page import="com.javatpoint.LoginBean"%>

<p>You are successfully logged in!</p>
<%
LoginBean bean=(LoginBean)request.getAttribute("bean");
out.print("Welcome, "+bean.getName());
%>
```

loginError.jsp

```
<%@ include file="index.jsp" %>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-  
  app_2_5.xsd"  
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-  
  app_3_0.xsd"  
  id="WebApp_ID" version="3.0">
```

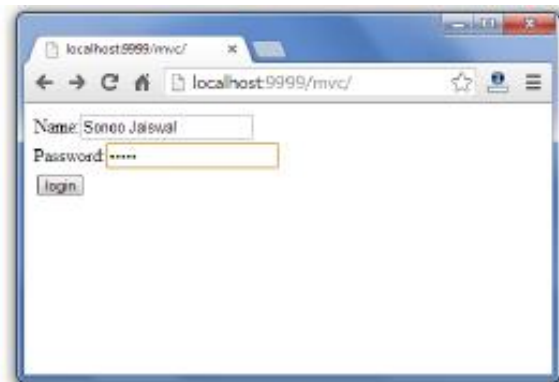
```
  <servlet>  
    <servlet-name>s1</servlet-name>  
    <servlet-class>com.javatpoint.ControllerServlet</servlet-class>  
  </servlet>  
  <servlet-mapping>  
    <servlet-name>s1</servlet-name>  
    <url-pattern>/ControllerServlet</url-pattern>  
  </servlet-mapping>  
</web-app>
```

Output



A web browser window showing the URL `localhost:9999/mvc/`. The page contains a login form with two input fields: "Name:" and "Password:". Below the "Password:" field is a "login" button.

index.jsp



A web browser window showing the URL `localhost:9999/mvc/`. The "Name:" field is filled with "Sonoo Jaiswal" and the "Password:" field is filled with ".....". The "login" button is highlighted with a yellow border.

index.jsp



A web browser window showing the URL `localhost:9999/mvc/ControllerSe`. The page displays the message "You are successfully logged in!" followed by "Welcome, Sonoo Jaiswal".

loginSuccess.jsp