CSC584 Enterprise Programming

Chapter 7 – **Development of Enterprise Application**

MARSHIMA MOHD ROSLI

DEPT COMPUTER SCIENCE

Chapter 6 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….
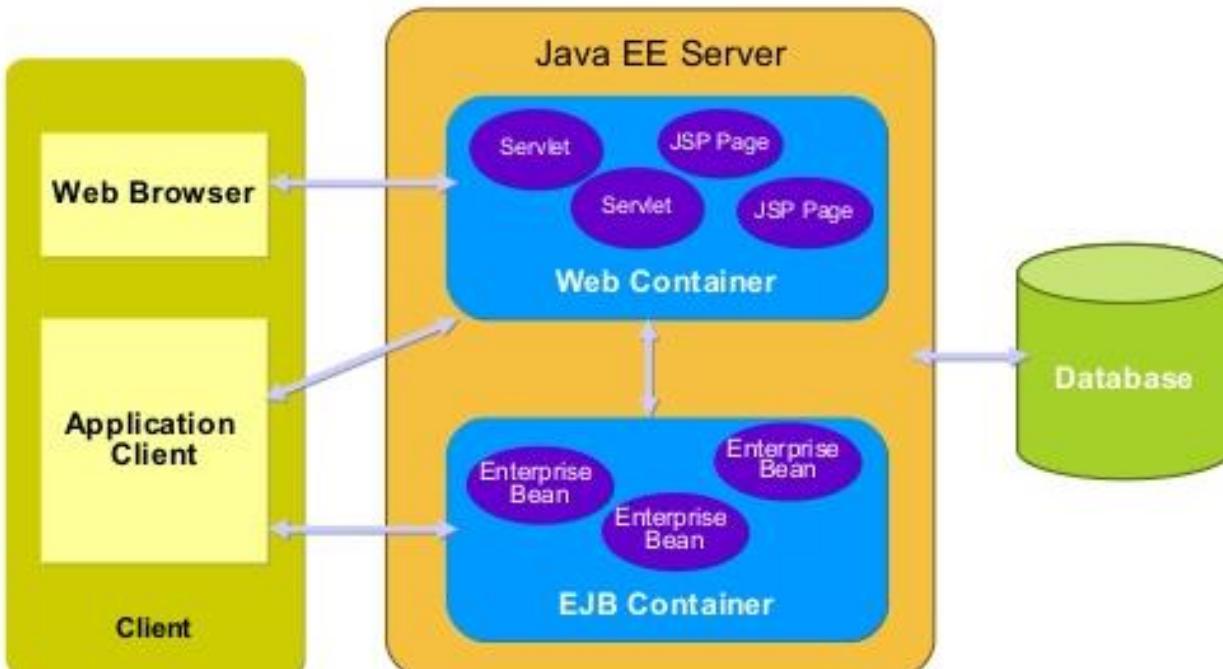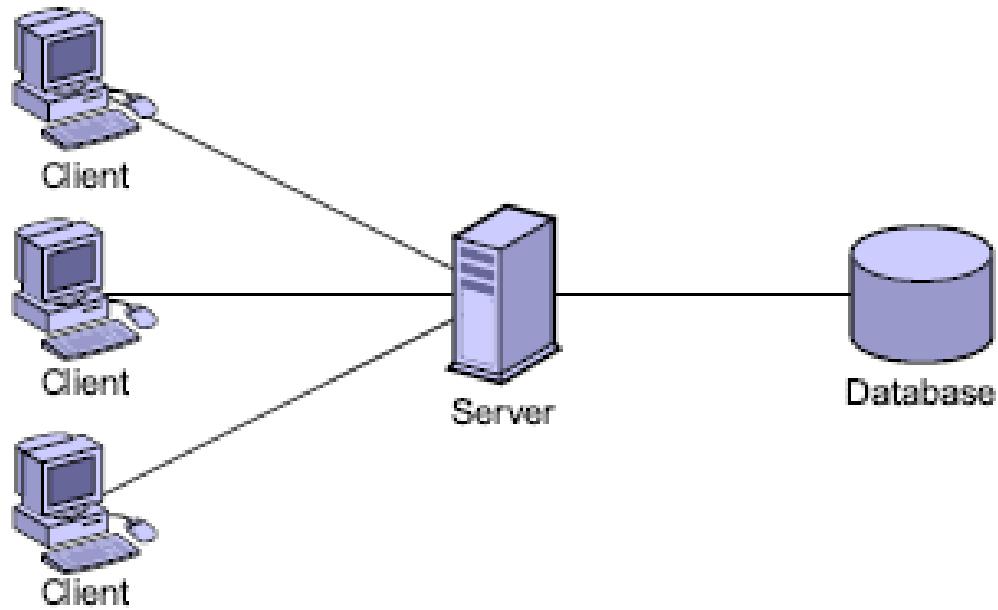
Java EE architecture ?　　Web components?
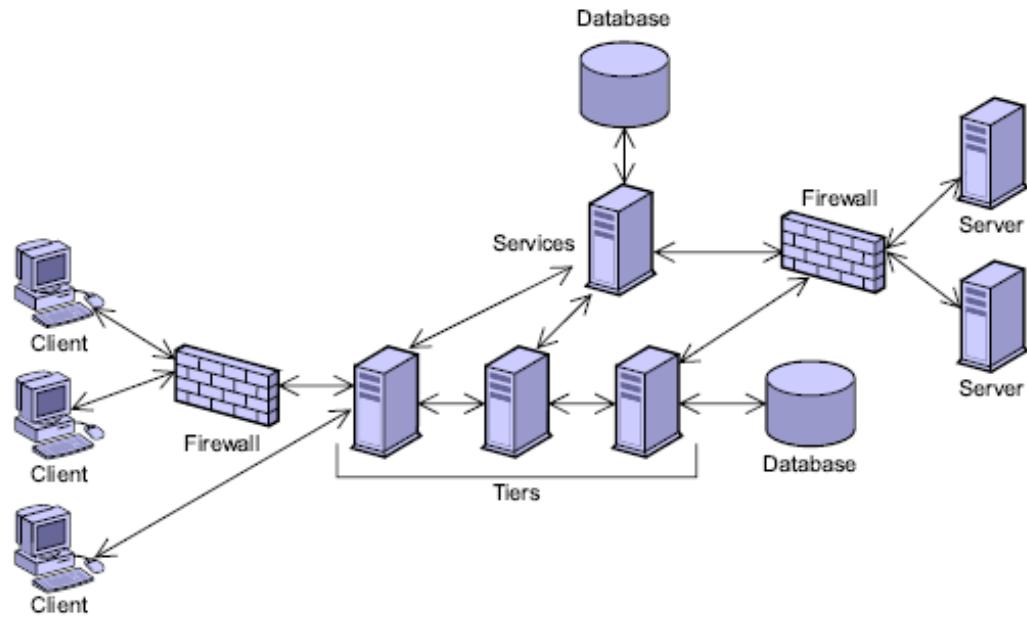
Java EE Architecture - General

- Presentation Tier
- Business Logic Tier
- Data (EIS) Tier
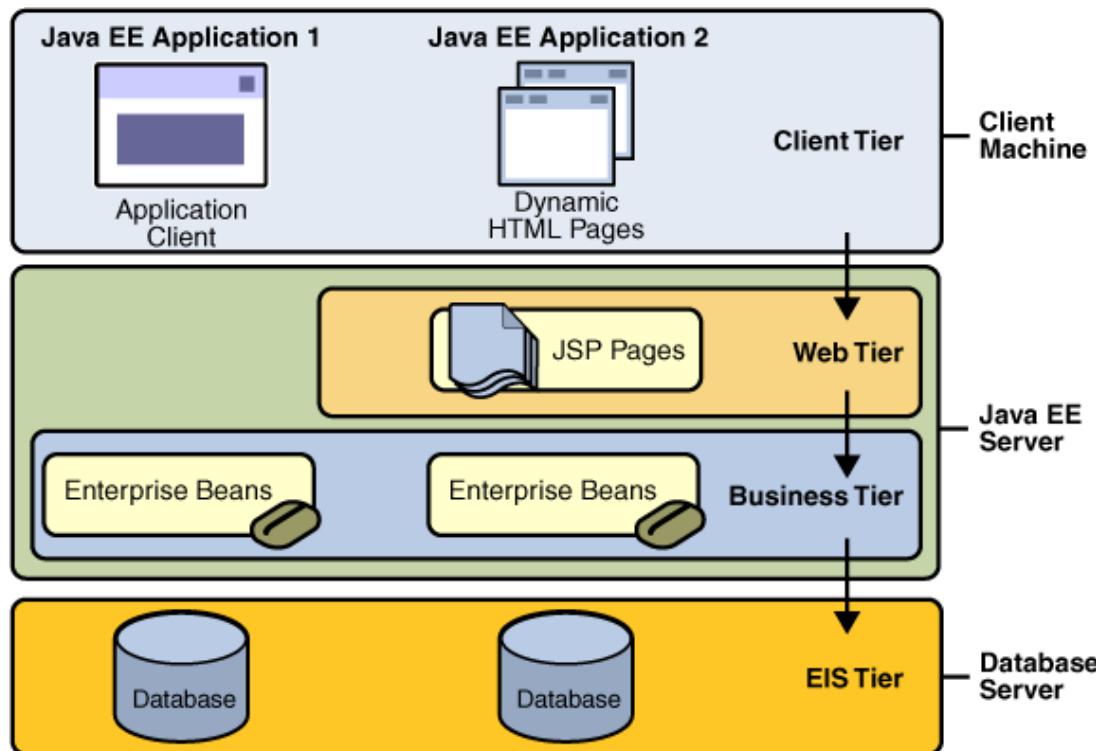
Before….
Minimum
Distributed
Systems
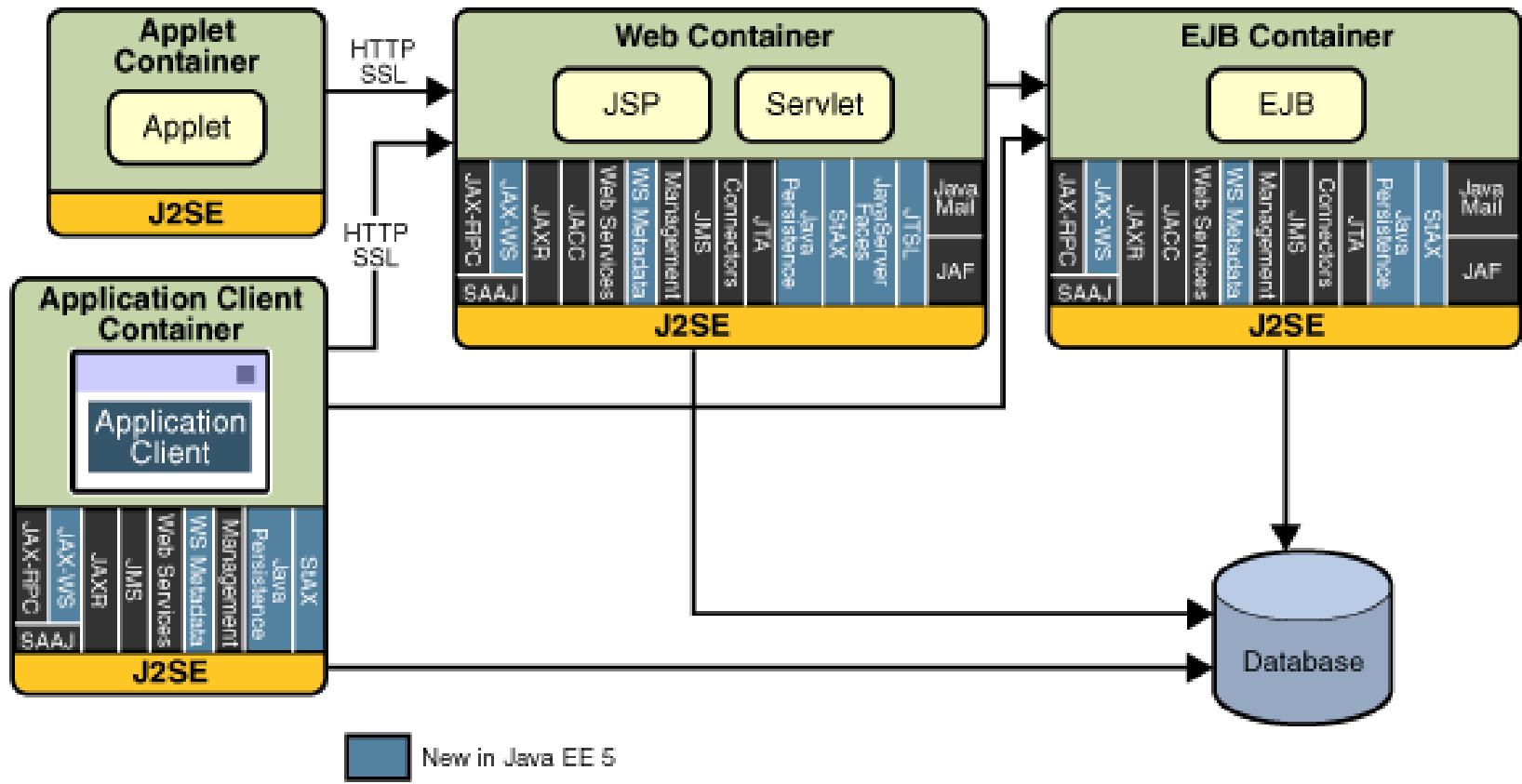
Next….
Highly
Distributed
Systems

Java EE Distributed Multitiered 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.
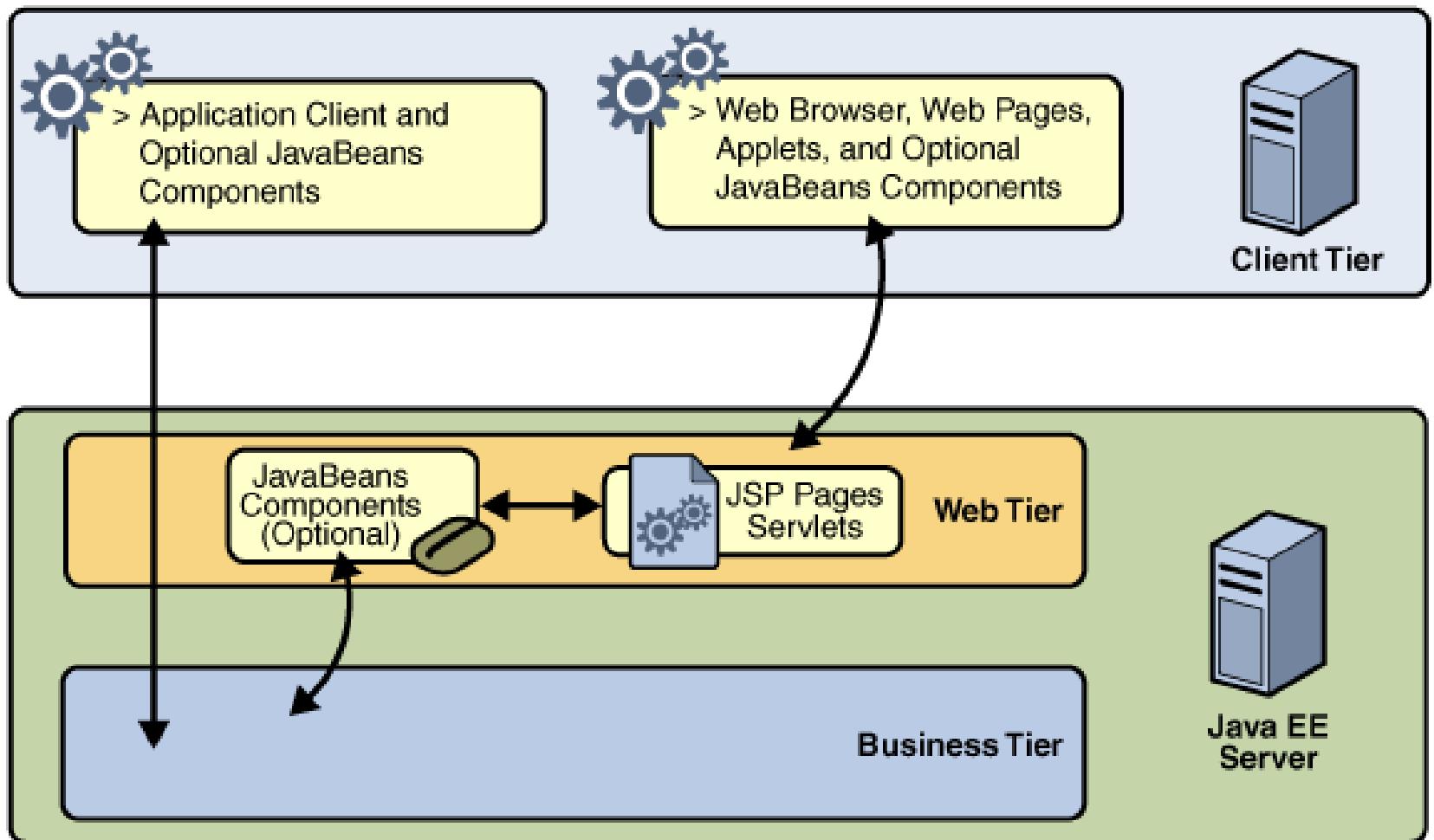
Web container: Manages the execution of JSP, Servlet, and Java Server Faces.
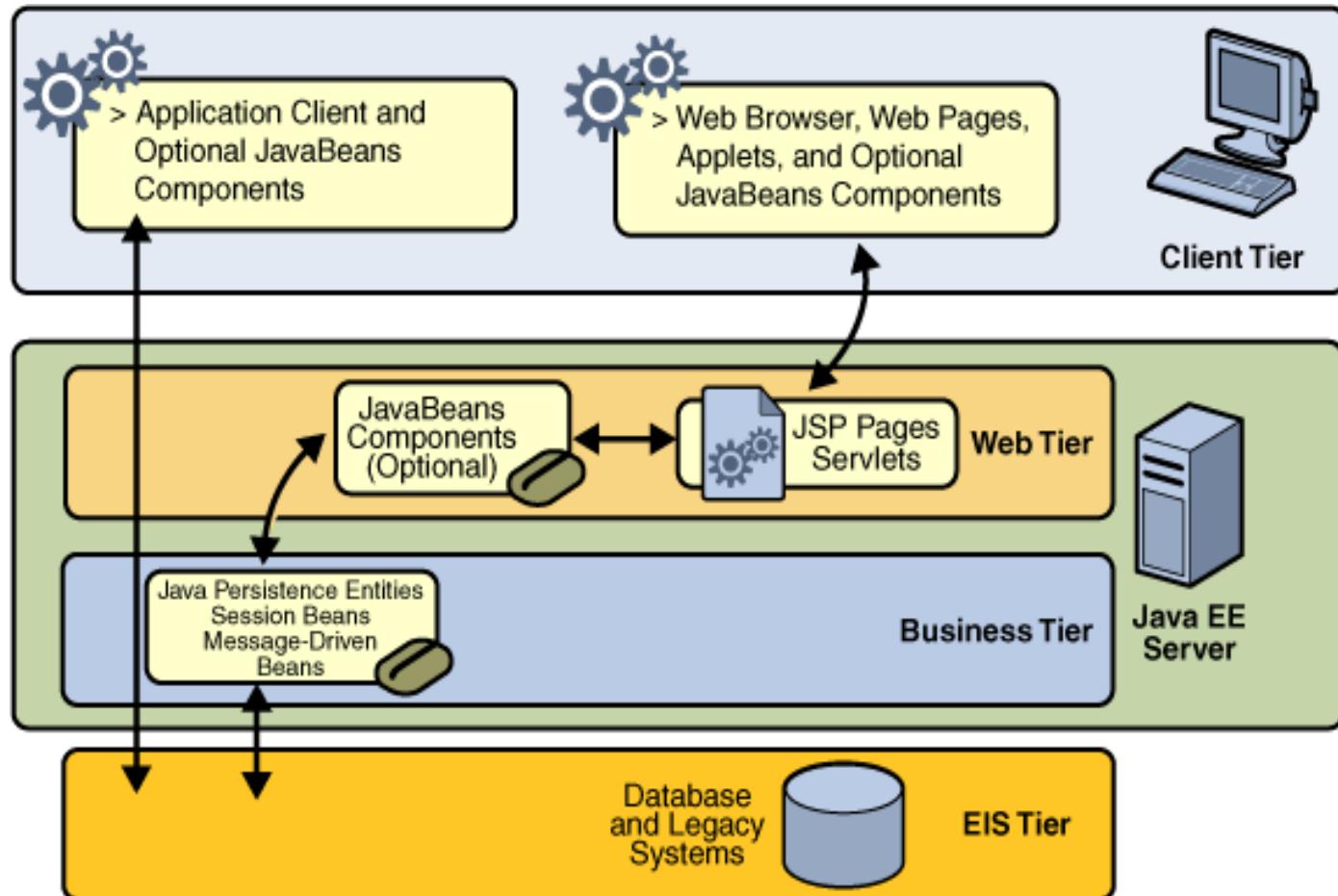
Application client container: Manages the execution of application client components.
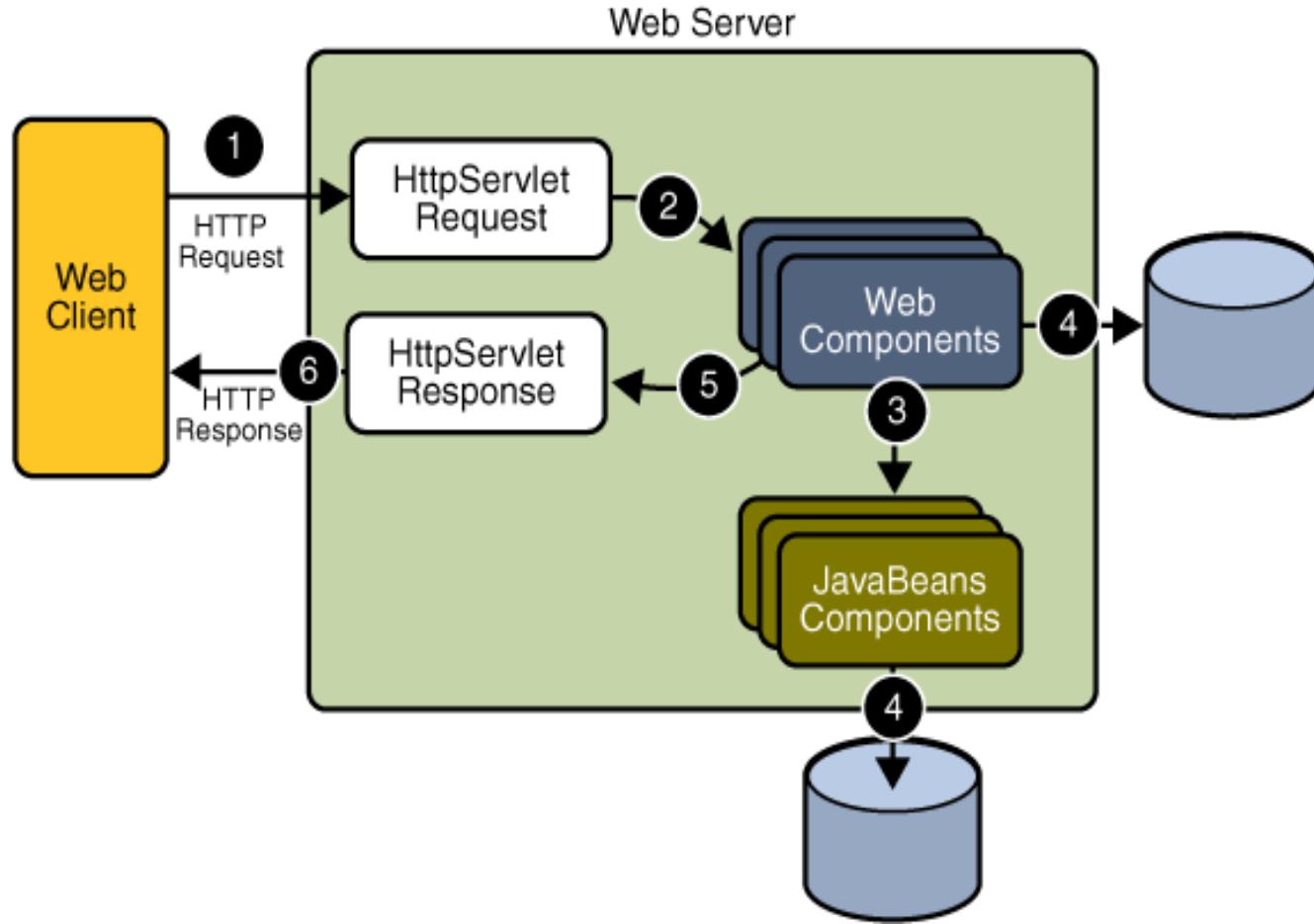
Web-Tier overview
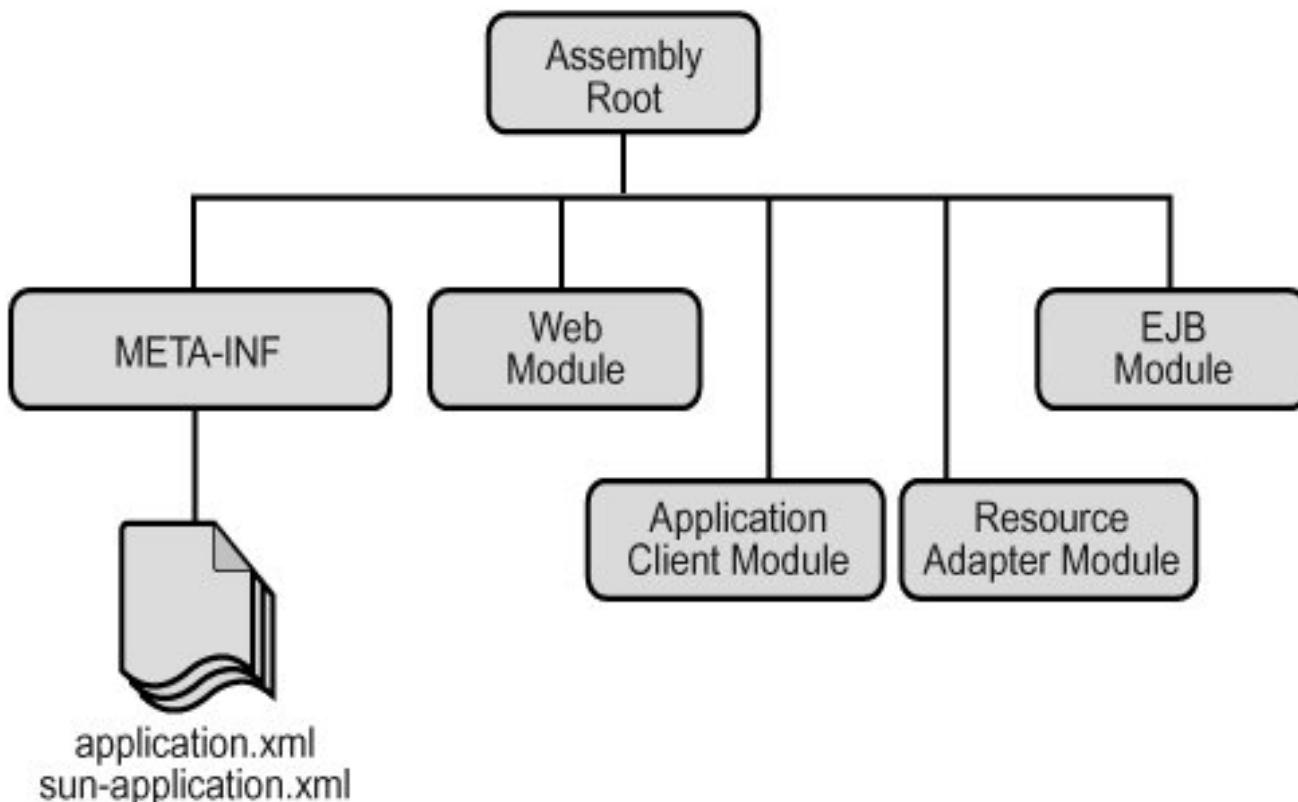
Business-Tier overview

Java Web Application Request Handling

Packaging applications
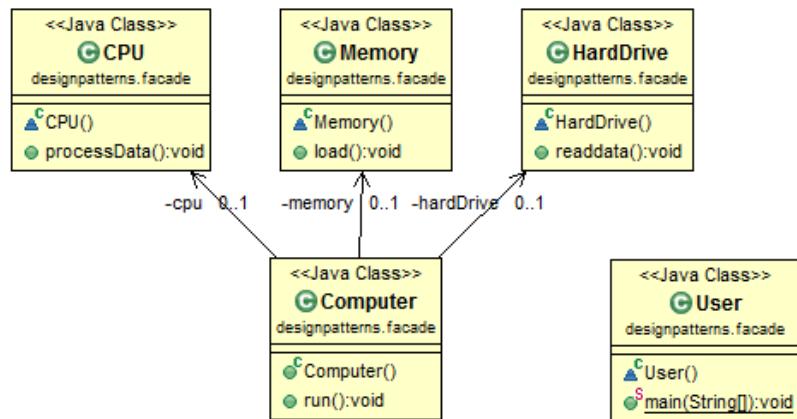
# Packaging applications

❖ **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
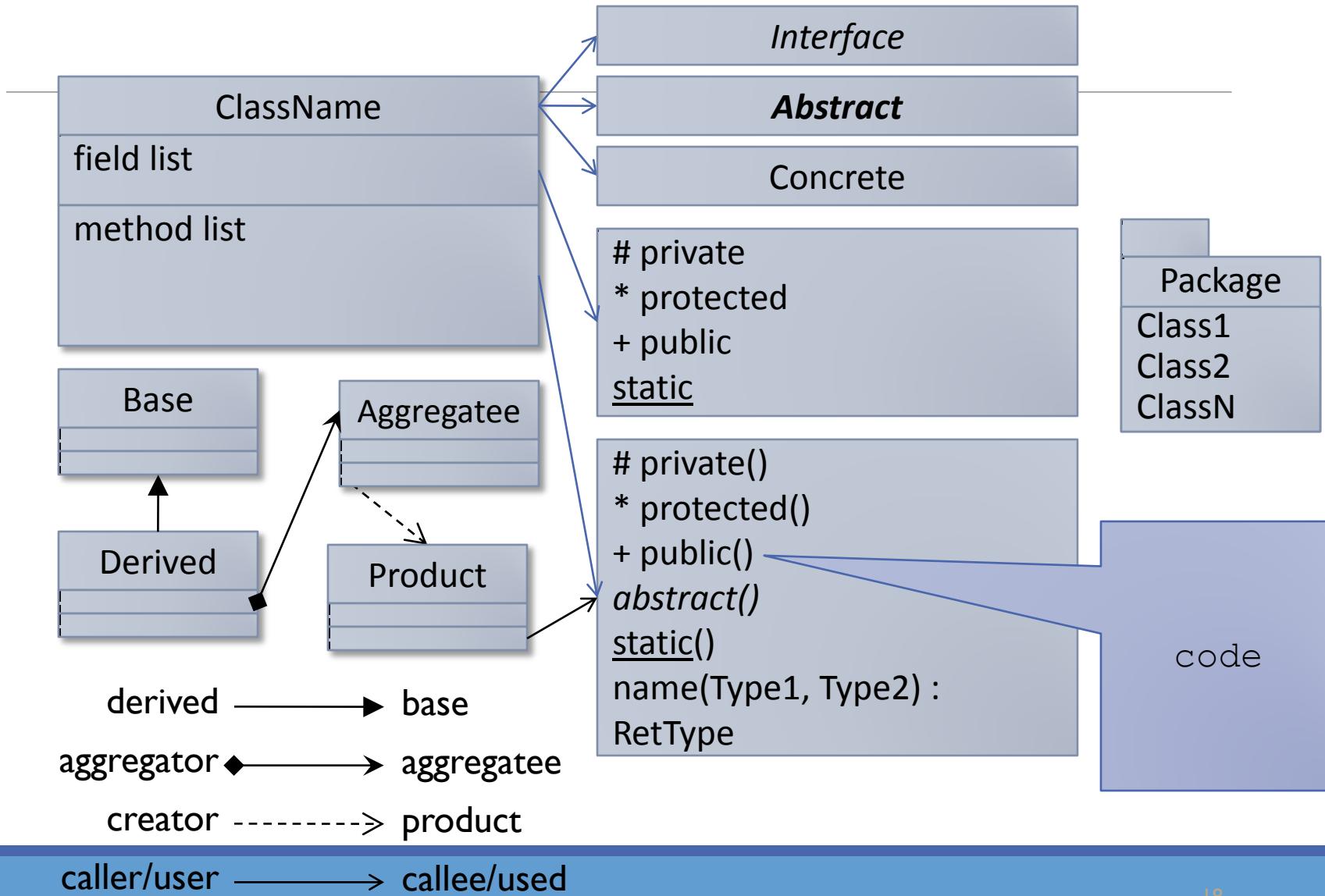
A **design pattern** is a **general reusable** solution to a commonly occurring problem in software design. A design pattern is **not** a **finished** design that can be transformed directly into code.

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.

Design pattern

UML class diagram recall

Interface		
ClassName	Abstract	
field list	Concrete	
method list		

private
* protected
+ public
static

Package
Class1
Class2
ClassN

Base

Aggregatee

Derived

Product

private()
* protected()
+ public()
abstract()
static()
name(Type1, Type2) :
RetType

code

derived ⟶ base

aggregator ◆⟶ aggregatee

creator ----> product

caller/user ⟶ callee/used

Simple Design Patterns

Singleton

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

```java
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.
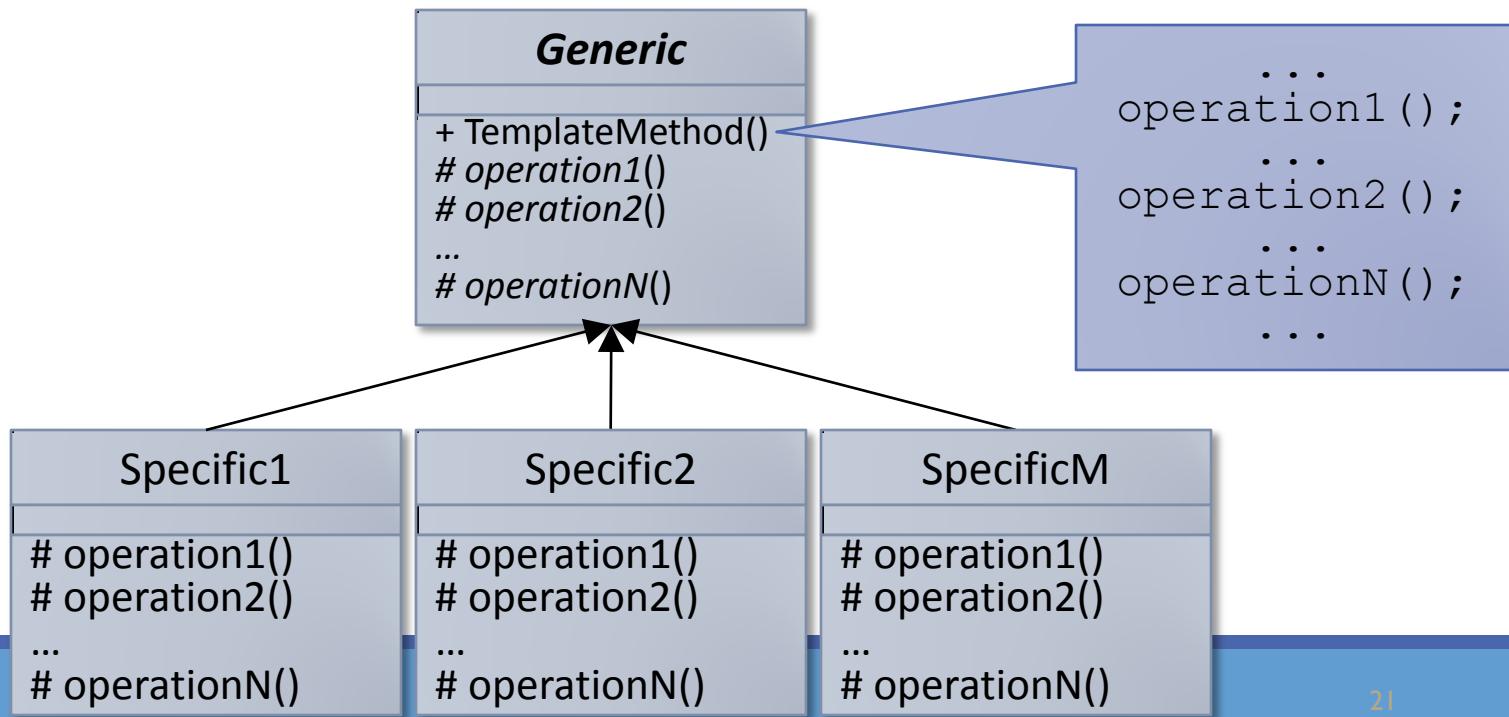
```
return _instance;
```

Singleton
_instance : Singleton
Singleton()   + getInstance() : Singleton

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.
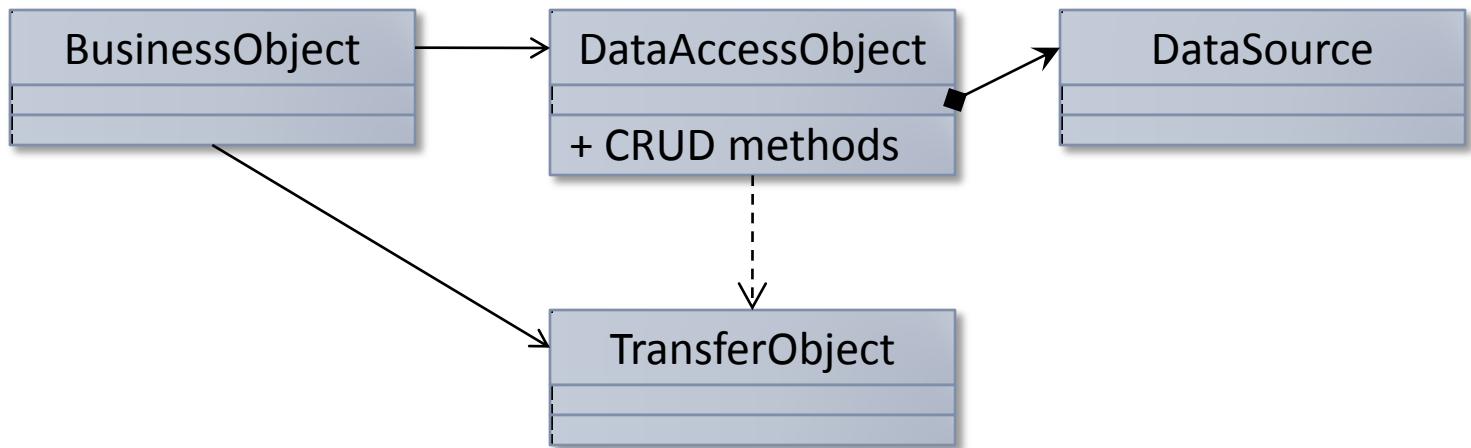
Data Access Objects (DAO) / 2

BusinessObject: represents the data client.

DataAccessObject: abstracts the underlying data access implementation for the BusinessObject to enable transparent access to the data source.
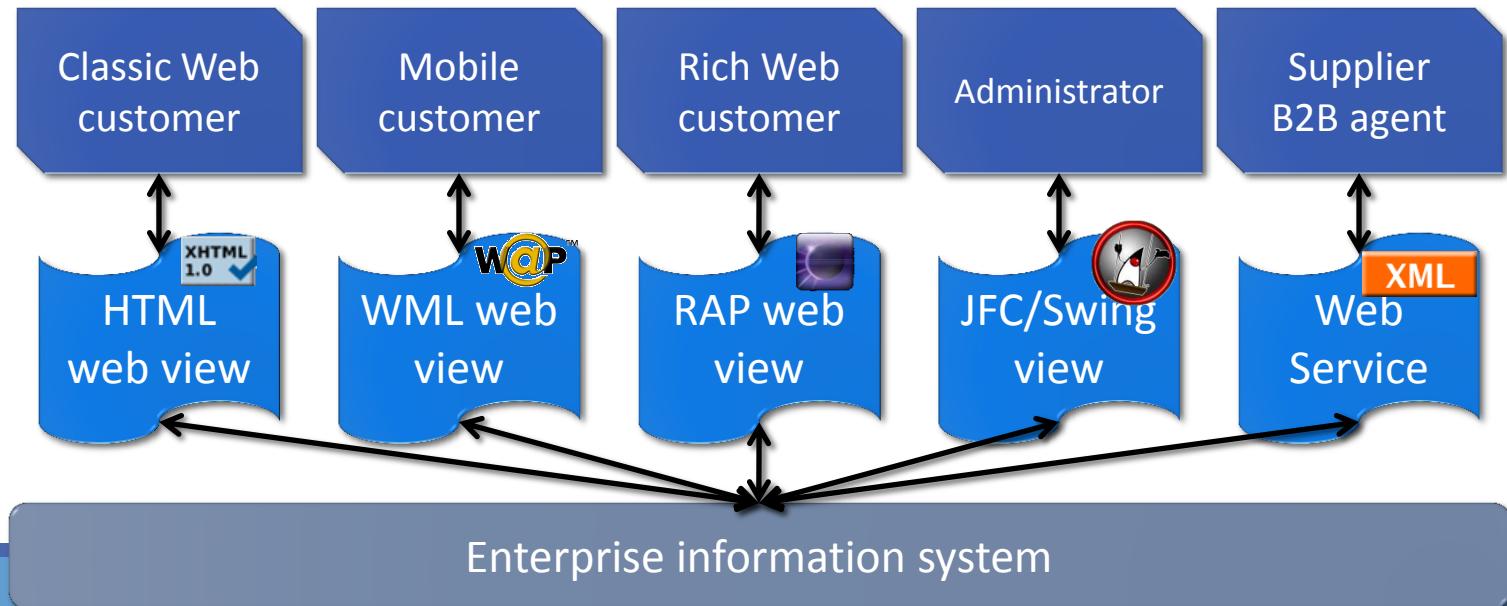
DataSource: represents a data source implementation.

TransferObject: the data carrier, DAO may use it to return data to the client.
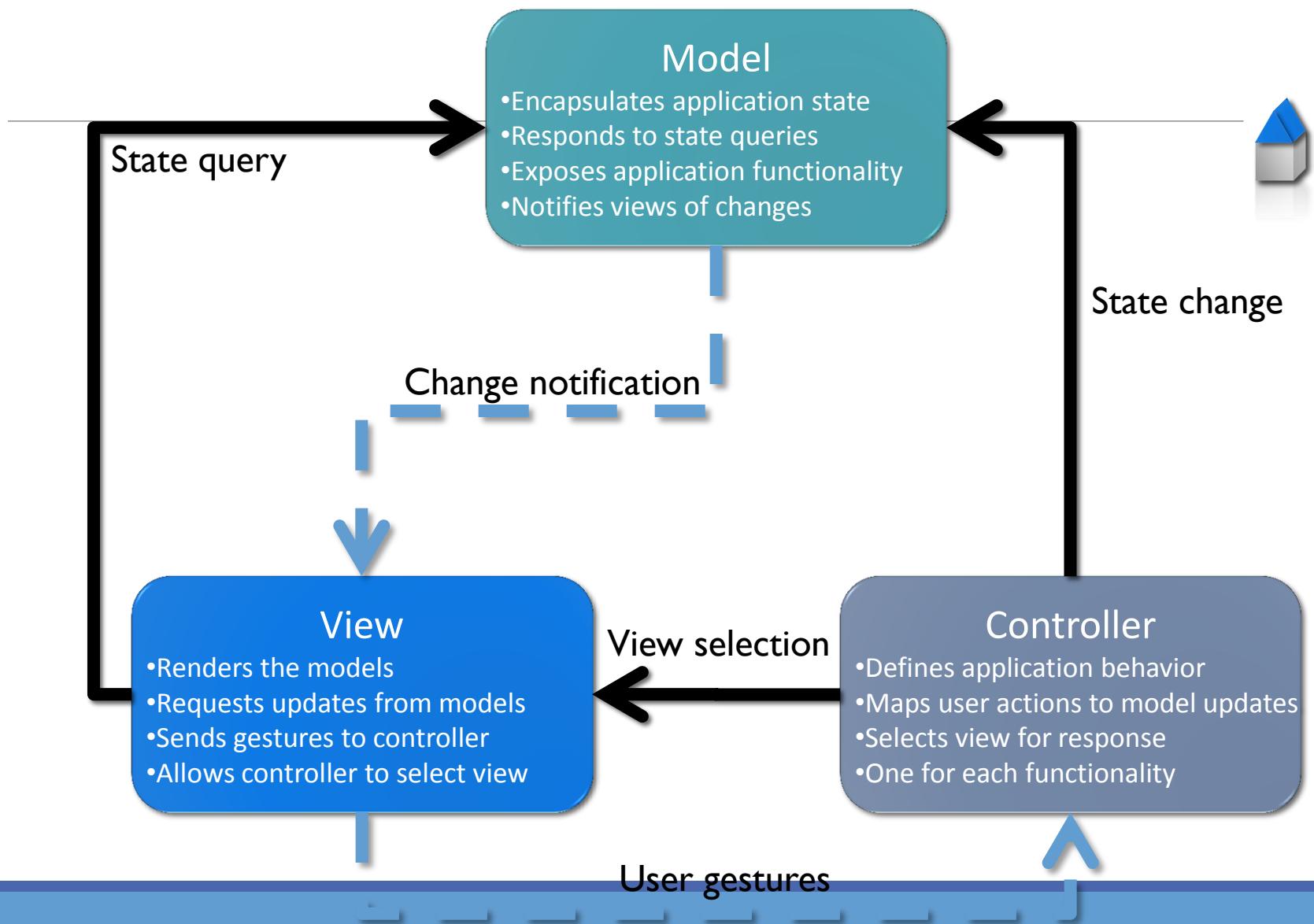
Model-View-Controller (MVC)

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.

Classic Web customer	Mobile customer	Rich Web customer	Administrator	Supplier B2B agent
HTML web view	WML web view	RAP web view	JFC/Swing view	Web Service

Enterprise information system

Model-View-Controller / 2

Model
- Encapsulates application state
- Responds to state queries
- Exposes application functionality
- Notifies views of changes

View
- Renders the models
- Requests updates from models
- Sends gestures to controller
- Allows controller to select view

Controller
- Defines application behavior
- Maps user actions to model updates
- Selects view for response
- One for each functionality

State query

State change

Change notification

View selection

User gestures

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

incoming request

return response

Front controller

handle request

Controller

create model

model

delegate rendering of response

return control

model

render response

View template

Web container

Session Façade

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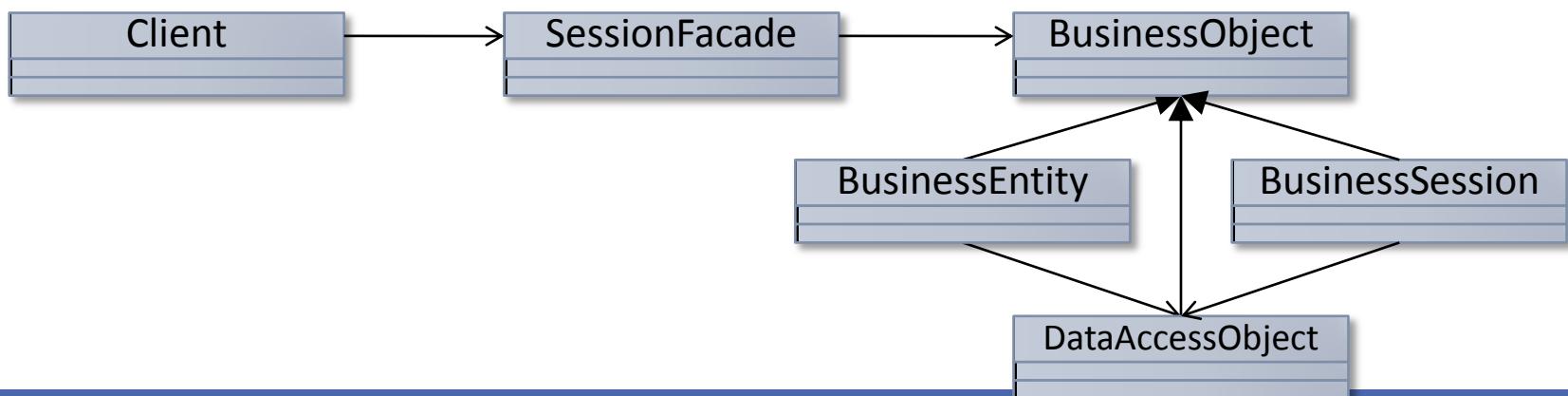
- Tight coupling, which leads to direct dependence between clients and business objects;
- Too many method invocations between client and server, leading to network performance problems;
- Lack of a uniform client access strategy, exposing business objects to misuse.

Session Façade / 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.

SessionFacade: a session bean which manages the relationships between numerous BusinessObjects and provides a higher level abstraction to the client.

BusinessObject: a role object that facilitates applying different strategies, such as session beans, entity beans and a DAO. A BusinessObject provides data and/or some services.

```
┌──────────────┐      ┌──────────────────┐      ┌──────────────────┐
│    Client    │─────▶│  SessionFacade   │─────▶│  BusinessObject  │
└──────────────┘      └──────────────────┘      └──────────────────┘
                                                    ▲           ▲
                                ┌───────────────────┐     ┌──────────────────┐
                                │  BusinessEntity   │     │ BusinessSession  │
                                └───────────────────┘     └──────────────────┘
                                           ▲                   ▲
                                        ┌──────────────────────┐
                                        │   DataAccessObject    │
                                        └──────────────────────┘
```

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.

sample.jsp

HTML
Java Code
HTML
HTML

The HTML displays the static portion of the page (text and graphics).
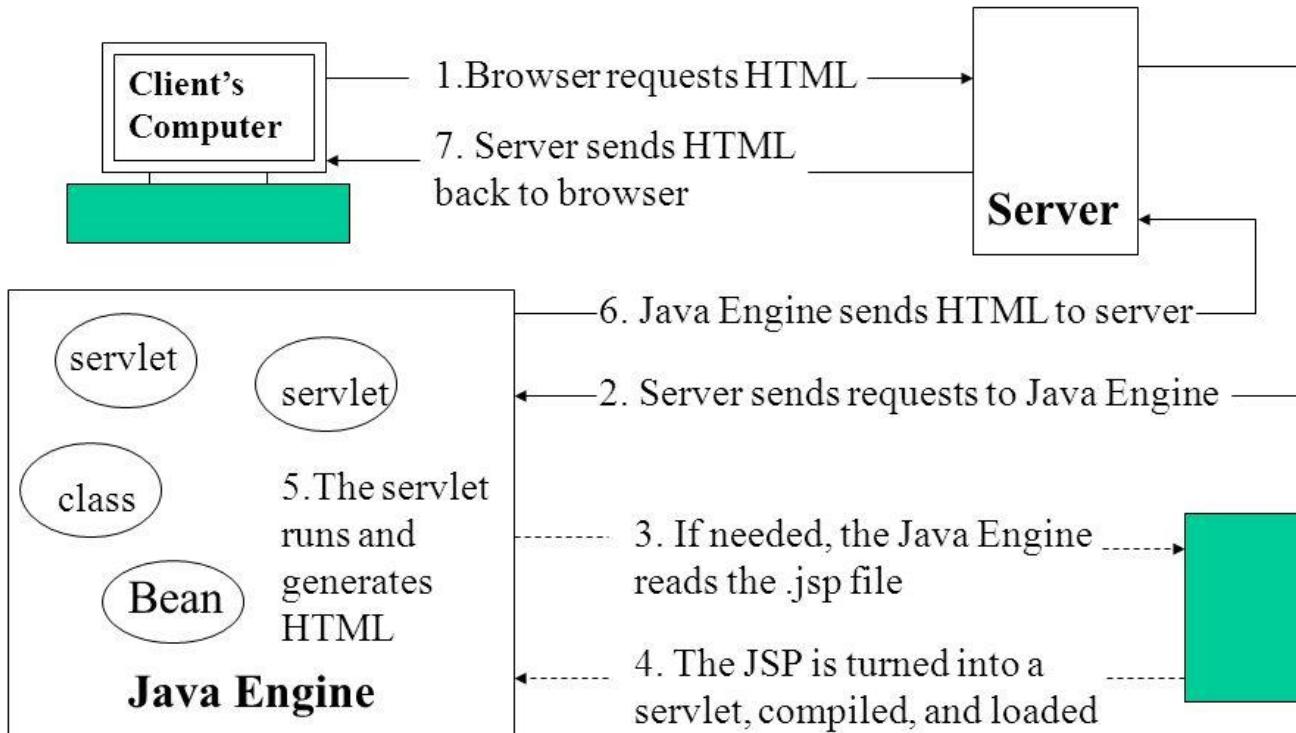
The Java code is used to generate dynamic content for the page (JDBC, RMI).

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

Java Server Pages (JSP)

Client's Computer

1. Browser requests HTML → **Server**

7. Server sends HTML back to browser

6. Java Engine sends HTML to server

2. Server sends requests to Java Engine

Java Engine

servlet

servlet

class

Bean

5. The servlet runs and generates HTML

3. If needed, the Java Engine reads the .jsp file

4. The JSP is turned into a servlet, compiled, and loaded

HTML & JSP

Compute Loan Payment

Loan Amount [   ]
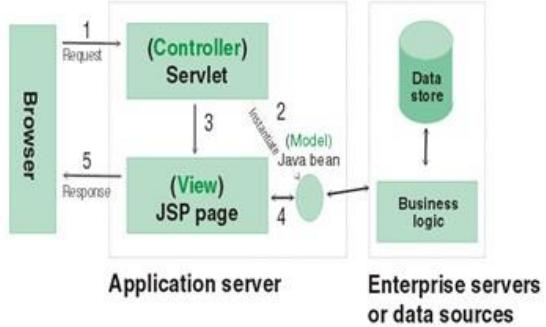
Annual Interest Rate [   ]

Number of Years [   ]

[Compute Loan Payment] [Reset]

```html
<!-- 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>
```

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

```java
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
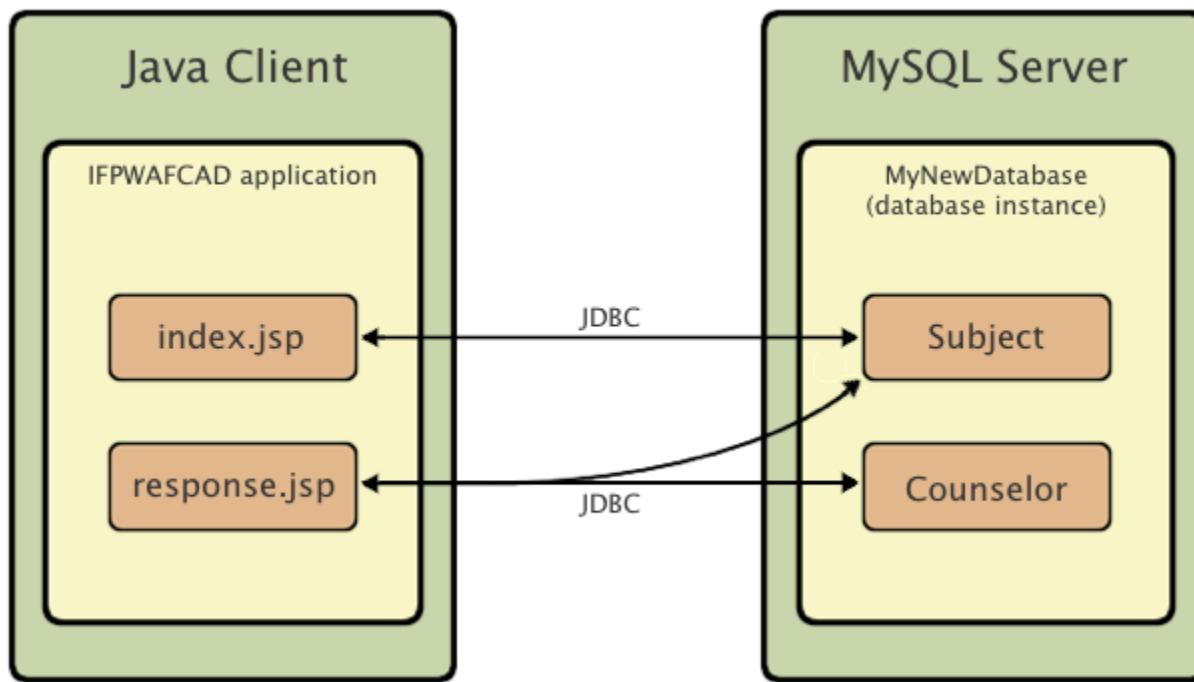
- A *servlet* is a Java class that can be loaded dynamically into and run by a special web server.
- This servlet-aware web server, is known as *servlet container*.
- Servlets interact with clients via a request-response model based on HTTP.
- Therefore, a servlet container must support HTTP as the protocol 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

```java
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

Web Project Discussions

❏ Web project
  ❏ Update status : % progress completed
  ❏ Project Task: deadline 23/11/18
    ❏ Submit user interface in a zip folder.
    ❏ HTML, JSP or Servlet.