JAVA DAVE ENVIRONMENT

Spring Framework, JSP, Web Container & More…

Java Boot Camp, August, 2017.

Dr. Kishore Biswas (Forrest/柯修)

PhD. Artificial Intelligence—Natural Language Processing.

CanWay IT Training ®.

CONTENTS

1) Spring Framework Example & Practice

Spring Environment Setup…

1) Installing Apache Common Logging API

- You can download the latest version of Apache Commons Logging API from https://commons.apache.org/logging/. Once you download the installation, unpack the binary distribution into a convenient location.

2) Setup Spring Framework Libraries

- Now if everything is fine, then you can proceed to set up your Spring framework. Following are the simple steps to download and install the framework on your machine.
- Make a choice whether you want to install Spring on Windows or Unix, and then proceed to the next step to download .zip file for Windows and .tz file for Unix.
- Download the latest version of Spring framework binaries from https://repo.spring.io/release/org/springframework/spring.
- At the time of developing this tutorial, **spring-framework-4.1.6.RELEASE-dist.zip** was downloaded on Windows machine. After the downloaded file was unzipped, it gives the following directory structure inside E:\spring.

Spring Environment Setup

3) Creating Java Project

4) Adding Required Libraries

Add Spring Framework and common logging API libraries in our project. To do this, right-click on your project name, then follow the following option available in the context menu − **Build Path → Configure Build Path** to display the Java Build Path window. Add External JARs button available under the Libraries tab to add the following core JARs from Spring Framework and Common Logging installation directories

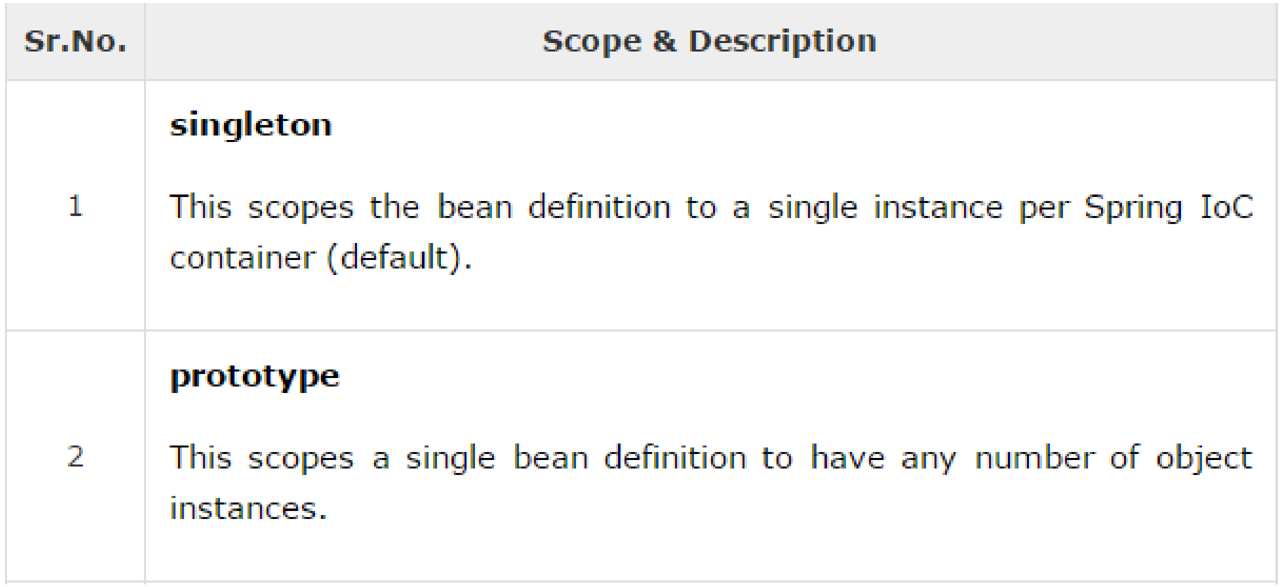
5)Creating Source Files

6) Creating Bean Configuration File

7) Running the Program

SPRING - BEAN SCOPES CODE EXAMPLE

- When defining a <bean> you have the option of declaring a scope for that bean. For example, to force Spring to produce a new bean instance each time one is needed, you should declare the bean's scope attribute to be prototype. Similarly, if you want Spring to return the same bean instance each time one is needed, you should declare the bean's scope attribute to be singleton.



Sr.No.	Scope & Description
1	**singleton** This scopes the bean definition to a single instance per Spring IoC container (default).
2	**prototype** This scopes a single bean definition to have any number of object instances.

```
<!-- A bean definition with singleton scope -->
<bean id = "..." class = "..." scope = "singleton">
  <!-- collaborators and configuration for this
bean go here -->
</bean>
```

```
<!-- A bean definition with singleton scope -->
<bean id = "..." class = "..." scope =
"prototype">
  <!-- collaborators and configuration for this
bean go here -->
</bean>
```

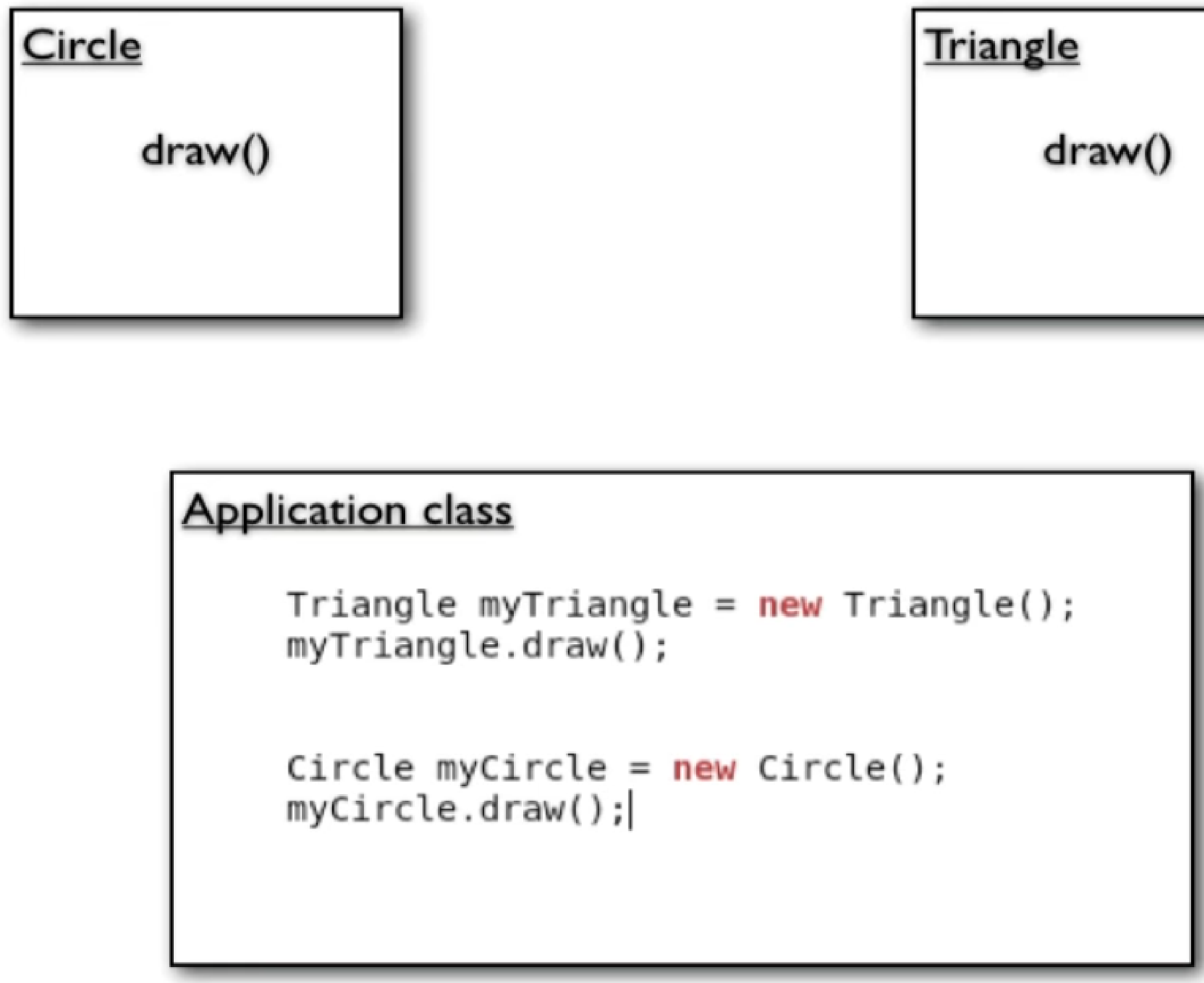
Spring - Bean Life Cycle Code Example

- The life cycle of a Spring bean is easy to understand. When a bean is instantiated, it may be required to perform some initialization to get it into a usable state. Similarly, when the bean is no longer required and is removed from the container, some cleanup may be required.

- Though, there are lists of the activities that take place behind the scene between the time of bean Instantiation and its destruction.

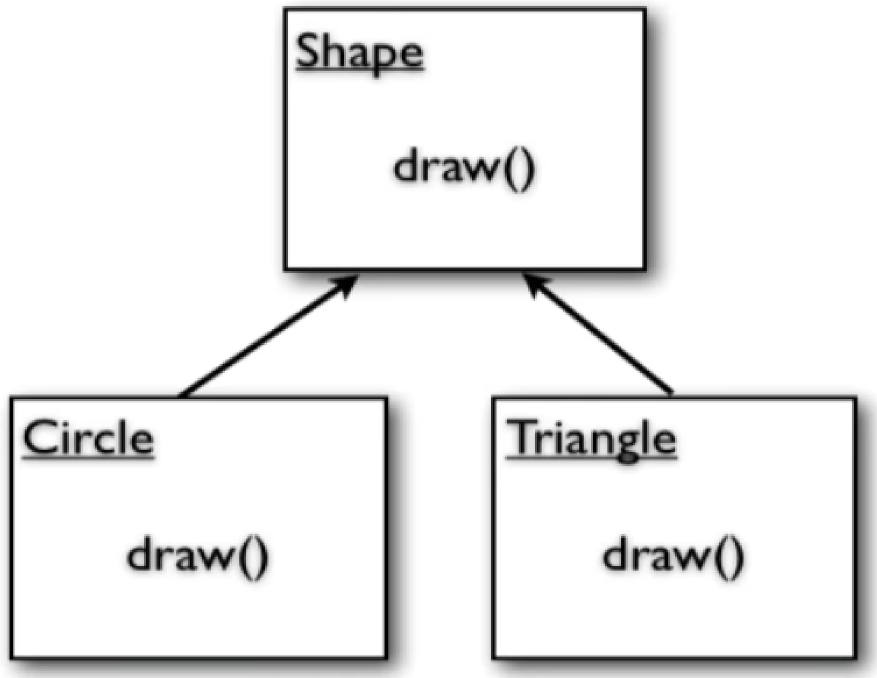
- To define setup and teardown for a bean, we simply declare the <bean> with initmethod and/or destroy-method parameters. The init-method attribute specifies a method that is to be called on the bean immediately upon instantiation. Similarly, destroymethod specifies a method that is called just before a bean is removed from the container.

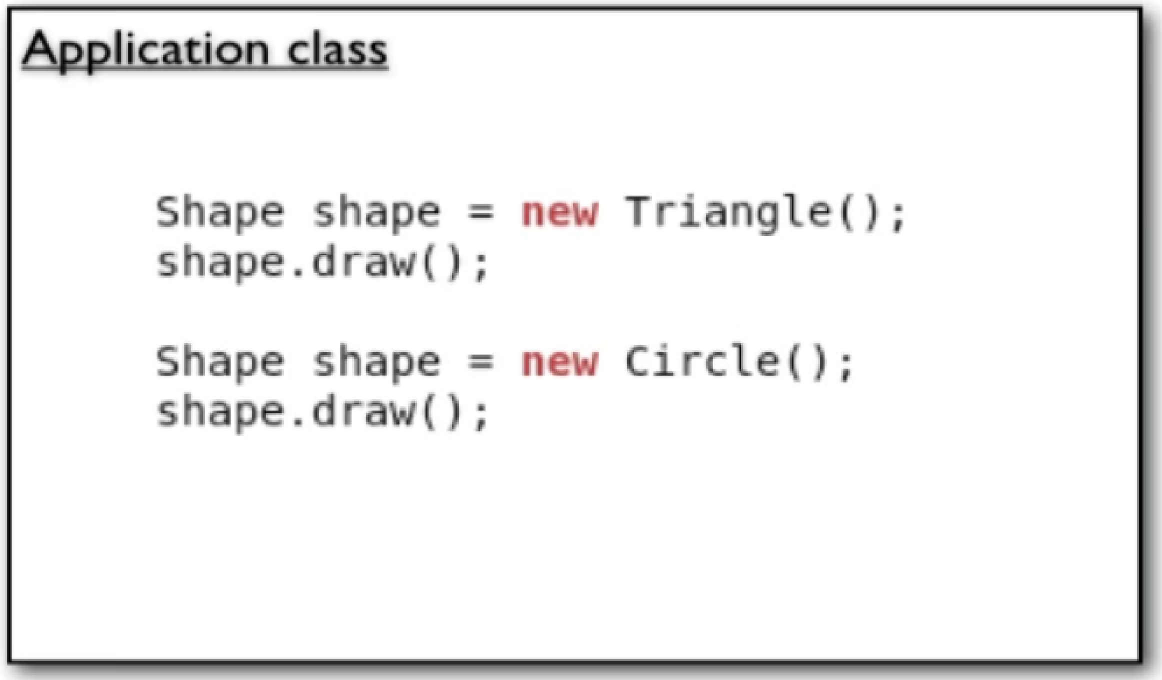
DEPENDENCY EXPLAINED...



```
Triangle myTriangle = new Triangle();
myTriangle.draw();


Circle myCircle = new Circle();
myCircle.draw();
```

Dependency Explained…

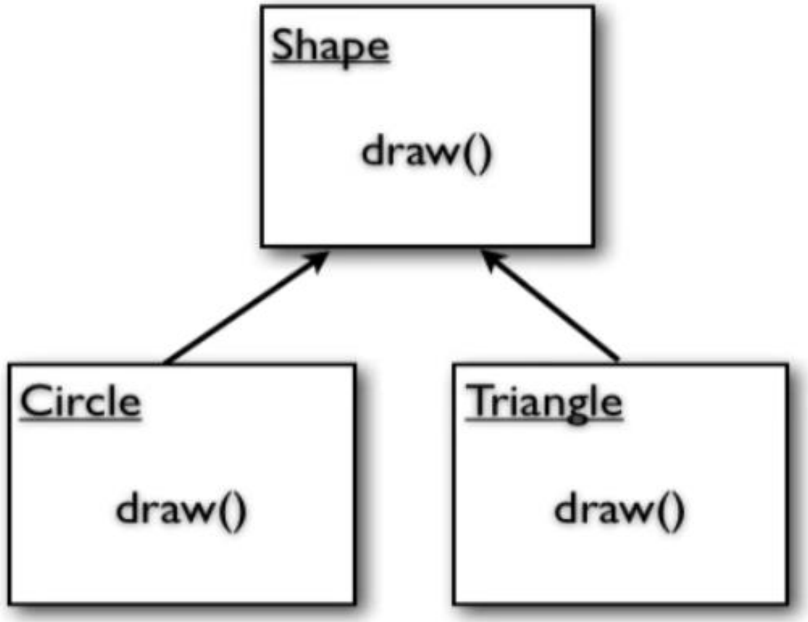


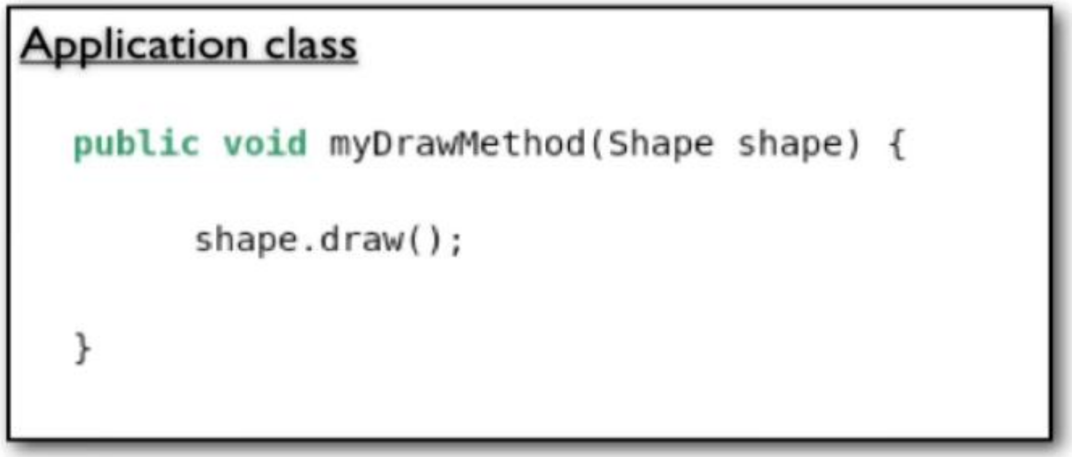

Application class

```
Shape shape = new Triangle();
shape.draw();

Shape shape = new Circle();
shape.draw();
```

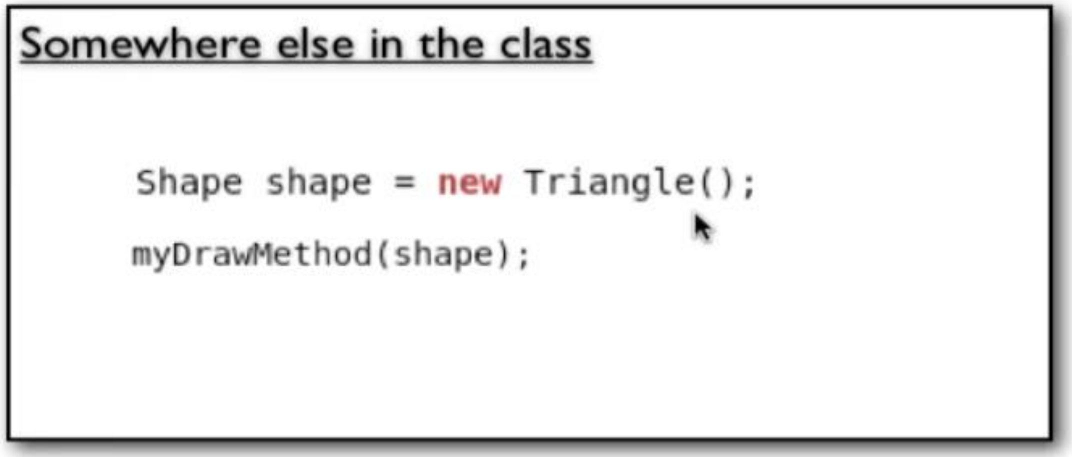
Using Polymorphism approach!

DEPENDENCY EXPLAINED…




Application class

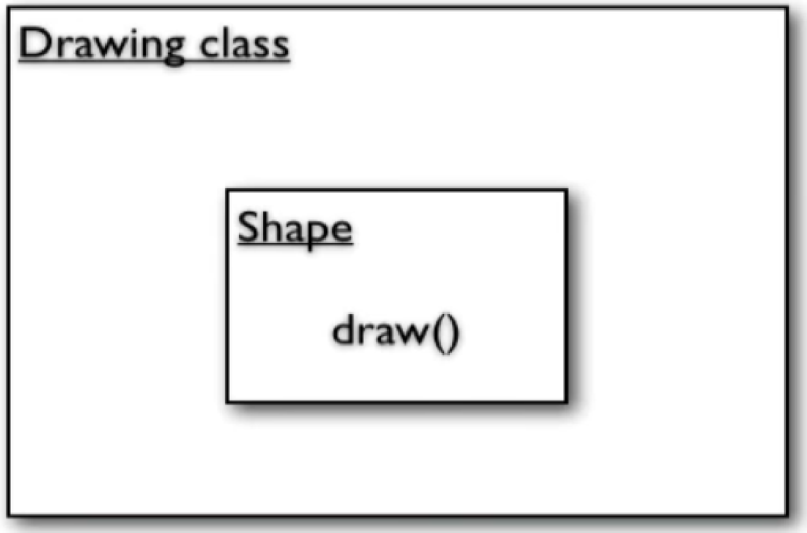
```
public void myDrawMethod(Shape shape) {

        shape.draw();

}
```

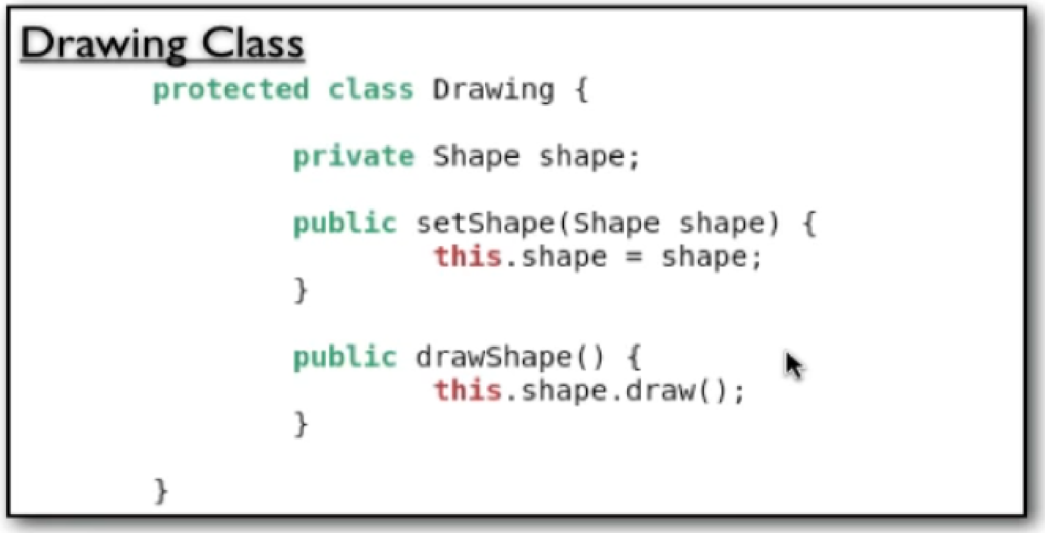

Somewhere else in the class

```
   Shape shape = new Triangle();
  myDrawMethod(shape);
```

A step further…

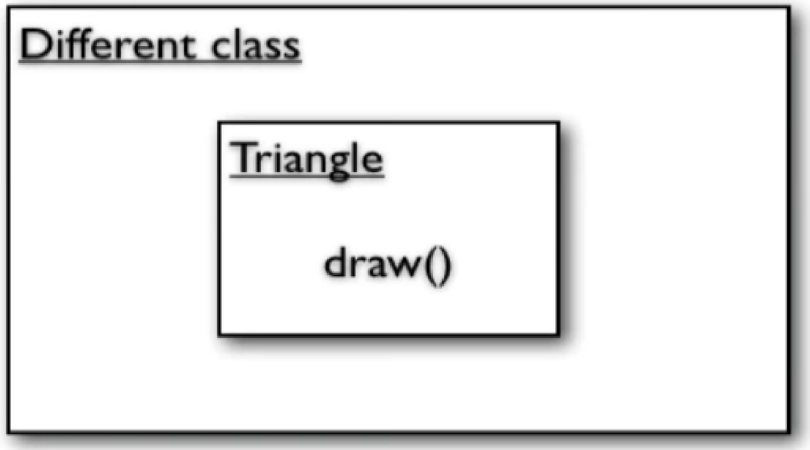
DEPENDENCY EXPLAINED

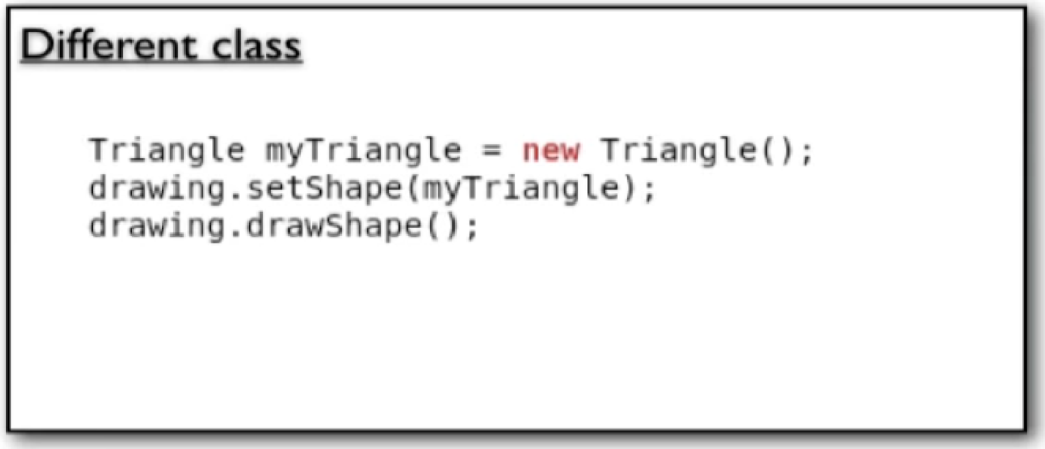




Drawing Class

```
protected class Drawing {

        private Shape shape;

        public setShape(Shape shape) {
                this.shape = shape;
        }

        public drawShape() {
                this.shape.draw();
        }

}
```





Different class

```
Triangle myTriangle = new Triangle();
drawing.setShape(myTriangle);
drawing.drawShape();
```

What is a Web Framework?

- A web framework is a software framework designed to simplify your web development life.
- Frameworks exist to save you from having to re-invent the wheel and help alleviate some of the overhead when you’re building a new site.

What is a Web Framework?

- Typically frameworks provide libraries for accessing a database, managing sessions and cookies, creating templates to display your HTML and in general, promote the reuse of code.

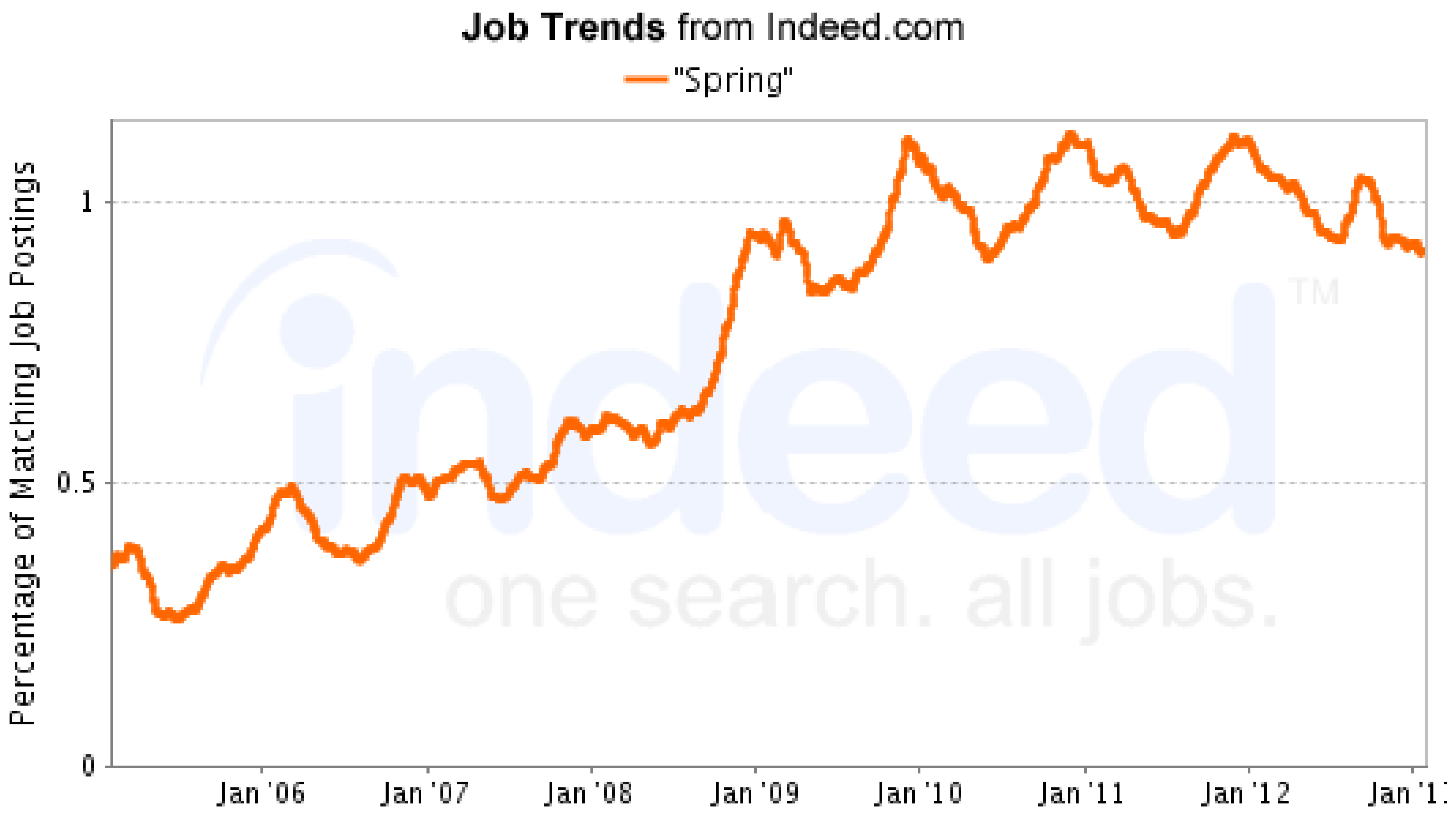
Spring Framework

- Open source application framework.
- Inversion of Control container (IoC).
- Lots of utility API-s.

Should I bother?

- Spring framework is still extremely popular in Java (web) applications.
- If you’re going to build web applications in Java then chances are that you will meet Spring.
- VMWare bought Spring with some 412 million usd!

Should I bother?



IoC in Spring

```
@Service
public class UserService {

	@Resource
	private UserDao userDao;

	public User getByUserName(String name) {
		User user = userDao.getByUserName(name);
		// additional actions
		return user;
	}
}
```

IoC in Spring

Declare bean, Spring will create it

```
@Service
public class UserService {

        @Resource
        private UserDao userDao;

        public User getByUserName(String name) {
                User user = userDao.getByUserName(name);
                // additional actions
                return user;
        }
}
```

IoC in Spring

```
@Service
public class UserService {

        @Resource
        private UserDao userDao;

        public User getByUserName(String name) {
                User user = userDao.getByUserName(name);
                // additional actions
                return user;
        }
}
```

Declare dependencies, Spring will inject them

IoC in Spring

- Spring handles the infrastructure (bean creation, dependency lookup and injection)
- Developer focus on application specific logic.

Dependency injection

```
public interface UserDao {
        User getByUserName(String name);
}

@Repository
public class JdbcUserDao implements UserDao {
        public User getByUserName(String name) {
                // load user from DB
        }
}

@Resource
private UserDao userDao;
```

Dependency injection

```
public interface UserDao {
        User getByUserName(String name);
}

@Repository
public class JdbcUserDao implements UserDao {
        public User getByUserName(String name) {
                // load user from DB
        }
}

@Resource
private UserDao userDao;
```

Universal abstraction

Dependency injection

```
public interface UserDao {
        User getByUserName(String name);
}

@Repository
public class JdbcUserDao implements UserDao {
        public User getByUserName(String name) {
                // load user from DB
        }
}

@Resource
private UserDao userDao;
```

One possible implementation. Spring will create and register it

Dependency injection

```
public interface UserDao {
        User getByUserName(String name);
}

@Repository
public class JdbcUserDao implements UserDao {
        public User getByUserName(String name) {
                // load user from DB
        }
}

@Resource
private UserDao userDao;
```

Spring can inject as an abstraction type

XML based configuration

- Bean can also be defined and injected in XML.

```
<bean id="userDao" class="example.JdbcUserDao" />

<bean id="userService" class="example.UserService">
  <property name="userDao" ref="userDao" />
</bean>
```

Dependency injection

- Your code depends on abstractions, Spring handles actual implementations.
- You can switch implementations easily.

What is MVC?

- The Model View Controller (MVC) pattern is a way of organising an application (not necessarily a web application) so that different aspects of it are kept separate. This is a good thing because:

What is MVC?

- It is good software engineering practice to maintain separation of concerns.
- An application might have more than one user interface
- Different developers may be responsible for different aspects of the application.

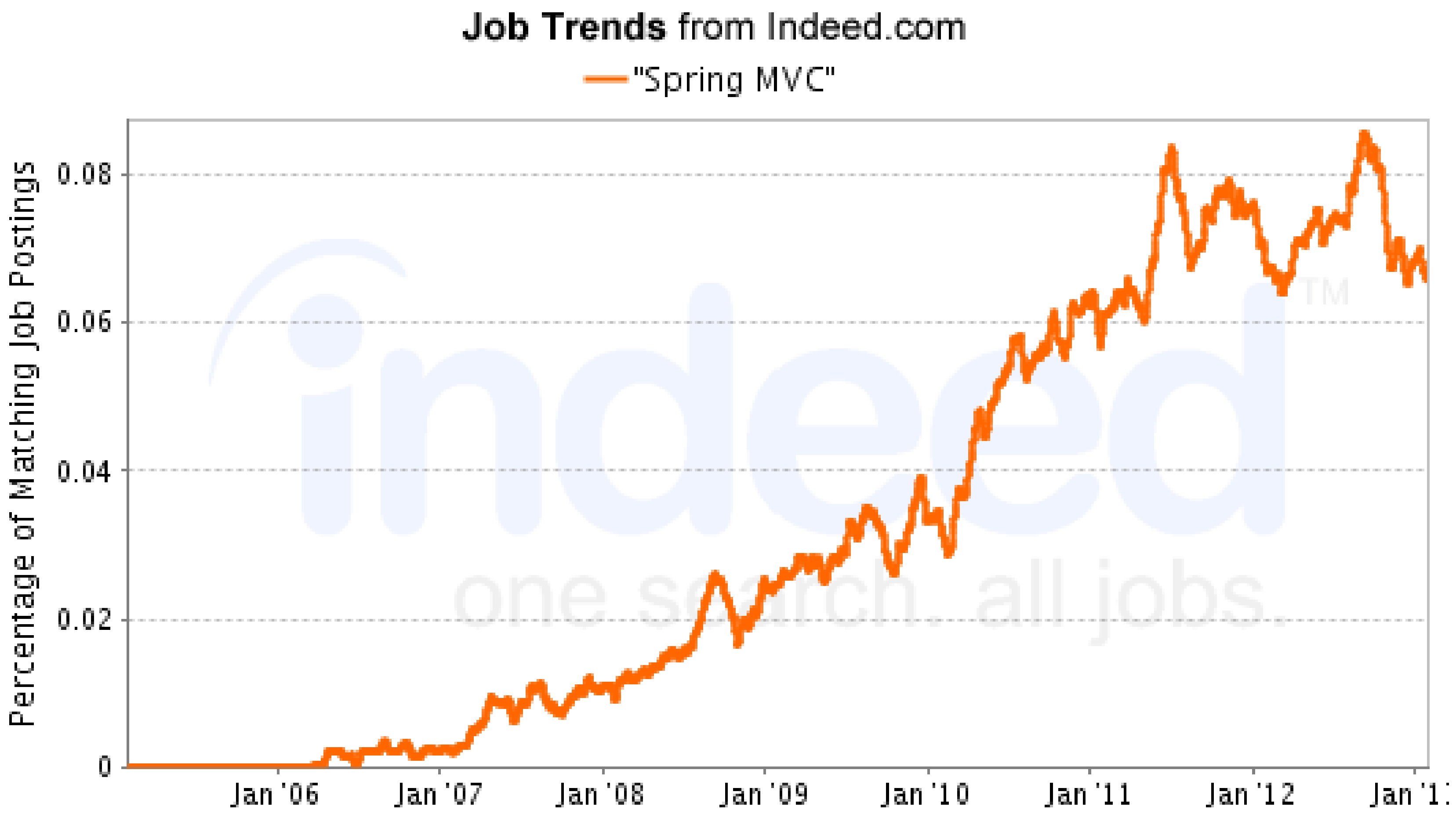
Spring MVC

- Spring based web framework.
- Implements the Model-View-Controller design pattern.
- Very flexible (we’ll see how exactly).

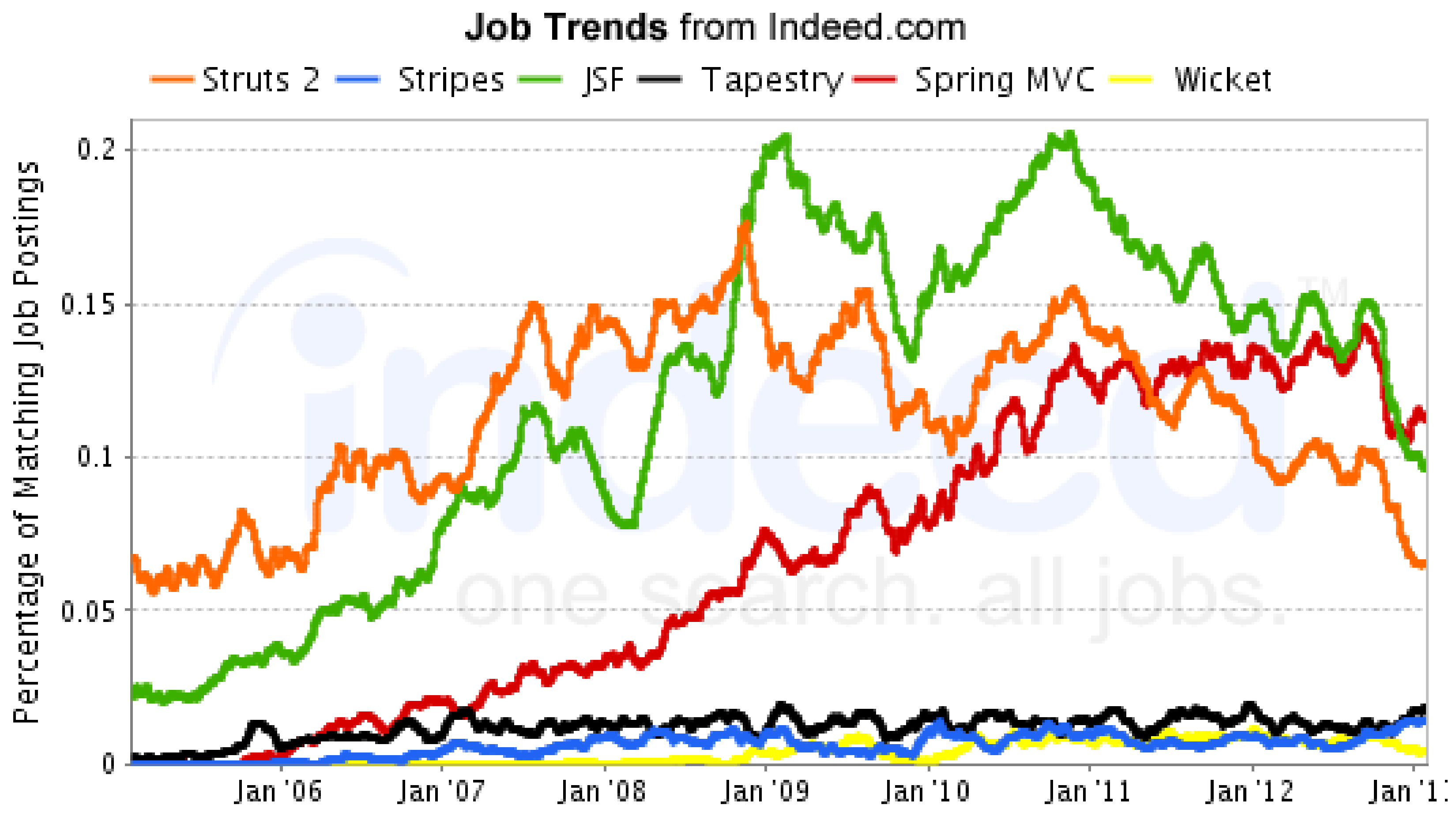
Spring MVC

- IoC again – framework handles the infrastructure, you focus on application specific things.

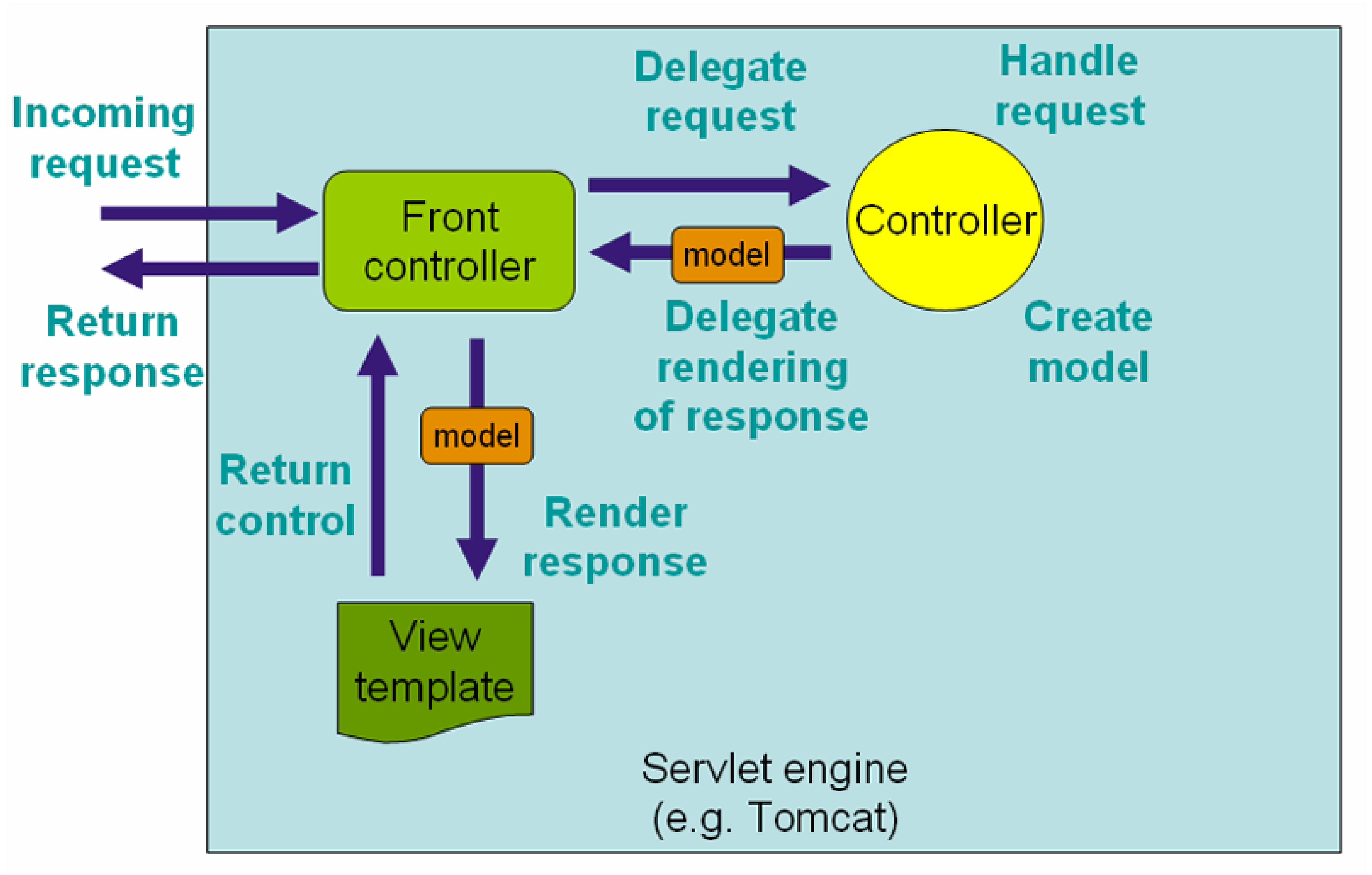
Should I bother?



Should I bother?



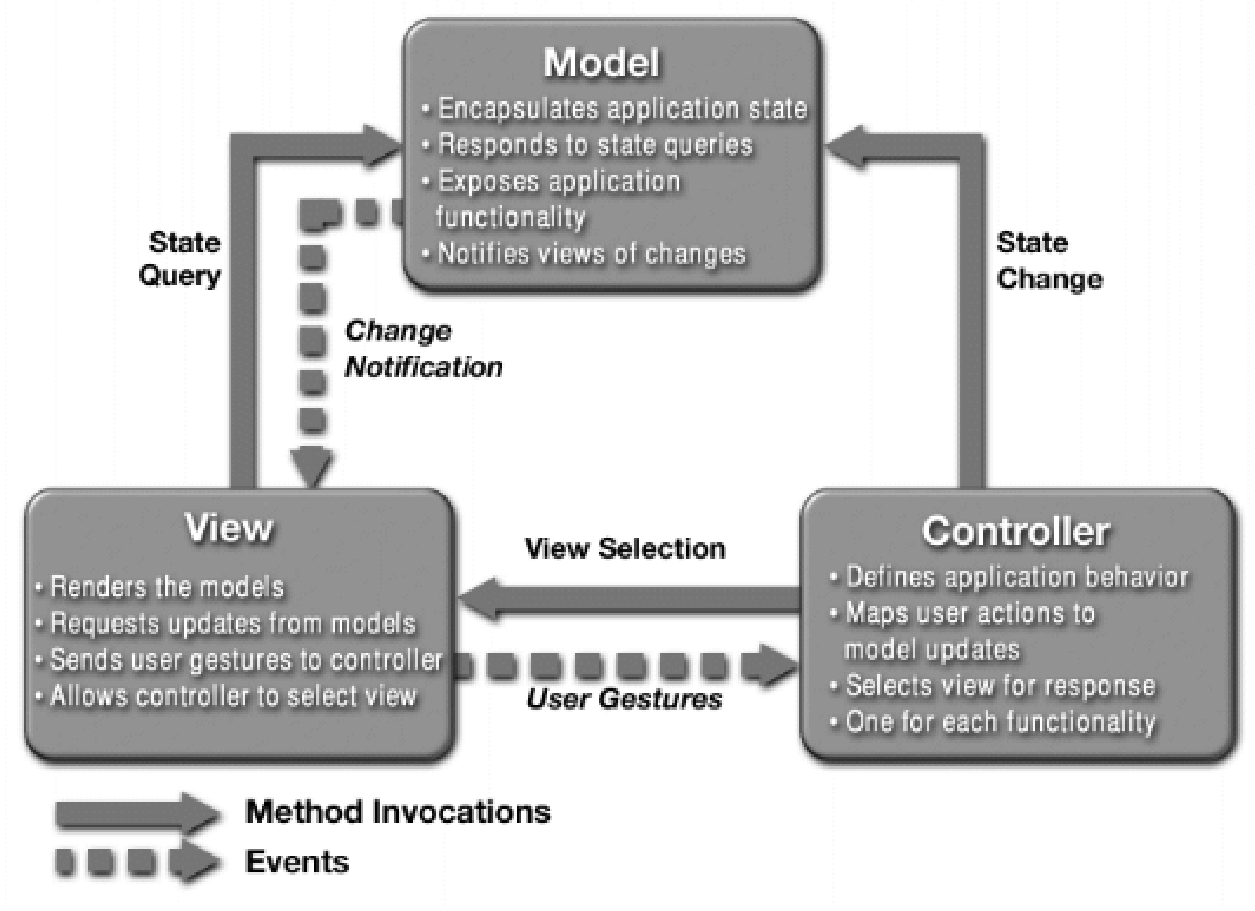
Architecture - DispatcherServlet



Recall: Model-View-Controller

- Model - The model represents enterprise data and the business rules that govern access to and updates of this data.
- View - The view renders the contents of a model.
- Controller - The controller translates interactions with the view into actions to be performed by the model.

Recall: Model-View-Controller


Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
Change Notification
State Change
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

Controller

```
@Controller
public class HelloController {

	@Resource
	private UserService userService;

	@RequestMapping("/hello")
	public String hello(Model model) {
		User user = userService.getByUserName("Cartman");
		model.addAttribute("user", user);

		return "hello";
	}
}
```

Controller

Declare controller

```
@Controller
public class HelloController {

        @Resource
        private UserService userService;

        @RequestMapping("/hello")
        public String hello(Model model) {
                User user = userService.getByUserName("Cartman");
                model.addAttribute("user", user);

                return "hello";
        }
}
```

Controller

```
@Controller
public class HelloController {

        @Resource
        private UserService userService;

        @RequestMapping("/hello")
        public String hello(Model model) {
                User user = userService.getByUserName("Cartman");
                model.addAttribute("user", user);

                return "hello";
        }
}
```

Inject Spring resources

Controller

```
@Controller
public class HelloController {

        @Resource
        private UserService userService;

        @RequestMapping("/hello")
        public String hello(Model model) {
                User user = userService.getByUserName("Cartman");
                model.addAttribute("user", user);

                return "hello";
        }
}
```

Method for handling requests

Controller

```java
@Controller
public class HelloController {

        @Resource
        private UserService userService;

        @RequestMapping("/hello")
        public String hello(Model model) {
                User user = userService.getByUserName("Cartman");
                model.addAttribute("user", user);

                return "hello";
        }
}
```

What requests to serve?

Controller

```
@Controller
public class HelloController {

        @Resource
        private UserService userService;

        @RequestMapping("/hello")
        public String hello(Model model) {
                User user = userService.getByUserName("Cartman");
                model.addAttribute("user", user);

                return "hello";
        }
}
```

Prepare model data

Controller

```
@Controller
public class HelloController {

        @Resource
        private UserService userService;

        @RequestMapping("/hello")
        public String hello(Model model) {
                User user = userService.getByUserName("Cartman");
                model.addAttribute("user", user);

                return "hello";
        }
}
```

Logical view name

Model

- Set of attributes that Controller collects and passes to the View.

Model

- Spring’s Model object…

```
@RequestMapping(value="/hello")
public String hello(Model model) {
   User user = userService.getByUserName(“cartman");
   model.addAttribute("user", user);
   ...
```

Model

- Spring’s Model object…

```
@RequestMapping(value="/hello")
public String hello(Model model) {
   User user = userService.getByUserName(“cartman");
   model.addAttribute("user", user);
   ...
```

Model

- Or plain java.util.Map

```
@RequestMapping(value="/hello")
public String hello(Map<String, Object> model) {
   User user = userService.getByUserName("cartman");
   model.put("user", user);
   ...
```

Model

- Or plain java.util.Map

```
@RequestMapping(value="/hello")
public String hello(Map<String, Object> model) {
   User user = userService.getByUserName("cartman");
   model.put("user", user);
   ...
```

View

- Any representation of output, invoked after the Controller, uses data from the Model to render itself.

View technologies

- Usually Java Server Pages that generate HTML.
- Out-of-the-box there are also PDF, XML, JSON, Excel and other views.
- You can create your own.

View

- Controller is totally decoupled from actual view technology.

```
@RequestMapping("/hello")
public String hello() {
  ...
  return "hello";
}
```

Just a logical view name to be invoked

JSP view

```
<bean class="org.springframework...InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
</bean>
```

$\longrightarrow$

"hello” */WEB-INF/jsp/hello.jsp*

JSP view (hello.jsp)

- Model attributes are accessible as EL (Expression Language) variables in JSP.

JSP: `<p>Hello, ${user.fullName}!</p>`

HTML: `<p>Hello, Eric Cartman!</p>`

JSON view

- JSON view transforms the whole model to JSON format.

```
<bean class="org.springframework...ContentNegotiatingViewResolver">
    <property name="defaultViews">
        <list>
            <bean class="org.springframework...MappingJacksonJsonView" />
        </list>
  </property>
</bean>
```

JSON view

- Outputs the Model as JSON document.

```
{"user":{"fullName":"Eric Cartman"}}
```

Request mapping

```
@RequestMapping("/hello")

@RequestMapping(value="/hello",
     method=RequestMethod.GET)

@RequestMapping(value="/hello", params=
{"param1", "param2"})

@RequestMapping(value="/hello",
     consumes="application/json",
     produces="application/json")
```

Path variables

```
@RequestMapping(value="/hello/{username}")
public String hello(
     @PathVariable String username,
     Model model) {
  ...
```

http://[SERVER]/hello/cartman

Path variables

```
@RequestMapping(value="/hello/{username}")
public String hello(
         @PathVariable String username,
         Model model) {
    ...
```

http://[SERVER]/hello/cartman

Request parameters

```
@RequestMapping(value="/hello")
public String hello(
  @RequestParam("username") String username,
  Model model) {
      ...
```

http://[SERVER]/hello?username=cartman

Type conversion

- HttpServletRequest parameters, headers, paths etc are all Strings.
- Spring MVC allows you to convert to and from Strings automatically.

Built-in conversion

- There are some standard built-in converters.

```
@RequestMapping("/foo")
public String foo(
     @RequestParam("param1") int intParam,
     @RequestParam("param2") long longParam) {
  ...
```

Type conversion

- You can also define your own PropertyEditors
- *PropertyEditorSupport* ***implements PropertyEditor***

Custom types

```
public class DateRange {

        private Date start;
        private Date end;

        public DateRange(Date start, Date end) {
                this.start = start;
                this.end = end;
        }

        public int getDayDifference() {
                // calculate
        }
}
```

Custom type editor

```
public class DateRangeEditor extends PropertyEditorSupport {

          private DateFormat dateFormat = new SimpleDateFormat("dd.MM.yyyy");

          public void setAsText(String text) throws IllegalArgumentException {
                    String[] parts = text.split("-");

                    Date start = dateFormat.parse(parts[0]);
                    Date end = dateFormat.parse(parts[1]);

                    setValue(new DateRange(start, end));
          }

          public String getAsText() {
                    DateRange dateRange = (DateRange) getValue();
                    return dateFormat.format(dateRange.getStart()) + "-" +
                              dateFormat.format(dateRange.getEnd());
          }
}
```

String to custom type

Custom type to String

Register and use

```
@Controller
public class MyController {

   @InitBinder
   public void initBinder(WebDataBinder binder) {
      binder.registerCustomEditor(DateRange.class,
         new DateRangeEditor());
   }

   @RequestMapping(value="/dateRange")
   public String dateRange(@RequestParam("range") DateRange range) {
               ...
   }
}
```

Method parameters

- It all about IoC and Convention Over Configuration.
- Your code simply declares what it needs from the environment, Spring makes it happen.
- The order is not important, the type is.

Method parameters

- Model/Map – model
- @RequestParam/@PathVariable annotated
- HttpServletRequest, HttpServletResponse, HttpSession – it is all based on Servlet API!

Method parameters

- java.io.Writer / java.io.OutputStream – if you want to generate response directly in controller
- …

Method parameters

- For example

```
@RequestMapping(value="/hello")
public String hello(Model model,
  Writer writer, HttpServletRequest request,
  HttpSession session) {
...
```

Return values

- Same principle – you return what Spring awaits from you.

Return value examples

- String – logical view name.
- void
  - If you write the response in controller
  - If you use default/content negotiating views (like JSON earlier)
- …

Non-intrusive

- Very important feature of a framework is non-intrusiveness.
- Spring MVC normally lets you do stuff according to MVC pattern.
- But it doesn’t prevent you from violating MVC if you really want to.

No view examples

```
@RequestMapping(value="/noView")
@ResponseBody
public String noView() {
      return "Too simple for a view";
}

@RequestMapping(value="/noView")
public void noView(Writer writer) {
      writer.write("Too simple for a view");
}
```

Form

- First you need to give Spring a new command object

```
@RequestMapping(value="/addUser")
public String add(Model model) {
  model.addAttribute("user", new User());
  return "add";
}
```

Command object

```
public class User {

    private String fullName;
    private int age;

    // getters/setters
...
```

Form JSP

```
<%@ taglib prefix="form"
uri="http://www.springframework.org/tags/form"%>

<form:form commandName="user">
        <form:input path="fullName" />
        ...
        <form:input path="age" />
        ...
        <input type="submit" value="Add" />
</form:form>
```

Form

- Spring will bind the data to our command object

```
@RequestMapping(value="/addUser", method=RequestMethod.POST)
public String save(User user) {
   // save user

   return "home";
}
```

Validation

- JSR-303 defines constraint annotations:

```
public class User {

   @NotNull
   @Size(max=20)
   private String fullName;

   @Min(10)
   private int age;
...
```

Validation

```
@RequestMapping(value="/addUser", method=RequestMethod.POST)
public String save(@Valid User user, BindingResult result,
              Model model) {

       if (result.hasErrors()) {
              model.addAttribute("user", user);
              return "add";
       }

       // save user
       return "home";
}
```

Validation

```
@RequestMapping(value="/addUser", method=RequestMethod.POST)
public String save(@Valid User user, BindingResult result,
              Model model) {

       if (result.hasErrors()) {
              model.addAttribute("user", user);
              return "add";
       }

       // save user
       return "home";
}
```

Check for validation errors.

Render the same view in case of errors

Validation

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>

<form:form commandName="user">
        <form:input path="fullName" />
        <form:errors path="fullName" />
        ...
        <form:input path="age" />
        <form:errors path="age" />
        ...
        <input type="submit" value="Add" />
</form:form>
```

Show validation errors

Validation

- It is of course possible to define your own constraints and validators.

2) JSP Code Example & Practice

JSP Environment Setup…

1) Setting up Web Server: Tomcat

- A number of Web Servers that support JavaServer Pages and Servlets development are available in the market. Some web servers can be downloaded for free and Tomcat is one of them.
- Apache Tomcat is an open source software implementation of the JavaServer Pages and Servlet technologies and can act as a standalone server for testing JSP and Servlets, and can be integrated with the Apache Web Server.
- Download the latest version of Tomcat from https://tomcat.apache.org/.
- Once you downloaded the installation, unpack the binary distribution into a convenient location. For example, in C:\apache-tomcat-5.5.29 on windows.

What is JSP?

- A technology that allows for the creation of dynamically generated web pages based on HTML, XML, or other document types.
- To deploy a JSP, a servlet container such as Apache Tomcat is required.
- JSP’s are translated into servlets at runtime

Syntax: Scriptlets

- A scriptlet can contain any number of Java language statements, variables or method declarations, or expressions that are valid in the page scripting language
- Scriptlet syntax:
    - <% code fragment %>

Syntax: Scriptlets

```
<html>
<head><title>Hello World</title></head>
<body>
Hello World!<br/>
<%
System.out.println("Your IP address is " +
request.getRemoteAddr());
%>
</body>
</html>
```

SYNTAX: DECLARATIONS

- A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.

```
<%! int i = 0; %>
<%! int a, b, c; %>
<%! Circle a = new Circle(2.0); %>
```

SYNTAX: EXPRESSIONS

- A JSP expression element contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file.
- You can use an expression within a line of text, whether or not it is tagged with HTML, in a JSP file.
- Expression syntax:
  - <%= expression %>

Syntax: Expressions

```
<html>
<head><title>A Comment Test</title></head>
<body>
<p>
   Today's date: <%= (new
java.util.Date()).toLocaleString()%>
</p>
</body>
</html>
```

DECISION MAKING STATEMENTS

```
<%! int day = 3; %>
<body>
<% if (day == 1 | day == 7) { %>
      <p> Today is weekend</p>
<% } else { %>
      <p> Today is not weekend</p>
<% } %>
</body>
```

Send Data from Servlet to JSP

- Given the following Search servlet:

```
public class Search extends HttpServlet
{
   protected void doPost(HttpServletRequest request,
          HttpServletResponse response) throws
ServletException, IOException
   {…}
}
```

SEND DATA FROM SERVLET TO JSP

- Within doPost, assume we received a search query from the HttpServletRequest request parameter.
  - String query = request.getParameter(“search_field");
    - “search_field” is the HTML element id of the input field of our search page
- Once we got the query, we can send it to our results page to display the wanted information
  - request.setAttribute("query", query); // Add the query to our request

    ```
    // Forward to results page
    String nextJSP = "/results.jsp";
    RequestDispatcher dispatcher = getServletContext()
        .getRequestDispatcher(nextJSP);
    dispatcher.forward(request, response);
    ```

Send Data from Servlet to JSP

- Access our query from results.jsp
  - <%! String query = request.getAttribute("query"); %>

```
<html>
<head><title>Results</title></head>
<body>
<p>
  Displaying Results For: <%= query%>
</p>
</body>
</html>
```

JSP SYNTAX : SCRIPTLET

- A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.
- Syntax:

  ```
  <% code fragment %>
  ```
- XML equivalent of the above syntax:

  ```
  <jsp:scriptlet>
      code fragment
  </jsp:scriptlet>
  ```
- Any text, HTML tags, or JSP elements you write must be outside the scriptlet.

JSP : Declarations

- A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.
- Syntax :

  <%! declaration; [declaration;]+ ... %>
- Example

  <%! int i = 0; %>

JSP : Expression

- A JSP expression element contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file.
- The expression element can contain any expression that is valid according to the Java Language Specification but you cannot use a semicolon to end an expression.
- Syntax

  <%= expression %>

JSP : COMMENTS

- JSP comment marks text or statements that the JSP container should ignore. A JSP comment is useful when you want to hide or "comment out" part of your JSP page.
- Syntax

  <%-- This is JSP comment --%>

  <!– this HTML comment -->

JSP : Actions

- JSP actions use constructs in XML syntax to control the behavior of the servlet engine. You can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugin.
- Syntax

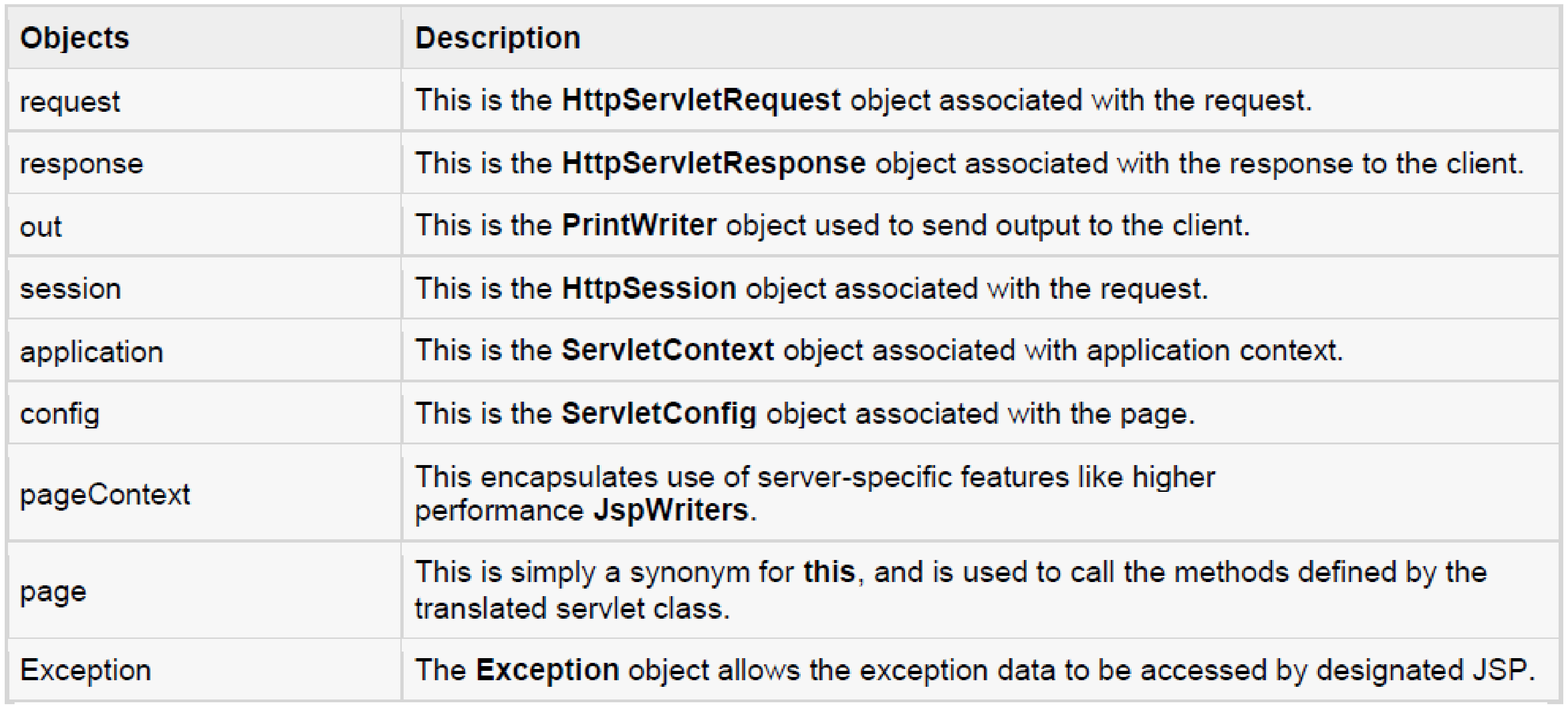
```
<jsp:action_name attribute="value" />
```

JSP : Actions List

Action	Purpose
jsp:include	Includes a file at the time the page is requested
jsp:useBean	Finds or instantiates a JavaBean
jsp:setProperty	Sets the property of a JavaBean
jsp:getProperty	Inserts the property of a JavaBean into the output
jsp:forward	Forwards the requester to a new page
jsp:plugin	Generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin
jsp:element	Defines XML elements dynamically.
jsp:attribute	Defines dynamically defined XML element's attribute.
jsp:body	Defines dynamically defined XML element's body.
jsp:text	Use to write template text in JSP pages and documents.

JSP : IMPLICIT OBJECTS

- JSP supports nine automatically defined variables, which are also called implicit objects.



Objects	Description
request	This is the **HttpServletRequest** object associated with the request.
response	This is the **HttpServletResponse** object associated with the response to the client.
out	This is the **PrintWriter** object used to send output to the client.
session	This is the **HttpSession** object associated with the request.
application	This is the **ServletContext** object associated with application context.
config	This is the **ServletConfig** object associated with the page.
pageContext	This encapsulates use of server-specific features like higher performance **JspWriters**.
page	This is simply a synonym for **this**, and is used to call the methods defined by the translated servlet class.
Exception	The **Exception** object allows the exception data to be accessed by designated JSP.

JSP : IF-ELSE STATEMENT

- Syntax:

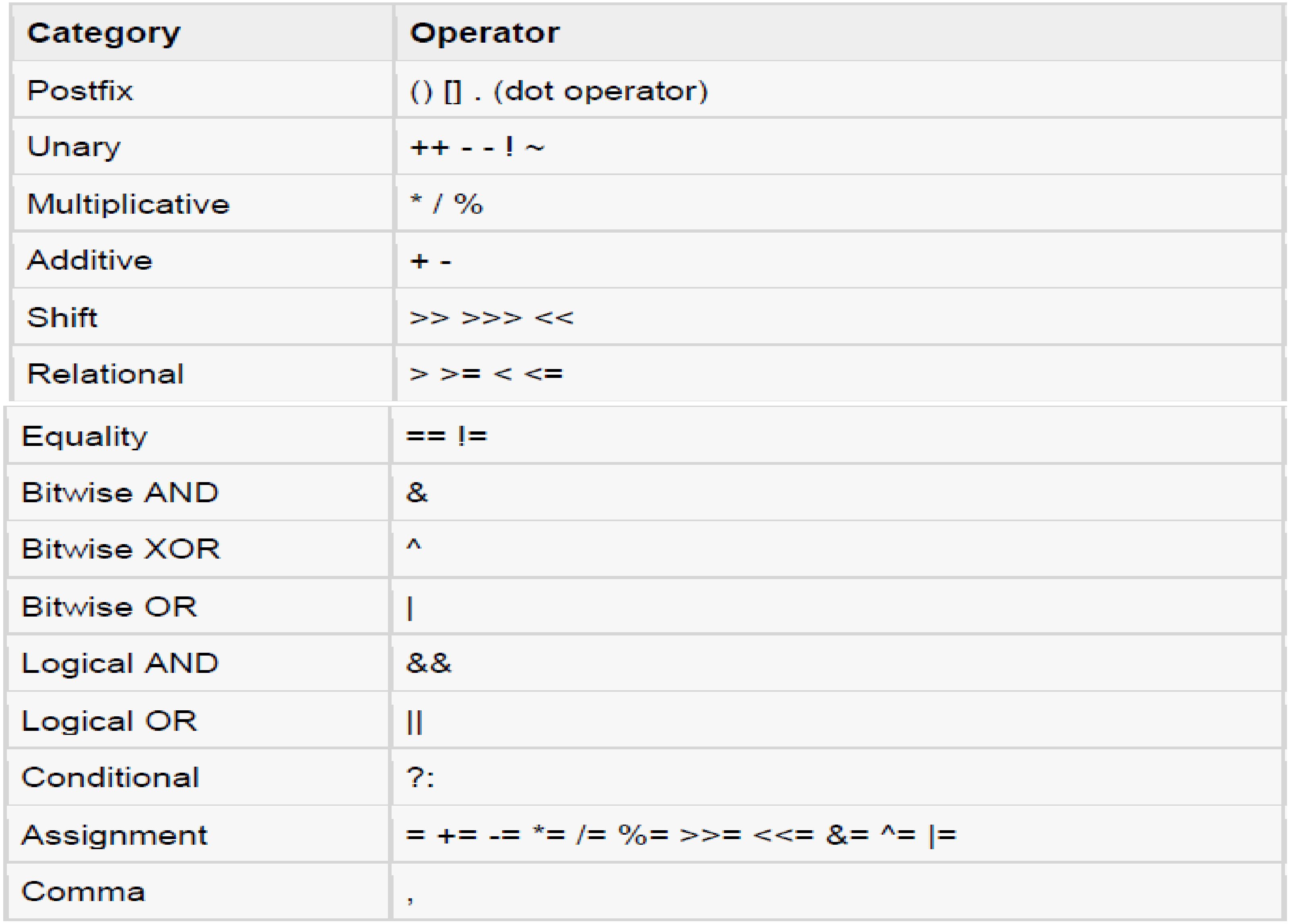
```
if(condition)
{
}
else if(condition)
{
}
else
{
}
```

JSP : FOR LOOP

- Syntax

```
<%! int loopVar; %>
<%
  for(initialize; condition; increDecre)
  {
        code
  }
%>
```

JSP : OPERATORS



Category	Operator
Postfix	() [] . (dot operator)
Unary	++ - - ! ~
Multiplicative	* / %
Additive	+ -
Shift	>> >>> <<
Relational	> >= < <=
Equality	== !=
Bitwise AND	&
Bitwise XOR	^
Bitwise OR	|
Logical AND	&&
Logical OR	||
Conditional	?:
Assignment	= += -= *= /= %= >>= <<= &= ^= |=
Comma	,

HOW TO INCLUDE ONE JSP FILE IN ANOTHER?

- Syntax

```
<jsp:include page="relative URL" flush="true" />
```

- flush : The boolean attribute determines whether the included resource has its buffer flushed before it is included.

Dynamic XML

- To create XML element

```
<jsp:element name="xmlElement">
…
</jsp:element>
```

- To set attribute of an XML element

```
<jsp:attribute name="xmlElementAttr">
   Value for the attribute
</jsp:attribute>
```

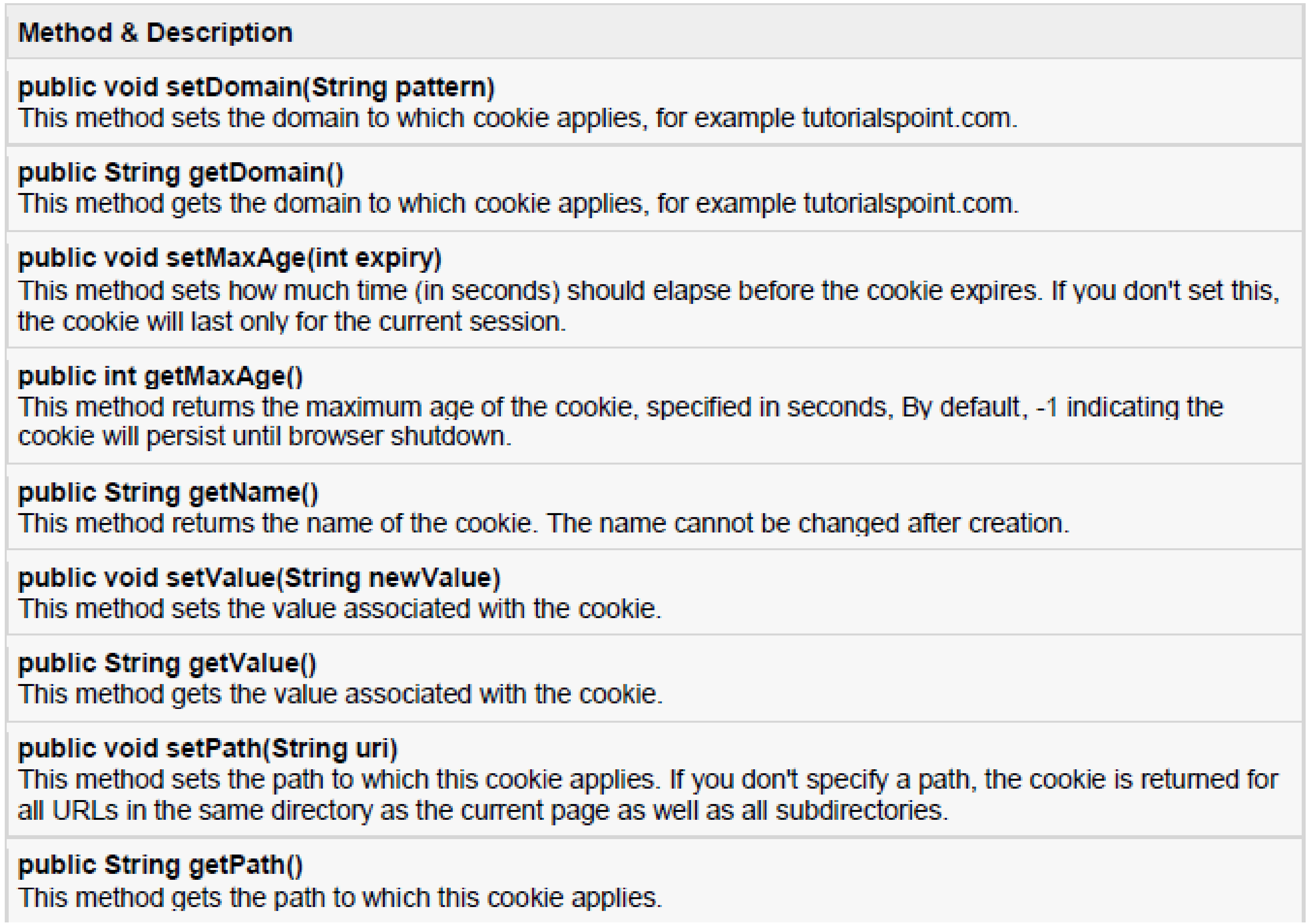
- To set body for XML element

```
<jsp:body>
   Body for XML element
</jsp:body>
```

JSP : Form Processing

- One can send data via GET or POST method.
- request.getParameter() method to get the value of a form parameter.
- getParameterValues() method if the parameter appears more than once and returns multiple values, for example checkbox.
- getParameterNames() method if you want a complete list of all parameters in the current request.

JSP : Servlet Cookies Methods



Method & Description
public void setDomain(String pattern) This method sets the domain to which cookie applies, for example tutorialspoint.com.
public String getDomain() This method gets the domain to which cookie applies, for example tutorialspoint.com.
public void setMaxAge(int expiry) This method sets how much time (in seconds) should elapse before the cookie expires. If you don't set this, the cookie will last only for the current session.
public int getMaxAge() This method returns the maximum age of the cookie, specified in seconds, By default, -1 indicating the cookie will persist until browser shutdown.
public String getName() This method returns the name of the cookie. The name cannot be changed after creation.
public void setValue(String newValue) This method sets the value associated with the cookie.
public String getValue() This method gets the value associated with the cookie.
public void setPath(String uri) This method sets the path to which this cookie applies. If you don't specify a path, the cookie is returned for all URLs in the same directory as the current page as well as all subdirectories.
public String getPath() This method gets the path to which this cookie applies.

JSP : Cookie Writing

1. **Creating a Cookie object** Cookie cookie = new Cookie("key","value");
2. **Setting the maximum age** cookie.setMaxAge(60*60*24);
3. **Sending the Cookie into the HTTP response headers**

   response.addCookie(cookie);

JSP : COOKIES READING

```
Cookie x = null;
Cookie[] a = request.getCookies();
if(a!=null)
{
  for(int i=0; i<a.length; i++)
  {
        x = a[i];
  out.println(x.getName() + " : " + x.getValue() + "<br>" );
  }
}
```

JSP : Cookie Deleting

- Read an already existing cookie and store it in Cookie object.
- Set cookie age as zero using **setMaxAge() method to delete an existing cookie.**
- Add this cookie back into response header.

Session Tracking

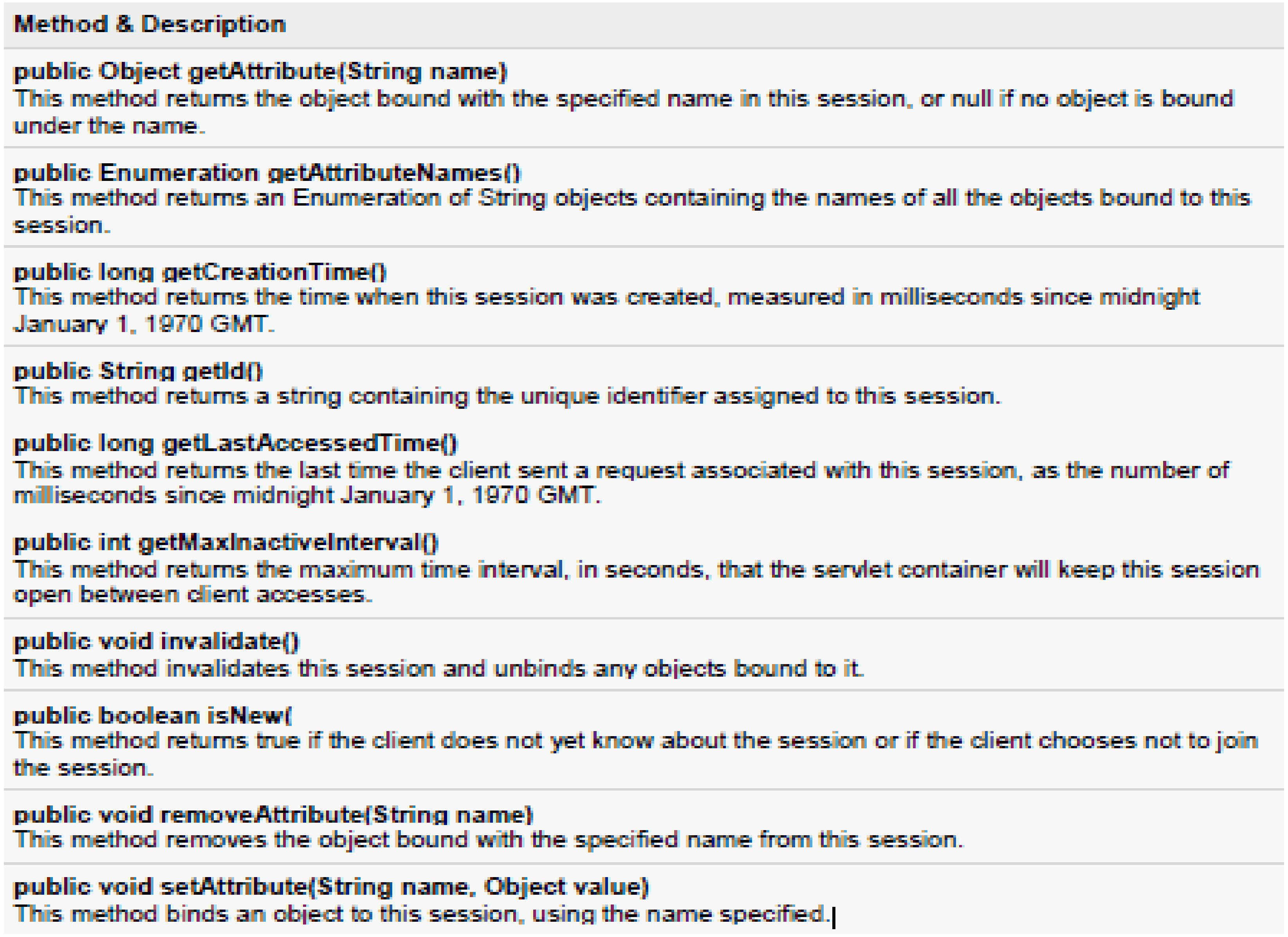
- HTTP is a "stateless" protocol which means each time a client retrieves a Web page, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request.
- Still there are following four ways to maintain session between web client and web server:

1. Cookies
2. Hidden form fields
   `<input type="hidden" name="sessionid" value="12345">`
3. URL rewriting
   http://xyz.com/file.htm?sessionid=12345
4. Session

JSP : SESSION

- JSP makes use of servlet provided HttpSession Interface which provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.
- By default, JSPs have session tracking enabled and a new HttpSession object is instantiated for each new client automatically.

SESSION METHODS



Method & Description
public Object getAttribute(String name) This method returns the object bound with the specified name in this session, or null if no object is bound under the name.
public Enumeration getAttributeNames() This method returns an Enumeration of String objects containing the names of all the objects bound to this session.
public long getCreationTime() This method returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
public String getId() This method returns a string containing the unique identifier assigned to this session.
public long getLastAccessedTime() This method returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
public int getMaxInactiveInterval() This method returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses.
public void invalidate() This method invalidates this session and unbinds any objects bound to it.
public boolean isNew(This method returns true if the client does not yet know about the session or if the client chooses not to join the session.
public void removeAttribute(String name) This method removes the object bound with the specified name from this session.
public void setAttribute(String name, Object value) This method binds an object to this session, using the name specified.

PAGE REDIRECTION

```
<%
// New location to be redirected
   String site = new String("http://www.google.com");
   response.setStatus(response.SC_MOVED_TEMPORARILY);
   response.setHeader("Location", site);
%>
```

3) Web Container, web.xml, Filter, Listener

Web Applications & Components

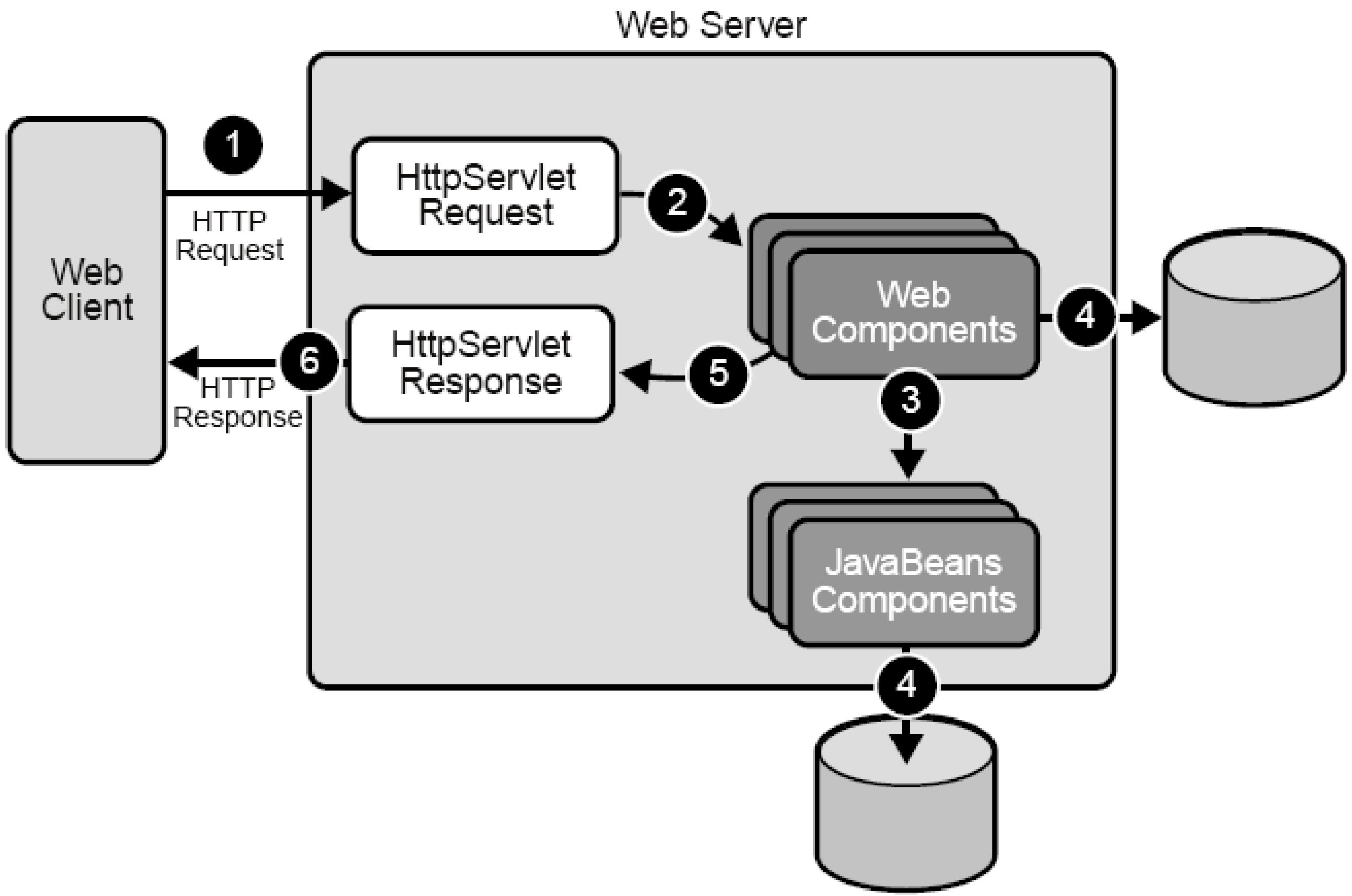
- Two types of web applications:
  - Presentation-oriented (HTML, XML pages)
  - Service-oriented (Web services)

- **Web components** provide the dynamic extension capabilities for a web server:
  - Java servlets
  - JSP pages
  - Web service endpoints

Web Application Interaction

- **[client]** sends an HTTP request to the web server
- **[web server]** HTTP request → **`HTTPServletRequest`**
- This object is delivered to a web component, which can interact with JavaBeans or a DB to generate dynamic content
- **[web component]** generates an **`HTTPServletResponse`** or pass the request to another web component
- **[web server]** **`HTTPServletResponse`** → HTTP response
- **[web server]** returns HTTP response to the client

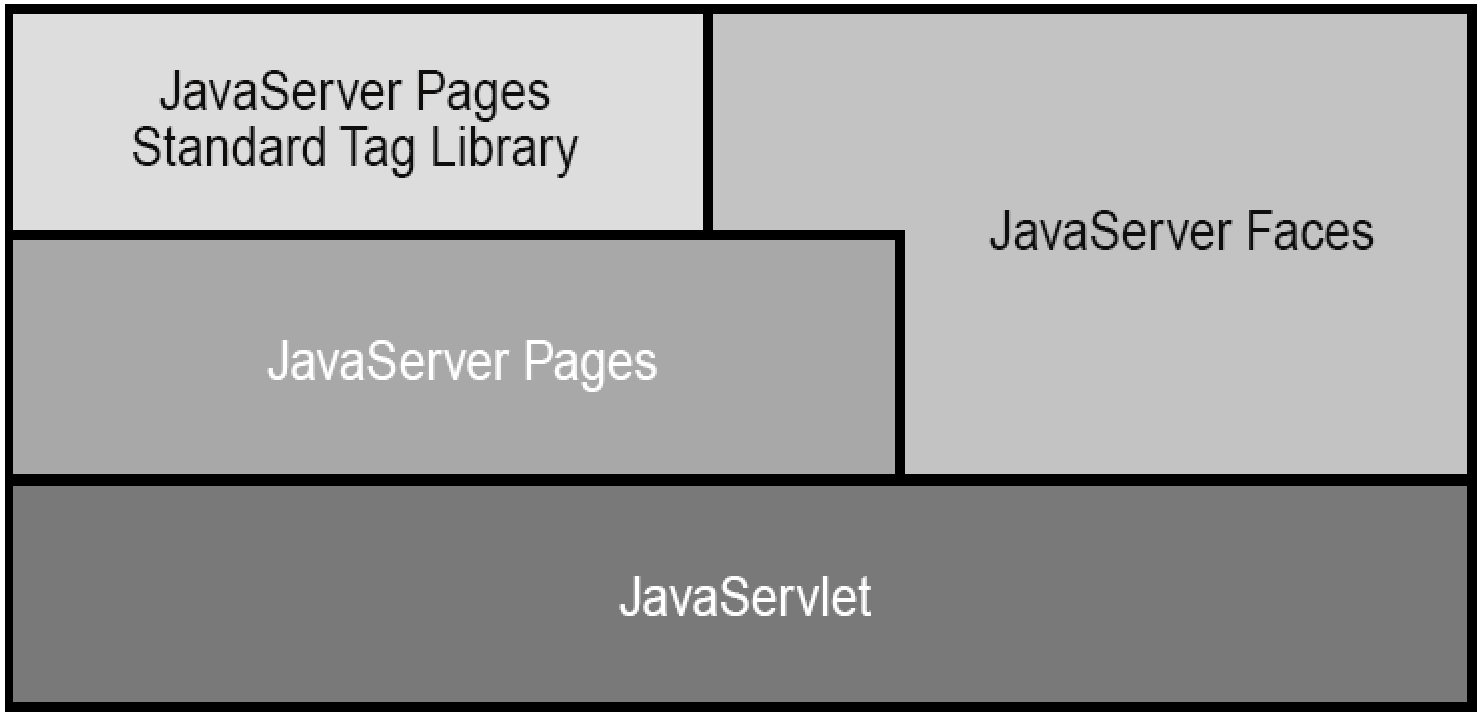
WEB APPLICATION INTERACTION



Web Components

- **Servlets -** Java classes that dynamically process requests and construct responses
- **JSP pages -** text-based documents that execute as servlets but allow a more natural approach to creating static content
- Appropriate usage
  - **Servlets** - service-oriented applications, control functions
  - **JSP** - generating text-based markup (HTML, SVG, WML, XML)

JAVA WEB APPLICATION TECHNOLOGIES



Java Servlet technology is the foundation of all the web application technologies

Web Containers

- Web components are supported by the services of a runtime platform called a **web container**
- In J2EE, a web container "implements the web component contract of the J2EE architecture“
- Web container services:
  - request dispatching
  - security
  - concurrency
  - life-cycle management
  - naming, transactions, email APIs

WEB CONTAINER EXAMPLES

- Non-commercial
  - Apache Tomcat
  - Jetty
- Commertial
  - Sun Java System Application Server
  - BEA WebLogic Server
  - Oracle Application Server
  - WebSphere
- Open source
  - JBoss

DEPLOYMENT

- Web components have to be installed or **deployed** to the web container
- Aspects of web application behaviour can be configured during application **deployment**
- The configuration information is maintained in a XML file called a web application **deployment descriptor**

Web Application Development

- A web application consists of:
  - Web components
  - Static resource files (such as images)
  - Helper classes and libraries

- The process for creating and running a web application is different from that of traditional stand-alone Java classes

Development Cycle

1. Develop the web component code
2. Develop the web application deployment descriptor
3. Compile the web application components and helper classes referenced by the components
4. Optionally package the application into a deployable unit
5. Deploy the application into a web container
6. Access a URL that references the web application

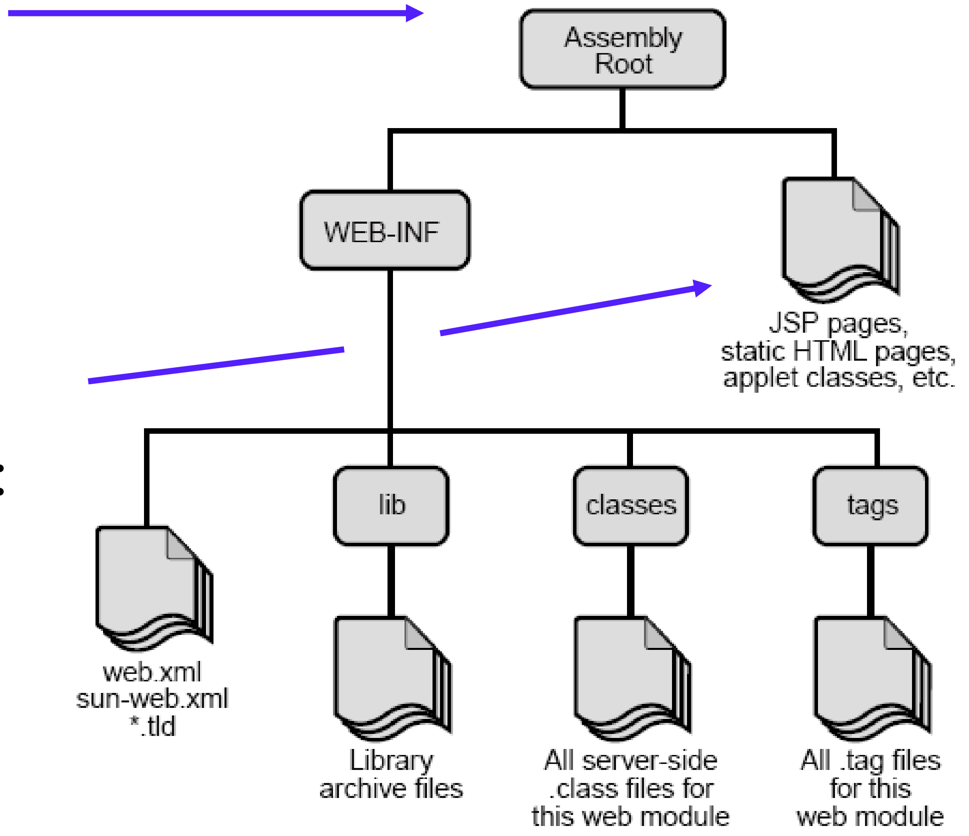
Web Modules

- According to Java EE architecture and Java Servlet Specification:
    - Web components and static web content files such as images are called *web resources*
    - A *web module* is the smallest deployable and usable unit of web resources
    - Web module corresponds to a *web application*
- A web module has a specific structure

Web Module Structure

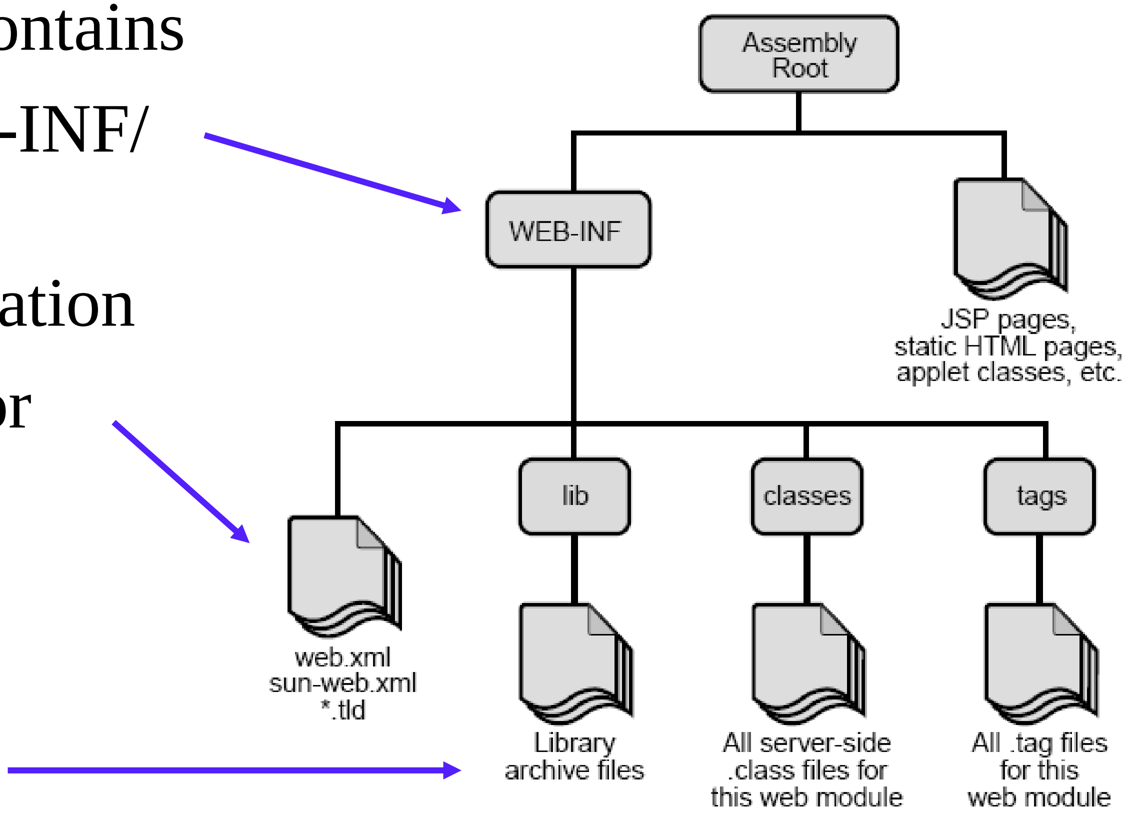
- The top-level directory of a web module is the *document root* of the application

- The document root contains:
  - JSP pages
  - client-side classes
  - client-side archives
  - static web resources


Assembly Root
WEB-INF
JSP pages, static HTML pages, applet classes, etc.
web.xml sun-web.xml *.tld
lib
Library archive files
classes
All server-side .class files for this web module
tags
All .tag files for this web module

WEB MODULE STRUCTURE

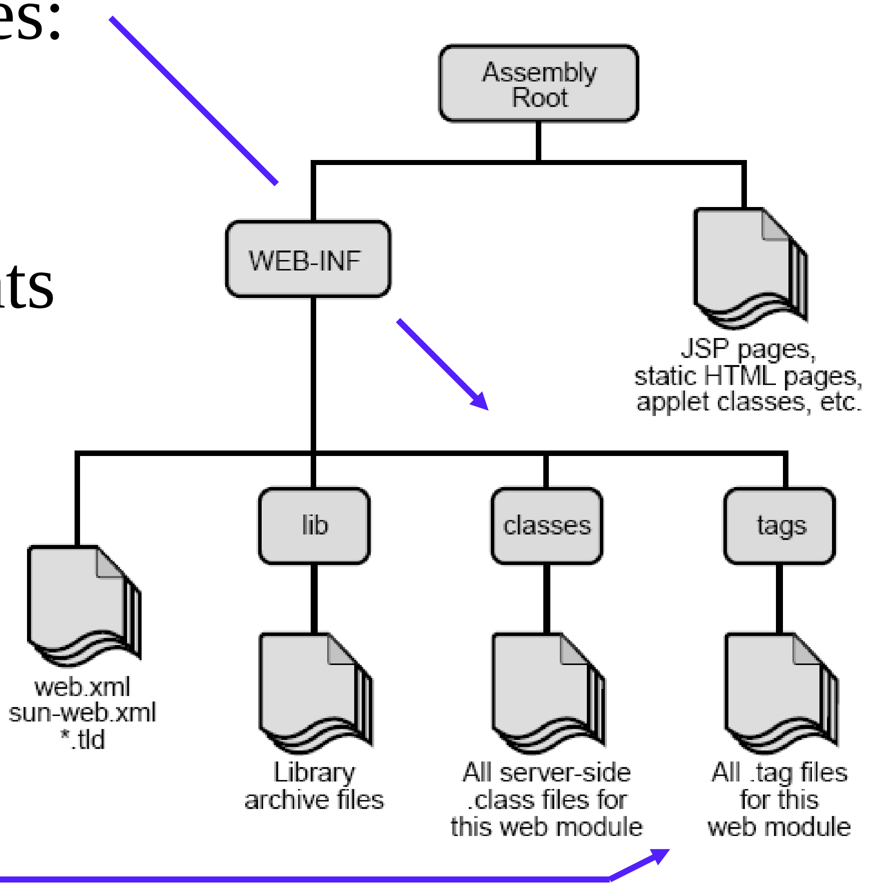
- The document root contains a subdirectory /WEB-INF/
- **web.xml**: web application deployment descriptor
- **lib**: JAR archives of libraries called by server-side classes


Assembly Root
WEB-INF
JSP pages, static HTML pages, applet classes, etc.
web.xml sun-web.xml *.tld
lib
Library archive files
classes
All server-side .class files for this web module
tags
All .tag files for this web module

Web Module Structure

- **classes**: server-side classes:
  - servlets
  - utility classes
  - JavaBeans components

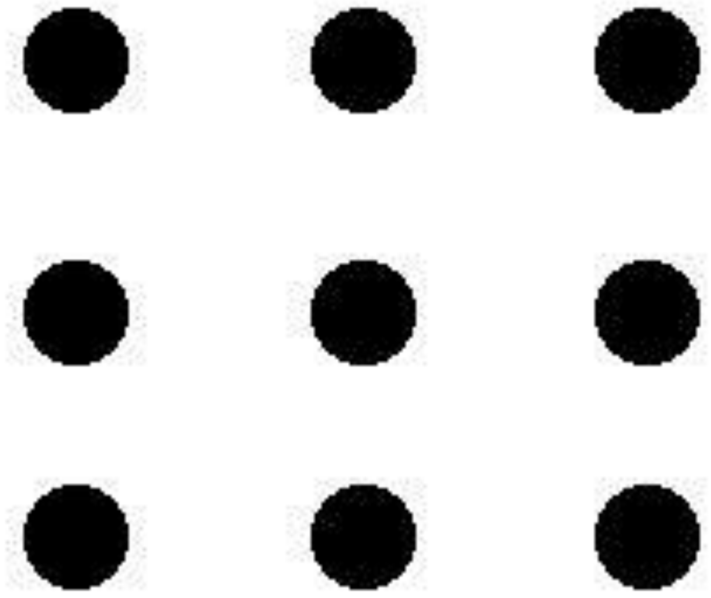
- **tags**: tag files, which are implementations of tag libraries


Assembly Root
WEB-INF
JSP pages, static HTML pages, applet classes, etc.
web.xml
sun-web.xml
*.tld
lib
Library archive files
classes
All server-side .class files for this web module
tags
All .tag files for this web module

Games and Fun!

- 1) A bear was sitting on some sort of pavement, 5 meter high from the water level. Then it suddenly felt into the water just in 1 second. The question is …………..What color was the bear and why?

Games and Fun!



The "nine dots" puzzle

The goal of the puzzle is
to link all 9 dots using
four straight lines or less,
without lifting the pen.

JSP Exercise (DONE)

- 1) Write a program to display a “Hello World” message in the Web browser. In addition, display the host name and Session Id. Write JSP code using HTML code.
- 2) Write a program to display the multiples of two. Include for loop in the scriptlet block and display the numbers
- 3) Write a program to display an error message to the user if an exception occurs in the JSP page. In the JSP page, consider a null vector and find out the length of the string using size()method of Java. Create an error handler to handle the exception thrown by this JSP page.

Spring Exercise (Evening Practice)

- 4) Inner beans are the beans that are defined within the scope of another bean. Thus, a <bean/> element inside the <property/> or <constructor-arg/> elements is called inner bean. This page gives an example to inject inner bean in spring. Give and code example that uses constructor based injection and uses inner bean configuration. Say, there is a class called A, which requires Order object to be injected.
- For example the bean file could be …

- <bean class="com.java2novice.beans.Order">
- <property name="item" value="…." />
- <property name="price" value="…" />
- <property name="address" value="…." />
- </bean>
- </constructor-arg>
- </bean>

Spring Exercise (Evening Practice)

- 5) You can define initialization and destroy methods with in the spring bean. You can configure it using init-method, and destroy-method in the xml based configuration file. These are part of spring bean life cycle. The initialization method will be called immediately after bean creation, and destroy method will be called before killing the bean instance.
- Create a class NetworkManager that has an init() method which initializes http connection object, and then the destroy() method that will close the http connection.

Spring Exercise (Evening Practice)

6) A spring bean definition contains lot of information like property values, constructor arguments, and container specific information like init and destroy method settings and so on. Spring allows to inherit all these bean properties. A child bean definition can inherit configurations from its parent definition. The child bean definition can override some values, or add new values.

- Please write code to inherit parent bean values with in child bean. You may have Employee class, the parent bean sets the common value for the property called company. Any other bean definitions will inherits it.

Spring Exercise (Done)

- 7) We can inject values into spring bean from spring bean configuration file itself. It is very simple as we saw during class example explained. Write code to populate values to spring bean instance variables. You can have Employee class, which has three fields called name, role, employee id.