EBU6501 - Middleware

Week 1, Day 2: Introduction to JavaServer Pages

Dr. Gokop Goteng

Queen Mary
University of London

Lecture Aim and Outcome

- Aim
  - The aim of this lecture is to teach students how to write programs using JSP
- Lecture Outcome
  - At the end of this lecture students should be able to do the following:
    - Write simple JSP programs
    - Know the deployment directories where different JSP applications are located in Apache Tomcat Container
    - Understand and use Expression Language (EL) syntax in JSP

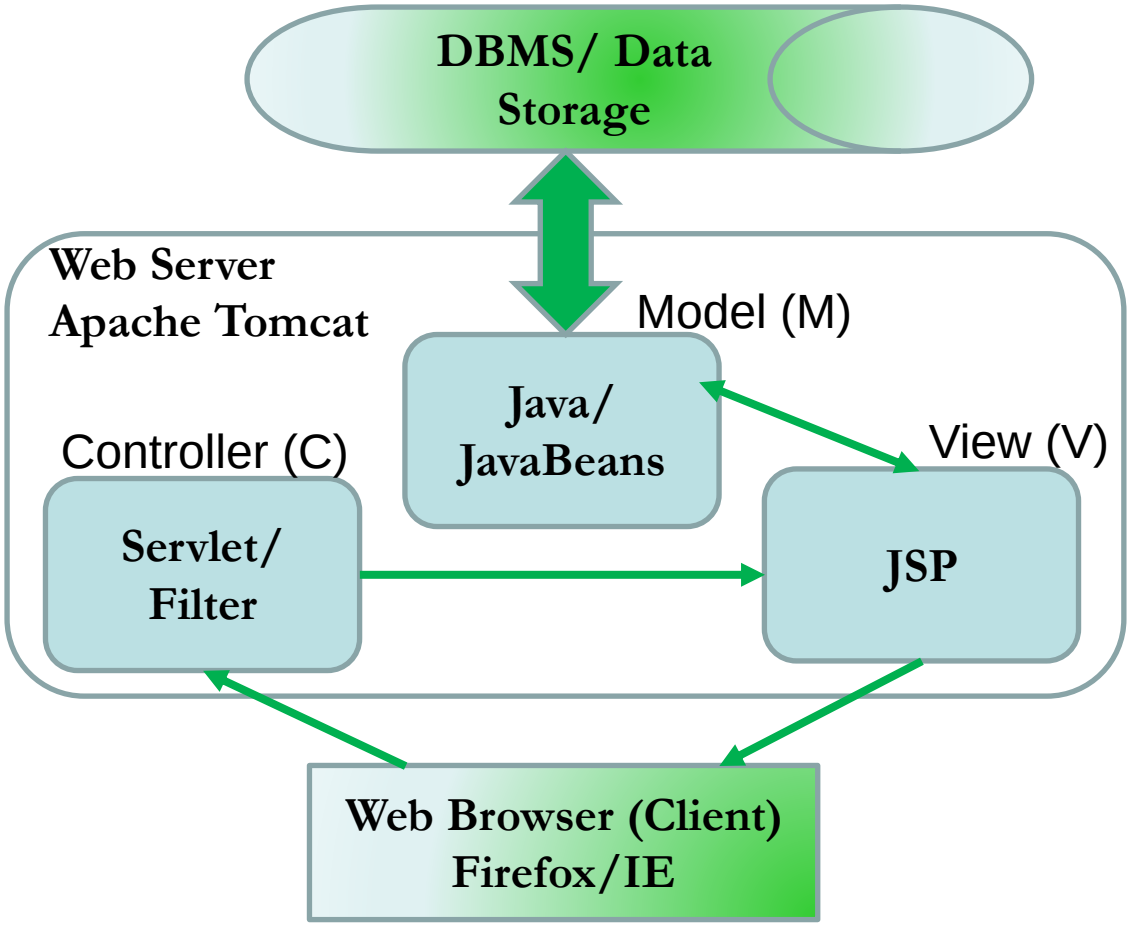
Lecture Outline

- JavaServer Pages – Introduction
- Architecture of JSP
- Web Server
- The Client
- Apache Tomcat Deployment Directory Overview
- Servlet – Controller
- Lifecycle of a Servlet
- Java and JavaBeans – Model
- Web Container – Apache Tomcat
- JSP Implementation
- JSP and Java Codes
- JSP Directives
- JSP Expression
- JSP Scriptlet
- JSP Declaration
- JSP Actions
- JSP Expression Language (EL)
- JSP and Bean
- Lifecycle of a JSP
- Quiz Questions

JavaServer Pages - Introduction

- JavaServer Pages (JSP)
  - It is similar to PHP
  - It is based on the Java programming language
  - Programmers develop dynamic modern web pages using JSP
  - It is developed in combination of XML, HTML and CSS
  - It is usually deployed using Apache Tomcat Servlet Container and Web server
  - It is originally created by Sun Micro Systems but is now acquired by Oracle
  - Java Servelets are Java classes used to replicate or extend the capabilities of the web server deployed with JSP
  - JSP is used to implement the “View” part of MVC (Model View Controller) design pattern

Architecture of JSP



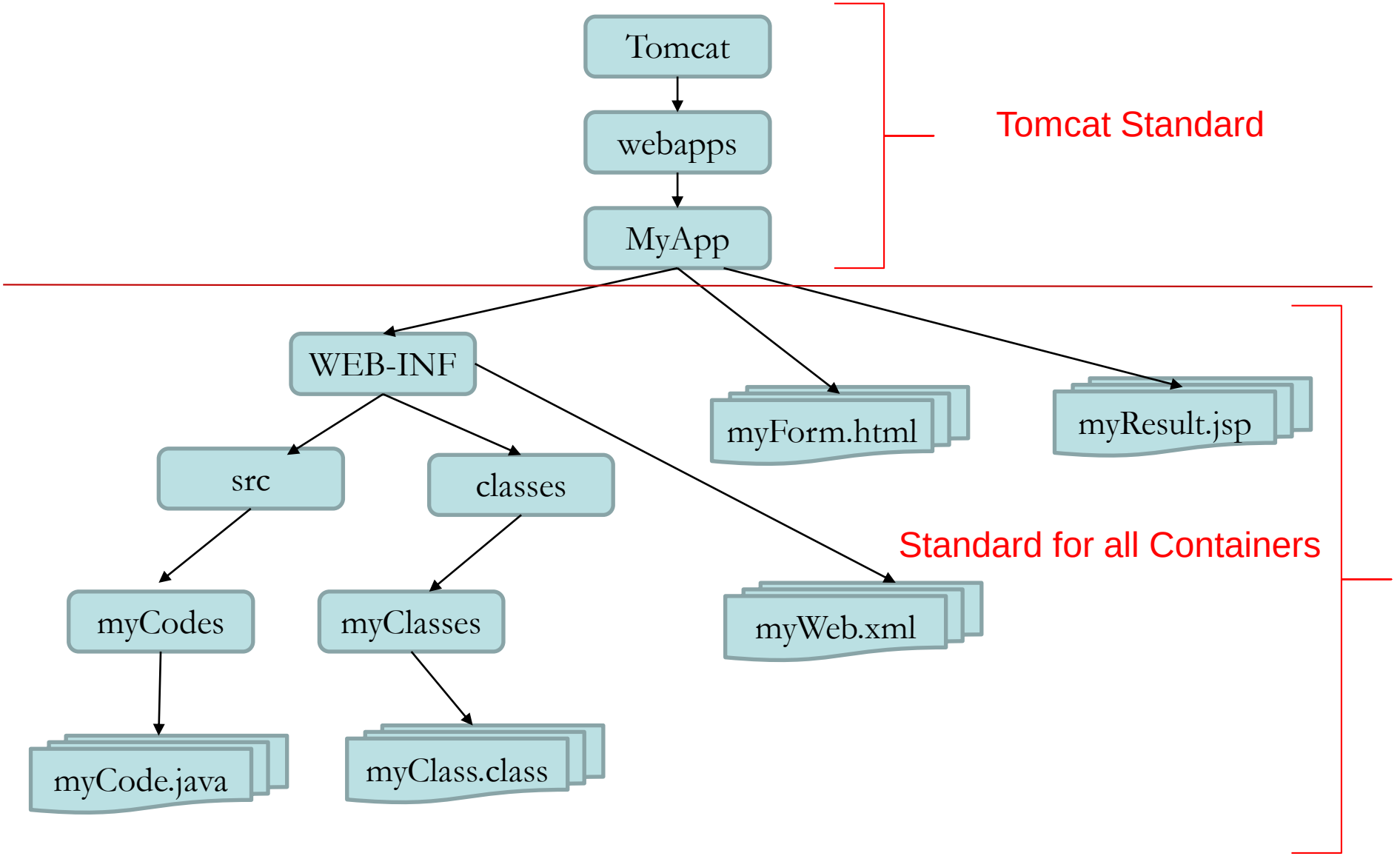
Web Server

- The web server receives the request from clients
- The web server looks for the resource that the client has requested for
- The web server finds the resource and
- The web server sends back results of the request to clients

The Client

- The web client (Browser) allows the user to request for resources from the web server
- The web client displays the results of the resource obtained from the web server to the user
- Both the web server and web client uses HTML and HTTP to communicate and exchange information
- HTML instructs the client (browser) how to display the contents to the user
- Both server and client use HTTP as the communication protocol on the web
- The web server uses HTTP to send information to the client as HTML

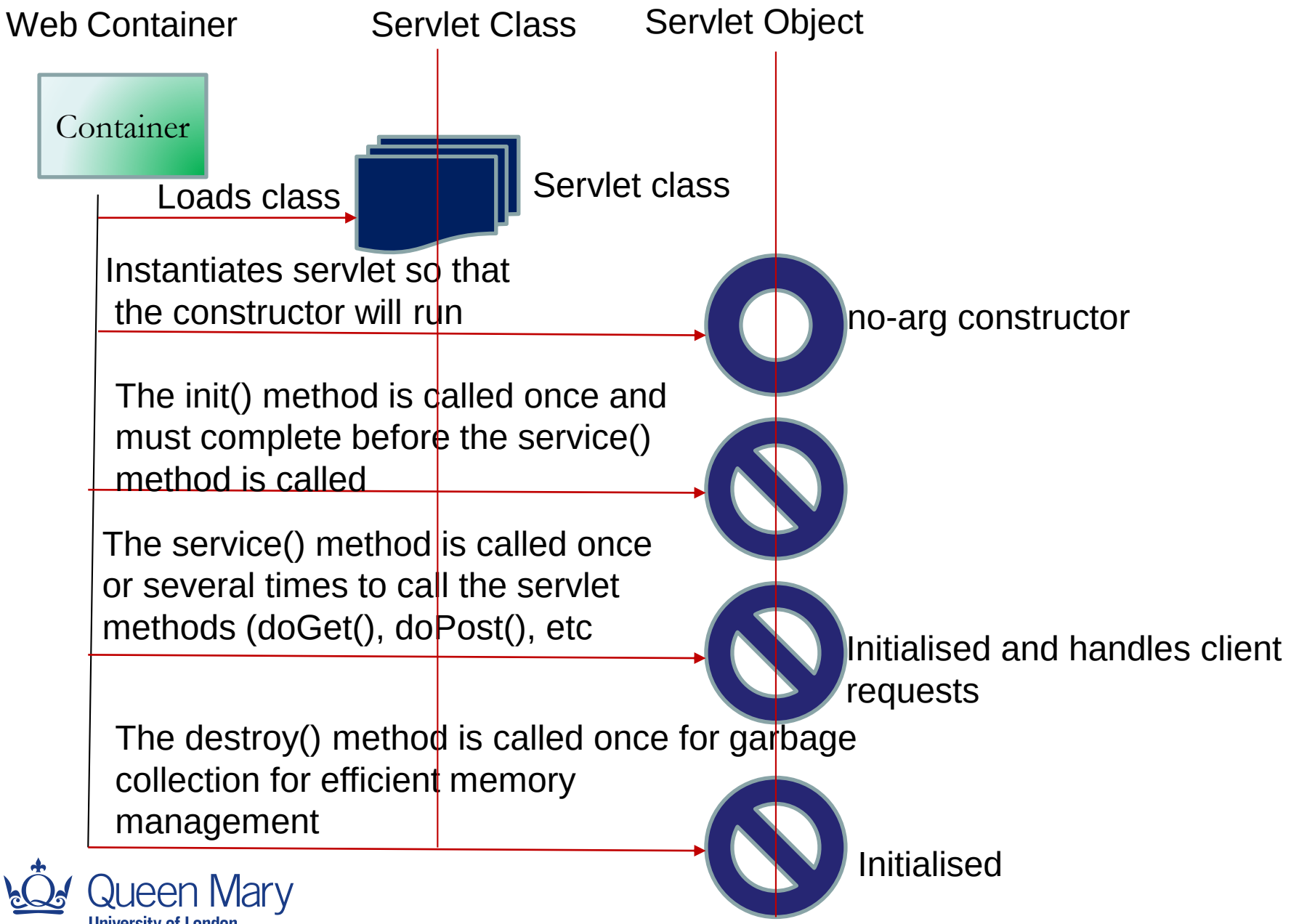
Apache Tomcat Deployment Directory Overview



Servlet - Controller

- Servlets main purpose is to serve clients
  - They take clients request and give back to clients response
  - Servlets use the HttpServletRequest and HttpServletResponse objects for serving clients
- Servlets are managed by the Container

Lifecycle of a Servlet



Java and JavaBeans - Model

- Java codes or JavaBeans are used to implement the business logic (actions) for the web application
- It can consist of plain old java objects (POJOs)
- It can contain POJOs and JavaBeans

Web Container – Apache Tomcat

- Servlets are deployed in a container
  - This is because servlets do not have the “main()” method that initiates the running of the application
- The container’s functions are
  - Support for JSP
    - The container translates JSP codes to Java codes
  - Life-cycle management
    - It controls the initiation and termination of servlets
  - Security
    - It uses the deployment descriptor (DD) for declaration of authentication and authorisation configurations, which avoids hard-coding security problems
  - Communication
    - The container facilitates communication among servlets, JSPs and web servers
  - Multi-threaded support
    - The container facilitates multi-threaded communication and request by creating new threads for every servlet request.

JSP Implementation

- Java codes implemented within HTML is known as JSP
- JSP can be implemented on its own in programming
- JSP can also be implemented as the View (V) component of the MVC model
  - Where the Model is implemented by Java or JavaBeans and the Controller is implemented by the servlet
- Java syntax are implemented in JSP within the “<%” and “%>” delimiters

JSP Implementation

- First Exercise
  - Create the file “hello_bupt.html”

```
<html>
 <body>
     Hello, welcome to BUPT!
</body>
</html>
```

  - Place this file “hello_bupt.html” at the root of your tomcat deployment e.g. tomcat/webapps/bupt/hello_bupt.html
  - Start your tomcat container and browser
  - Type http://localhost:8080/bupt/hello_bupt.html on your browser’s window
  - You will see “Hello, welcome to BUPT!” displayed
- Rename the same file “hello_bupt.html” to “hello_bupt.jsp” and repeat the same steps above with the new hello_bupt.jsp file
  - You will get the same output on your browser
  - This is your first JSP application

JSP and Java Codes

- You can make your JSP to provide dynamic functionalities by adding Java codes within the “<%=“ and “%>” delimiters
- Example

```
<html>
<body>
<%
        java.util.Date myDate = new java.util.Date();
%>
Hello, welcome to BUPT! The time is <%= myDate %>
</body>
</html>
```

JSP Directives

- JSP directives are usually enclosed within the “<%@” and “%>” delimiters
- Page directive
  - <%@ page import = “java.util.*” %>
  - The page directive is dynamic at run time
  - It defines page properties such as content type, session type
  - The page directive can use up to 13 attributes
    - Import, contentType, errorPage, etc
  - Import multiple classes/packages by separating them with a comma
    - <%@ page import = “java.util.*, foo.*” %>
- Include directive
  - <%@ include file = “hello_bupt.jsp” %>
  - The include directive is static
  - It adds the text and codes to the current page at translation time

JSP Directives

- The taglib directive
  - This defines the libraries available to JSP
  - <%@ taglib tagdir="/WEB-INF/tags/foo" prefix="foo" %>
- The attribute directive
  - The attribute directive is only used for tag files, nothing else
  - <%@ attribute name="<text>" required="[Boolean]" rtexprvalue="[Boolean]" %>
    - <%@ attribute name="bookTitle" required="true" rtexprvalue="true" %>
    - If required="true", then the attribute is not optional

JSP Expression

- JSP expression code uses the “<%=“ and “%>” delimiters
- JSP expressions are used to evaluate some actions or values
- Example

```
<%@ page import=“java.util.*” %>
<html>
<body>
Hello, welcome to BUPT! The time is now:
<%= new Date() %>
</body>
</html>
```

JSP Expression and out.print() Method

- The JSP expression is similar to the out.print() method in Java
  - The expression <%= new Date() %> is converted to out.print(new Date());
- Do not end your expression with a semi-colon
  - <%= new Date(); %> is wrong because it will be converted to
    - out.print(new Date(););

JSP Scriptlet

- When a JSP uses the “<%” and “%>” delimiters, we say it is using scriptlet
- Example

```
<html>
<body>
The date is:
<% out.print(new java.util.Date()); %>
</body>
</html>
```

JSP Declaration

- To declare variables and objects, JSP uses the “<%!” and “%>” delimiters
- Example

```
<%@ page import="java.util.*" %>
<html>
<body>
<%! Date myDate = new Date();
Date getDate()    {
            System.out.println( "getDate() method" );
            return myDate;
  }
%>
Hello, welcome to BUPT!  The time is now <%= getDate() %>
</body>
</html>
```

JSP Actions

- JSP actions do not use the scriplet delimiters, “<%” and “%>”
- There are two types of JSP actions
  - Standard and custom actions
- The standard format for JSP action is
  - <jsp: tag />
- The custom action format is
  - <c: tag />
- An action can have a start tag, body and end tag
- An action may or may not have a body
- The examples given above <jsp: tag /> and <c: tag /> do not have bodies
- Example of standard action
  - <jsp: include page=“hello_bupt.jsp” />
- Example of custom action
  - <c: set var=“rate” value =“60” />

JSP Actions

- Other Examples

```
<html>
<body>
<jsp:include page="hello_bupt.jsp"
    <jsp:param name="myTitle" value="Welcome to BUPT" />
</jsp:include>
</body>
</html>

<jsp:forward page="hello_bupt.jsp" />
```

JSP Expression Language (EL)

- The JSP Expression Language (EL) does not use the scripting delimiters such as “<% and “%>”
- EL always start with a $ (dollar sign) followed by an open curly brace (“{“) and the followed by an object or an attribute and then finally closed by a closing curly brace (“}”).
  - Examples
    - ${person[“name”]}
    - ${person.name}
- The above 2 examples will print the same output
- The first variable before the period “.” must be an implicit object or an attribute

EL Implicit Objects

- Map Objects
  - pageScope
    - Define within the page scope
  - requestScope
    - Defined within the request scope
  - sessionScope
    - Defined within the request scope
  - applicationScope
    - Defined within the entire application scope
  - param
  - paramValues
  - Header
  - headerValues
  - Cookie
  - initParam
- pageContext Object
  - pageContext
- All the implicit objects are Maps except the pageContext

Using the dot (.) Operator in EL

- ${first.second}
  - The first variable (first) must be a map or a bean
  - The second variable (second) must be a map key or a bean property
  - The second variable (second) must comply with the normal Java naming rules for variables and identifiers
    - Must start with a letter (a-z, A-Z), or an underscore (_) or a dollar sign ($)
    - After the first character, numerical numbers (1, 2, 3,100, etc) can be added
    - It cannot be a Java keyword such as new, print, out, etc)

Using the square brackets [] Operator in EL

- ${first["second"]}
  - It is true for all that has been explained for the dot operator
    - Works on map or bean as the first variable and and map key or bean property for the second variable respectively
  - In addition, the [] operator works for Arrays and List objects
  - Also works for some literals as the argument in the [] operator

Other EL Operators

- Arithmetic operators
  - + (addition), - (subtraction), * (multiplication), / and div (division) % and mod (remainder)
- Relational operators
  - == and eq (equality), != and ne (not equals), < and lt (less than), > and gt (greater than), <= and le (less than or equal to), >= and ge (greater than or equal to)
- Logical operators
  - && and and (AND), II and or (OR), ! And not (NOT)

The Graceful Nature of EL

- It is good to handle some wrong inputs gracefully without always displaying errors to users on the browser
- EL handles “null” values by not printing them at all
- For example ${man} will print nothing
- ${89 + man} will print 89
- ${89/man} will print infinity
- {89 % man} will throw exception (error)
- ${90 == man} will print false

JSP and Bean

- JSP actions or tags are used with Bean to process forms
- JavaBeans are components that are used as data transfer objects in JSP
- The JavaBean classes implements Serializeable interface
- Example JSP and Bean standard actions

```
<jsp:useBean id="<text>" class="<class>" scope="<scope>"
/>


<jsp:useBean id="student" class="department.Student"
scope="session" />
```

JSP and Bean

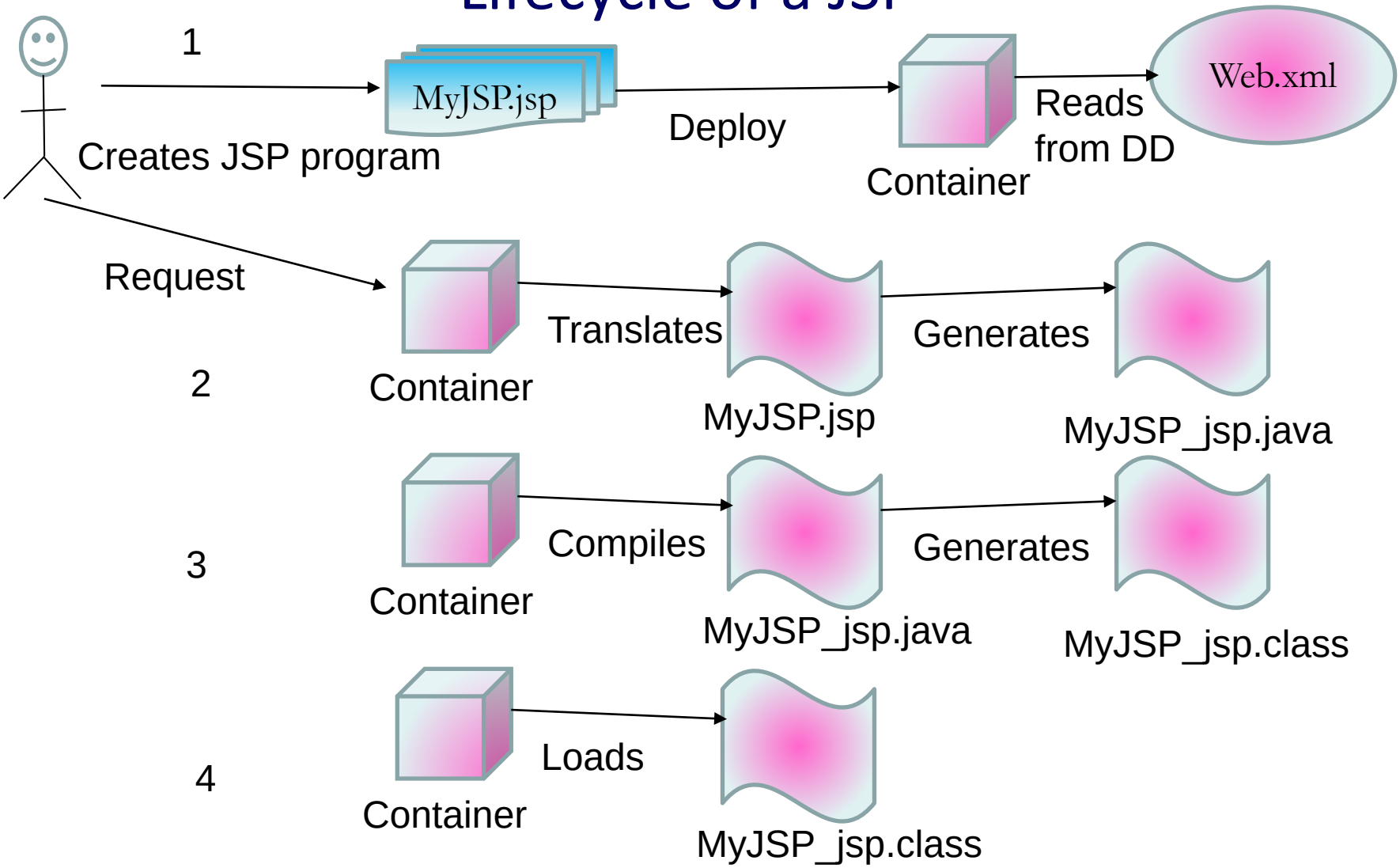
- You can get the property of a Bean

  <jsp:getProperty name="student" property="name" />

- You can set the property of a Bean

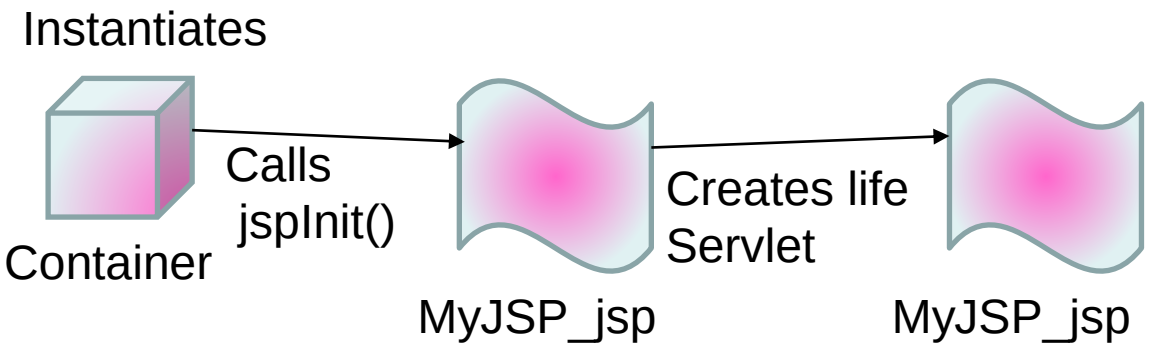
```
<jsp:useBean id="student" class="department.Student" scope="session"
   <jsp:setProperty name="student" property="Grade"  value="90%" />
</jsp:useBean>
```

Lifecycle of a JSP

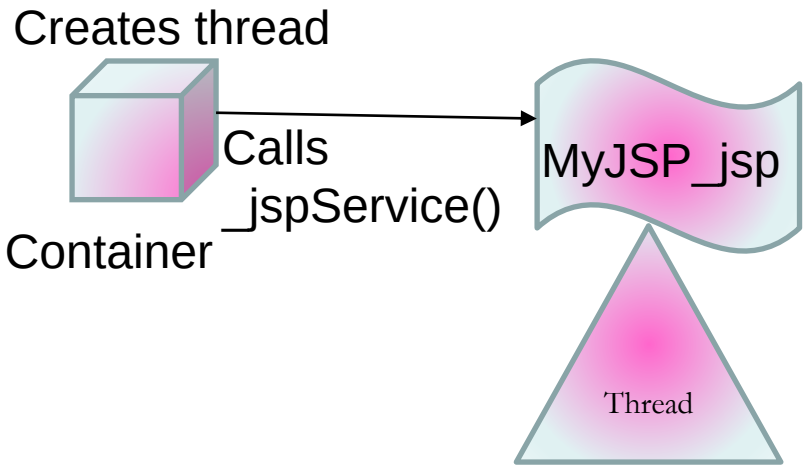


Lifecycle of a JSP

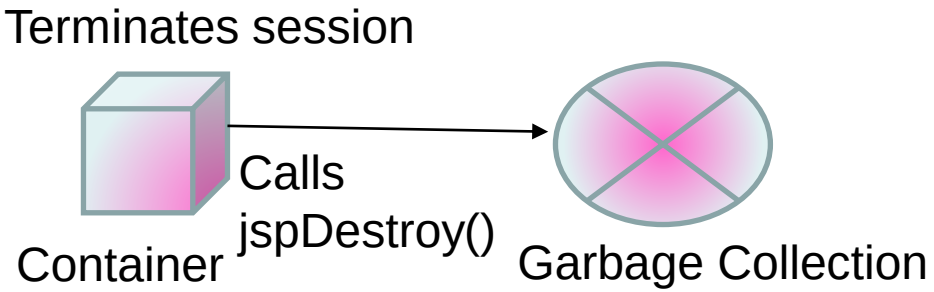
5



6



7



Quiz Questions

- Decide if the following are valid or not, giving your reasons for each answer
  - <%= 50 %>
  - <%= Math.random()+10; %>
  - <%= “100” %>
  - <%= new String[10] %>
  - <%= 10>100 %>
  - <%= 50*2 %>
  - <%= foo %>
  - <% = true %>