Jakarta Server Pages

Overview

```
01  <html>
02      <%
03          int a = 5;
04          int b = 8;
05      %>
06      <body>
07          The first number is : <%= a %> <br/>
08          The second number is : <%= b %> <br/>
09          The sum is : <%= ( a + b ) %> <br/>
10      </body>
11  </html>
```

Overview

Jakarta Server Pages (JSP) = a **server-side technology** for building dynamic web pages.

Part of **Jakarta EE** (formerly Java EE).

JSPs are **compiled into servlets** by the container.

Mix HTML + Java (scriptlets, expressions, declarations).

Provides **implicit objects** (request, response, session, etc.).

Supports **directives** (page, include, taglib)

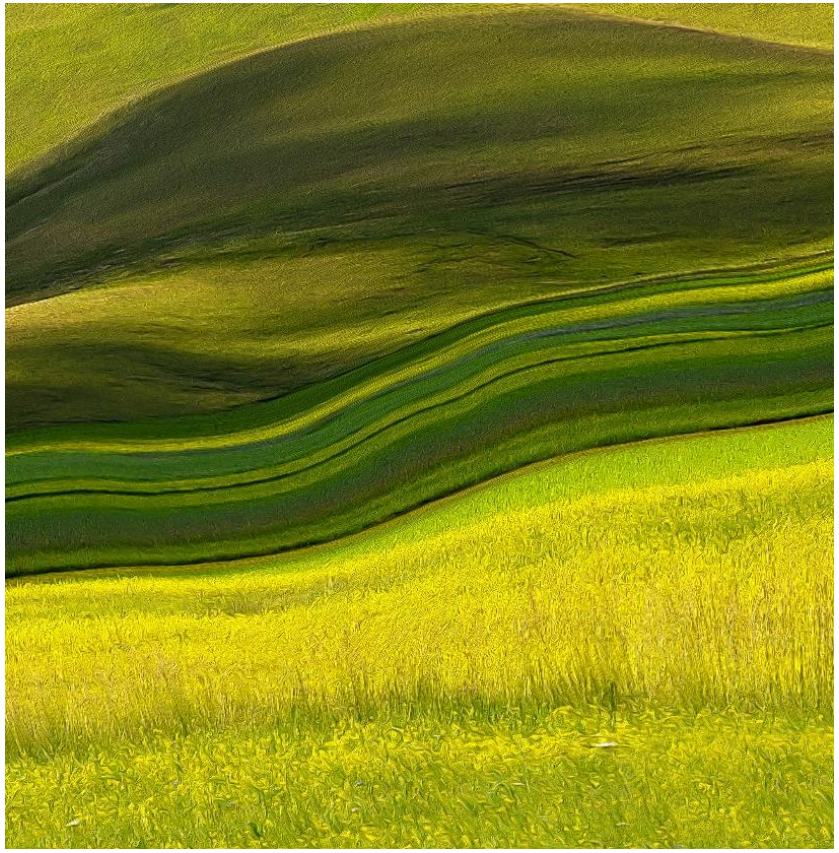
Can use **JSTL & Expression Language (EL)** for cleaner syntax

Overview

Life Cycle of JSP:

1. **Translation** → JSP converted into a servlet.
2. **Compilation** → Servlet compiled into bytecode.
3. **Loading & Instantiation** → Servlet loaded by container.
4. **Request handling** →method called for each reques
5. **Destroy** → Servlet destroyed when app stops.

Implicit Objects



Implicit Objects

Object	Type
request	jakarta.servlet.http.HttpServletRequest
response	jakarta.servlet.http.HttpServletResponse
out	jakarta.servlet.jsp.JspWriter
session	jakarta.servlet.http.HttpSession
application	jakarta.servlet.ServletContext
config	jakarta.servlet.ServletConfig
pageContext	jakarta.servlet.jsp.PageContext
page object	jakarta.servlet.jsp.HttpJspPage
exception	java.lang.Throwable

request Object

Represents HTTP request from client.

Methods:

- `getParameter()`
- `getAttribute()`

Example:

```
String name = request.getParameter("username");
```

response Object

Represents HTTP response to client.

Methods:

- `setContentType(),`
- `sendRedirect()`

Example:

```
response.sendRedirect("home.jsp");
```

out Object

Represents JspWriter for sending output.

Methods:

- `print()`,
- `println()`,
- `flush()`.

Example:

```
out.println("<h2>Hello</h2>");
```

session Object

Represents user session across requests.

Methods:

- `setAttribute()`,
- `getAttribute()`,
- `invalidate()`.

Example:

```
session.setAttribute("user","Alice");
```

application & config

application (ServletContext): shared across app.

Example:

- `application.setAttribute("appName","Portal");`
- `config (ServletConfig):` initialization params.

Example:

```
config.getInitParameter("dbDriver");
```

pageContext, page & exception

pageContext: gives access to all implicit objects.

page: refers to current JSP (like this).

exception: only available in error pages.

out Object

Represents JspWriter for sending output.

Methods:

- `print()`,
- `println()`,
- `flush()`.

Example:

- `out.println("<h2>Hello</h2>");`

session Object

Represents user session across requests.

Methods:

- `setAttribute()`,
- `getAttribute()`,
- `invalidate()`.

Example:

- `session.setAttribute("user","Alice");`

application & config

application (ServletContext): shared across app.

Example:

```
application.setAttribute("appName","Portal");
```

```
config (ServletConfig): initialization params.
```

Example:

```
config.getInitParameter("dbDriver");
```

pageContext

pageContext: gives access to all implicit objects.

Can store attributes in different scopes (page, request, session, application)

```
<%
  pageContext.setAttribute("x", 100);
  out.println(pageContext.getAttribute("x"));
%>
```

page

page: refers to current JSP (like this).

Rarely used directly, but available.

```
<%
   out.println("This page object: " +
page.toString());
%>
```

exception

Available **only in error pages** (`isErrorPage="true"`)

Represents the **Throwable** object that caused the error.

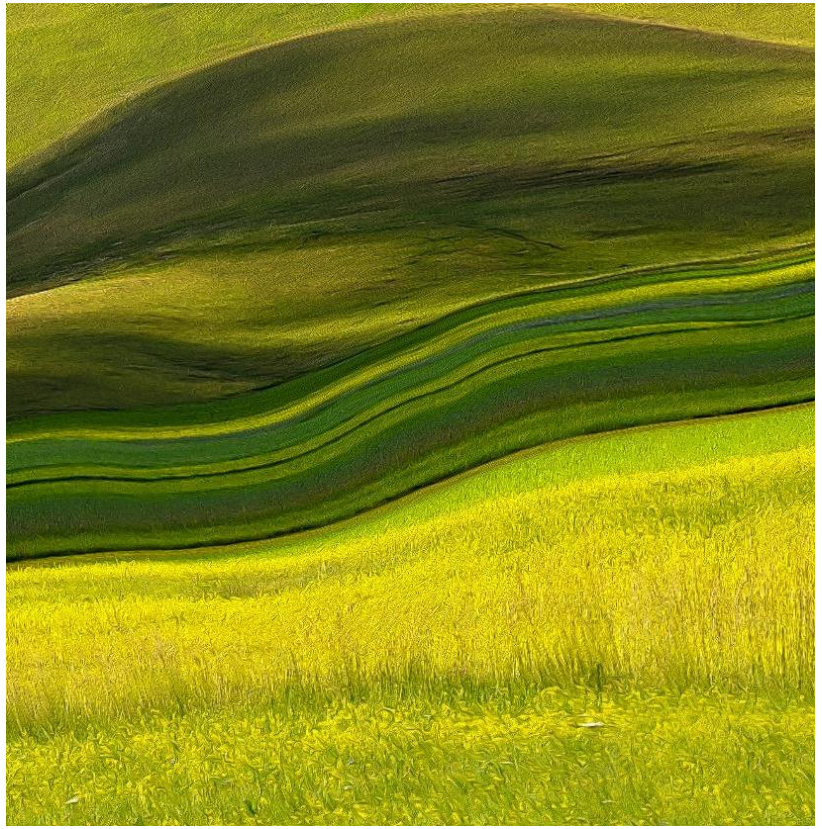
```
<%@ page isErrorPage="true" %>

<%

  out.println("Error: " + exception.getMessage());

%>
```

Directives



Directives

Directives control the processing of an entire JSP page. It gives directions to the server regarding processing of a page.

```
<%@ directive name [attribute name="value" attribute name="value" ]%>
```

There are three types of Directives in JSP:

- **Page** Directive
- **Include** Directive
- **TagLib** Directive

Directives - Page

```
<%@ page attribute1="value1" attribute2="value2" …%>
```

Attributes :

- import; session;
- isErrorPage; errorPage;
- ContentType; isThreadSafe; extends; info;
- language; autoflush; buffer

Directives – Page

Page – import: **attribute** of the **page directive** allows a JSP to use Java classes/packages

```
<%@ page import="java.util.List, java.util.ArrayList" %>
<%
    List<String> students = new ArrayList<>();
    students.add("Alice");
    students.add("Bob");
    out.println("Students: " + students);
%>
```

Directives – Page

Page – session: session management

`<%@ page session="true"%>:` (default) → JSP creates/uses an **HttpSession** object, `session` is available

```
session.setAttribute("user", "Alice");
out.println(session.getAttribute("user"));
```

`<%@ page session="false"%>:` No HttpSession created, `session` not available

Directives – Page

Page – isErrorPage. Indicates if this page is an error handler

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

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

Page – errorPage: Indicates the page that will handle errors that may occur in this page

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

Page – contentType: used to set the content type of a JSP page

```
<%@ page contentType="text/html"%>
```

Directives – Include

Include: directive is used to **statically include** another file into a JSP **at translation (compile) time**

The content of the included file is **copied into the JSP** before it is compiled into a servlet

```
<%@include file ="value"%>
```

index.jsp

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

```

```

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

Directives - taglib

`taglib` **directive** is used to declare a **custom tag library** in a JSP page.

It allows JSP to use **custom tags** instead of Java code inside the page
Typically used with **JSTL (Jakarta Standard Tag Library)** or custom tag libraries

```
<%@ taglib uri="uri_of_tag_library" prefix="prefixName" %>
```

Directives - taglib

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
prefix="c" %>
<html>
  <body>
    <c:set var="name" value="Jakarta JSP" />
    <c:out value="${name}" />
  </body>
</html>
```

Actions

Actions: provides several standard action tags intended for a specific tasks such **including** other resource or **forwarding the request** to other resources or working with java bean objects..

Syntax : <jsp:action_name attribute="attribute_value" />

Directives vs Actions:

- Directives are used **during translation phase** while actions are used **during request processing phase**.
- Actions are **re-evaluated** each time the page is accessed

Actions - jsp:include

used to include any other resource (may be html, jsp etc) at a runtime.

Syntax: **<jsp:include page="another file" />**

```
<html>
  <head>
    <title> Main Page </title>
  </head>
  <body>
    <H4> This is main page and to explain include directive </H4>
    <br/>
    <jsp:include page="displayIP.jsp" />
  </body>
</html>
```

Actions - jsp:forward

used to forward request any other resource (may be html, jsp etc).

Syntax: **<jsp:forward page="another file" />**

```
<html>
  <head>
   <title> Main Page </title>
  </head>
  <body>
    <H4> This is main page and to explain include directive </H4>
    <br/>
    <jsp:forward page="displayIP.jsp" >
    <jsp:param name="param1" value="value1"/>
    <jsp:param name="param2" value="value2"/>
    </jsp:forward>
  </body>
</html>
```

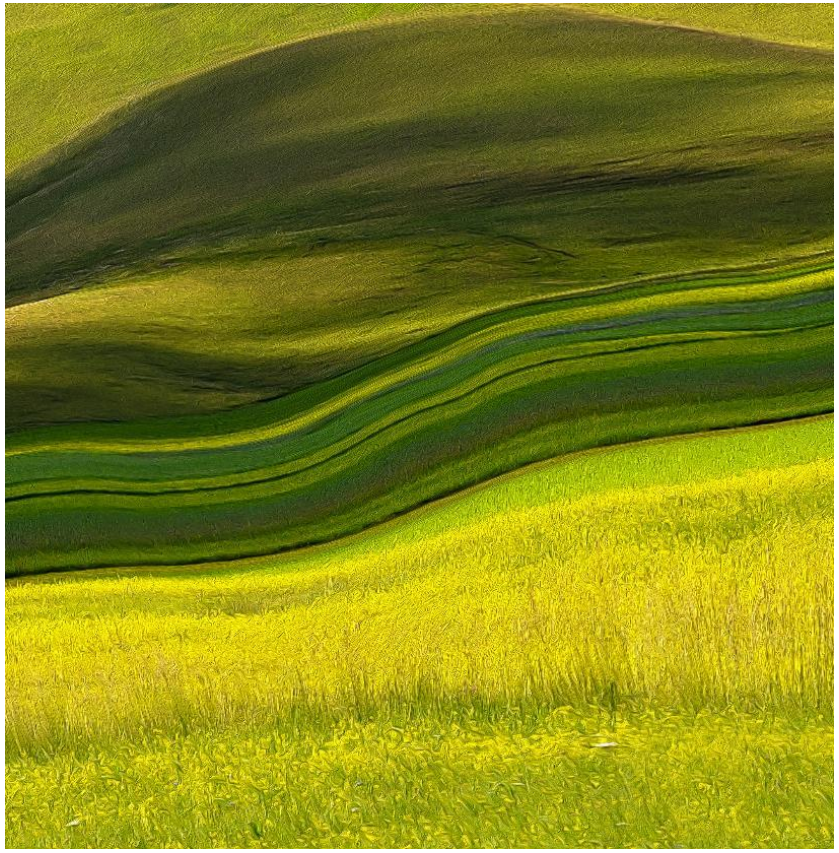
Actions - jsp:useBean

Creates a scripting variable associated with a Java object.

Syntax:

```
<jsp:useBean
  id= "instanceName"
  scope= "page | request | session | application"
  class= "packageName.className"
</jsp:useBean>
```

JavaServer Pages Standard Tag Library (JSTL)



JSTL

JSTL: a collection of **custom tag libraries** for solving common problems such as **iterating over a map** or **collection**, **conditional testing**, **XML processing**, and even **database access** and **data manipulation**

To use a JSTL library in a JSP page, use the taglib directive with the following format:

```
<%@ taglib prefix="prefix"  uri="uri" %>
```

JSTL

```
<!-- JSTL Jakarta (dùng cho taglib fmt, c:forEach, ...) -->
<dependency>
    <groupId>jakarta.servlet.jsp.jstl</groupId>
    <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
    <version>3.0.2</version>
</dependency>
<dependency>
    <groupId>org.glassfish.web</groupId>
    <artifactId>jakarta.servlet.jsp.jstl</artifactId>
    <version>3.0.1</version>
</dependency>
```

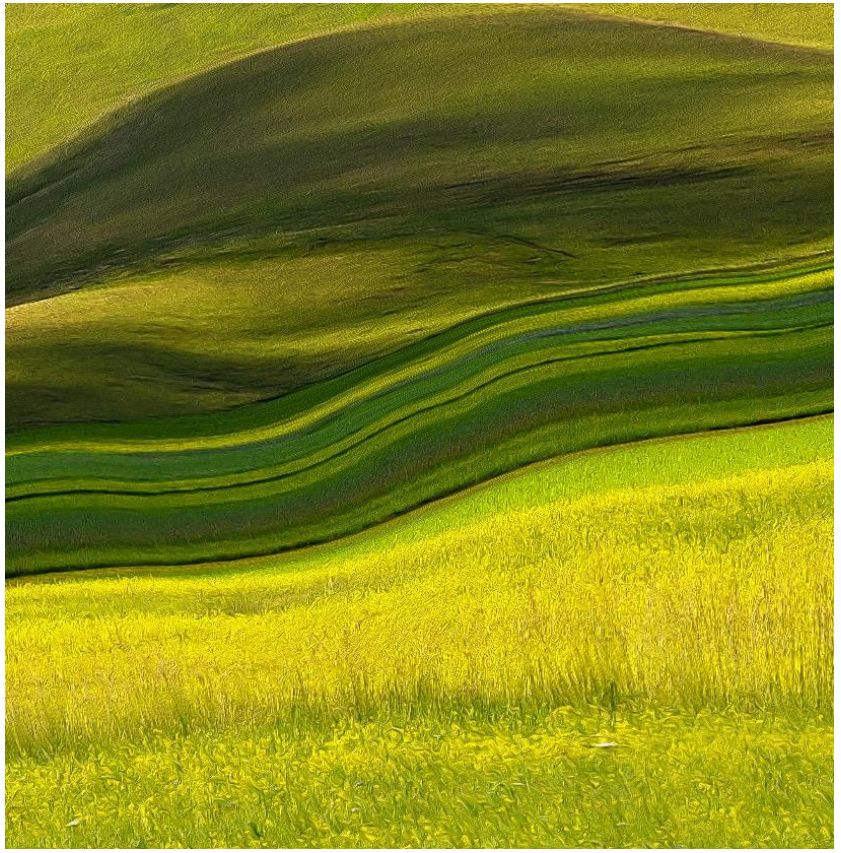
JSTL

Functional Area	URI	Prefix
core	jakarta.tags.core	c
XML processing	jakarta.tags.xml	x
capable formatting	jakarta.tags.fmt	fmt
relational db access (SQL)	jakarta.tags.sql	sql
Functions	jakarta.tags.functions	fn

JSTL Libraries

```
1. <%@ taglib prefix="c" uri="jakarta.tags.core" %>
2. <%@ taglib prefix="fn" uri="jakarta.tags.functions"%>
3. <%@ taglib prefix="sql" uri="jakarta.tags.sql"%>
4. <%@ taglib prefix="fmt" uri="jakarta.tags.fmt"%>
5. <%@ taglib prefix= ="x"  uri= "jakarta.tags.xml"%>
```

Expression Language (EL)



JSP Expression Language (EL)

Provide another **convenient way** to **minimize** the use of **scripting tag** in JSP

To **access** the data , **process** the data, **store** it in some scope in JSP

General syntax: **${expression}**

Expression Language Implicit Objects (1)

Name	Description
pageScope-	use to get the value of attribute stored in page scope
requestScope	Use to get the value of attribute stored in request scope.
sessionScope	use to get the value of attribute stored in session scope
applicationScope	use to get the value of attribute stored in application scope.
cookie	use to get the cookie value
initParam	use to get the context init params, we can't use it for servlet init params

Expression Language Implicit Objects (2)

Name	Description
pageContext	provides access to many objects request, session etc.
param	use to get request parameters (valid only for single value)
paramValues	use to get request parameter to an array of values
header	use to get the request header value (valid for single value)
headerValues	use to get the request header value to an array of values.

Expression Language Operators

Operator Type	Description
Arithmetic	+ , - , / or div , * , % or mod
Logical	&& , || , !
Relational	== (eq), != (ne), < (lt), > (gt), <= (le) , >= (ge)
Dot (.) Operator	Use to access property of any bean: ${bean.property_name}
[ ] operator	Use to get the data from array and list along with beans: ${myList[1]} or ${myList["1"]}

Disable Expression Language

```
<%@ page isELIgnored ="true | false" %>
```

Exception Handling

Ways to handle Exceptions – Try Catch

```
<%
    try
        {       }
    catch(Exception    e)
        {       }
%>
```

Ways to handle Exceptions – isErrorPage, errorPage

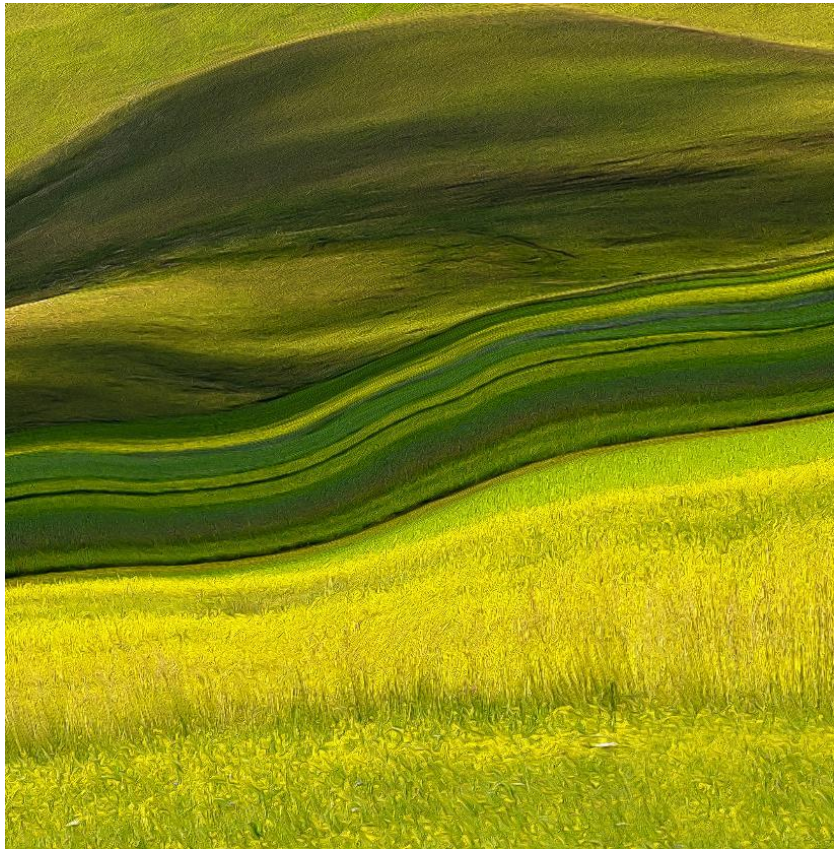
isErrorPage: use to inform container that declaring JSP page is an error page

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

errorPage: use to set the error page for the JSP

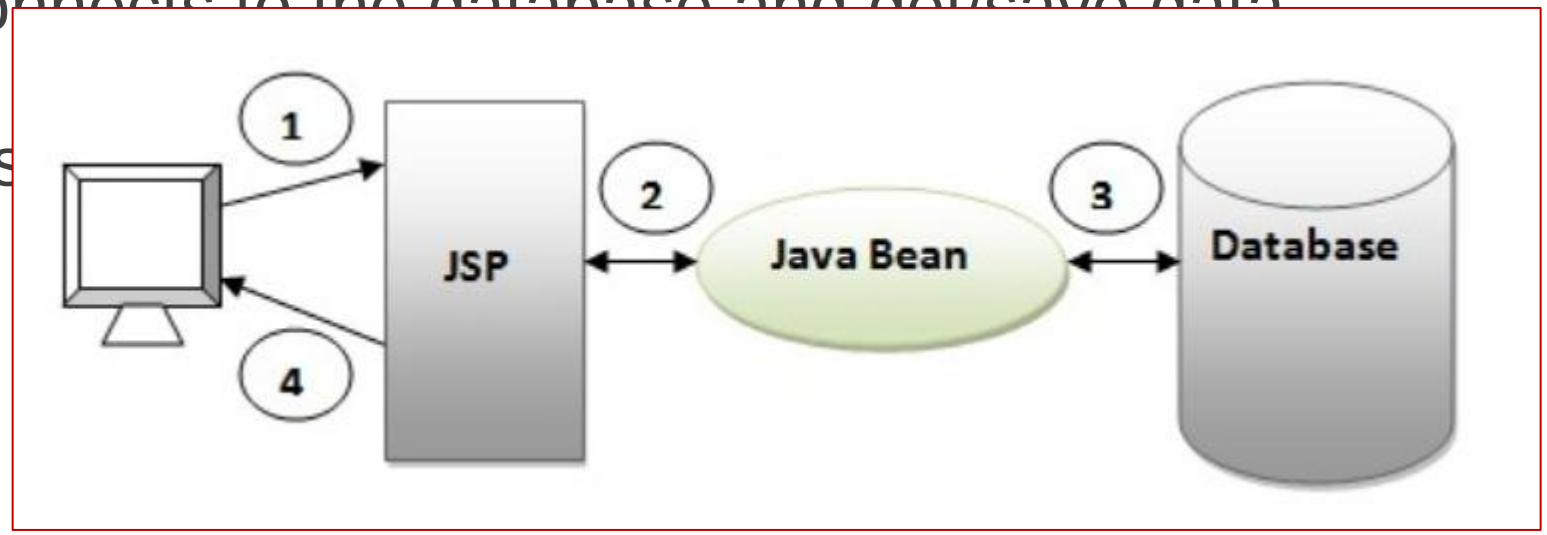
```
<%@ page errorPage="jsp page name" %>
```

JSP – Application design



Model 1 Architecture

1. Browser sends request for the JSP page
2. JSP accesses Java Bean and invokes business logic
3. Java Bean connects to the database and get/save data
4. Response is s



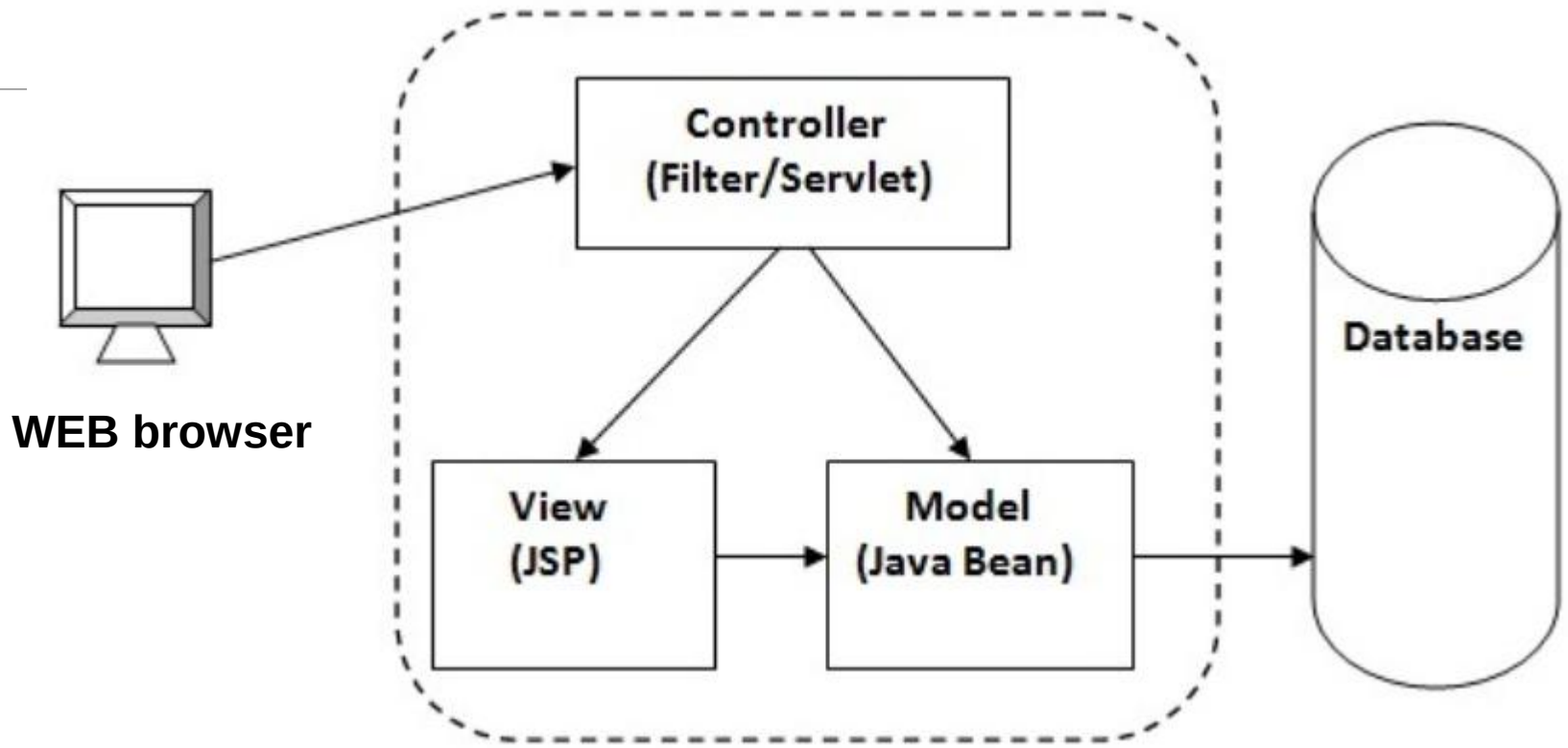
Model 1 Architecture

Easy and Quick to develop web application

Navigation control is decentralized: If JSP page name is changed that is referred by other pages→ change it in all the pages that leads to the maintenance problem

It is better for small applications but not for large applications

MODEL 2-MVC (Model View Controller)



MODEL 2-**MVC (Model View Controller)**

Navigation control is centralized. Now only controller contains the logic to determine the next page.

Easy to maintain

Easy to extend

Easy to test

Better separation of concerns

We need to write the controller code self.

If we change the controller code, we need to recompile the class and redeploy the application

MODEL 2-**MVC Example**

1. A Product class that is the template for the model objects. An instance of this class contains product information.

2. A ProductForm class, which encapsulates the fields of the HTML form for inputting a product. The properties of a ProductForm are used to populate a Product.

3. A ControllerServlet class, which is the controller of this Model 2 application.

4. An action class named SaveProductAction.

5. Two JSP pages (ProductForm.jsp and ProductDetails.jsp) as the views.

6. A CSS file that defines the styles of the views. This is a static resource.

QUESTIONS