JSP and Servlets

# Intro

* MVC frameworks like Struts are built on top of JSP and Servlets
* Must have JDK installed (not the JRE)

Projects: (Dynamic Web Projects)

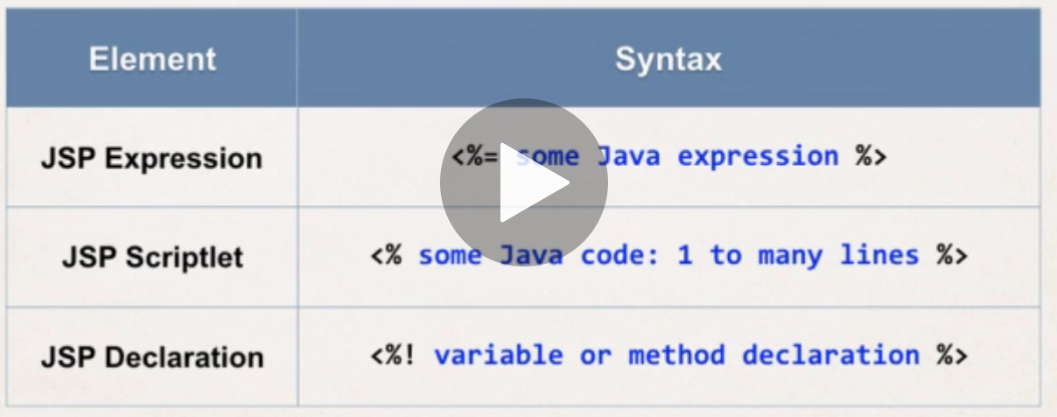
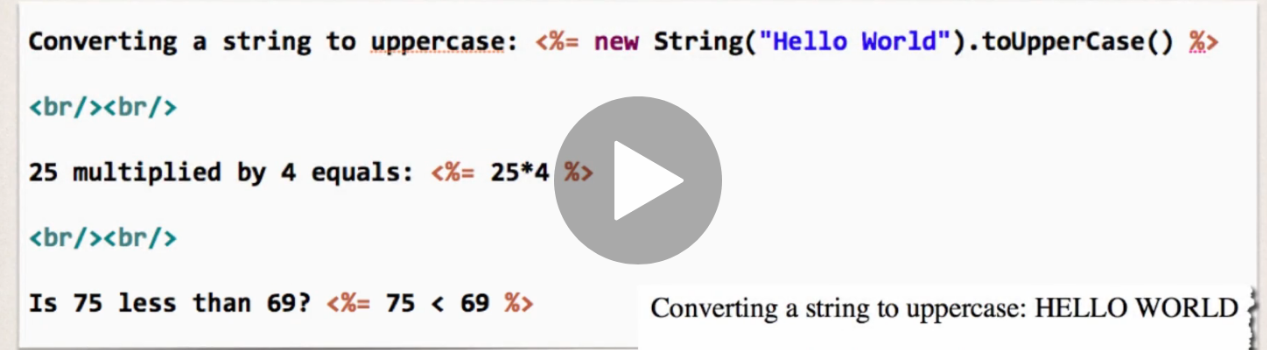
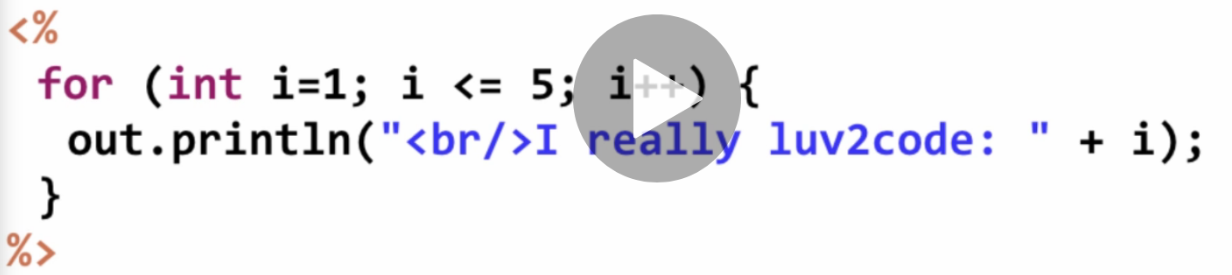
jspdemo project in Eclipse

tagdemo project

# IDE setup

* Use Eclipse Java EE
* Download tomcat from apache
* keep it within profile, extract and point Eclipse at it by going clicking on the servers tab at the bottom
* File – New – Dynamic Web Project
* JSP file goes in webcontent dir in Tomcat – when you have created a new project you see this directly in the navbar under the project in the top left
* Create a new file in the WebContent dir then right-click RunAs RunOnServer when ready to try it out

# JSP Scripting Elements

* 3 types
* **JSP Expression** – compute and expression and return it to the browser e.g. **the time o the server is <%= new java.util.Date() %>**
* **JSP Sriptlet –** 1 to many lines of proper Java code inline in the page, executed top down as page is processed
* to include content in the page use: **out.println(…)** e.g.
* minimize the amount of scriptlet code in JSP
  + you should refactor into a separate class or use MVC framework
* **JSP Declarations -** Declare a method (function in Python) in the JSP page and call the method in the same page multiple times from elsewhere
  + minimize the number of declarations and refactor into classes and MVC as above
* Note String at the start of the method name above declares the return type of the function.

# Calling a Java Class from JSP

* Create the Java class under Java Resources/ src folder in jsp project in Eclipse
  + New Package, give the package a name - com.love2code.jsp
  + Create a new class in the package – FunUtils
  + Eclipse auto-populates some stuff for you
* When calling from the JSP page have to: (CallaClass.jsp)
  + import at the top of the page <%@ page import=”com.love2code.jsp.FunUtils” %>
    - or can import “come.love2code.jsp.\*” to import all, or use a comma delimited list of methods from the class
    - then access the method as FunUtils.makeItLower()
  + or - give the fully qualified name of the method in the code (messy) – e.g. com.love2code.jsp.FunUtils

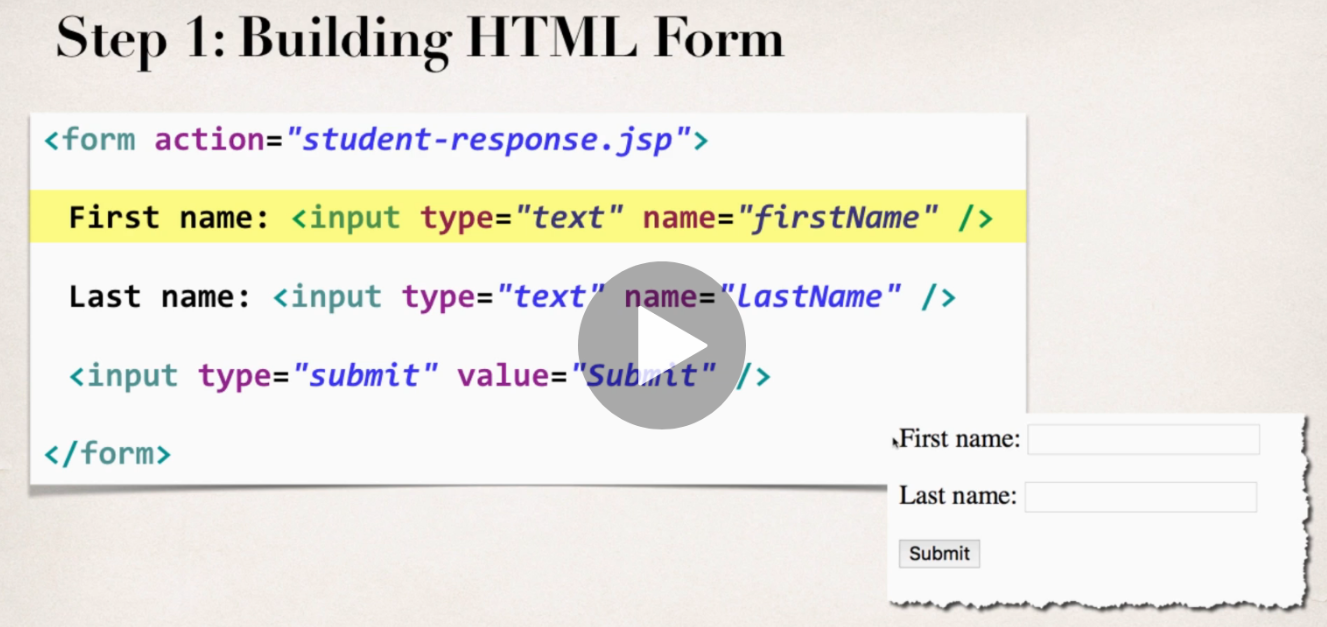
# JSP Built-in Server Objects

* List of commonly used JSP objects:
* response – useful for cookies
* examples in builtinTest.jsp

# Including files

* For e.g. including headers and footers on all web pages
* Can include html or jsp files
* <jsp:include page=”my-header.html”>
* example file-includes.jsp

# HTML Forms



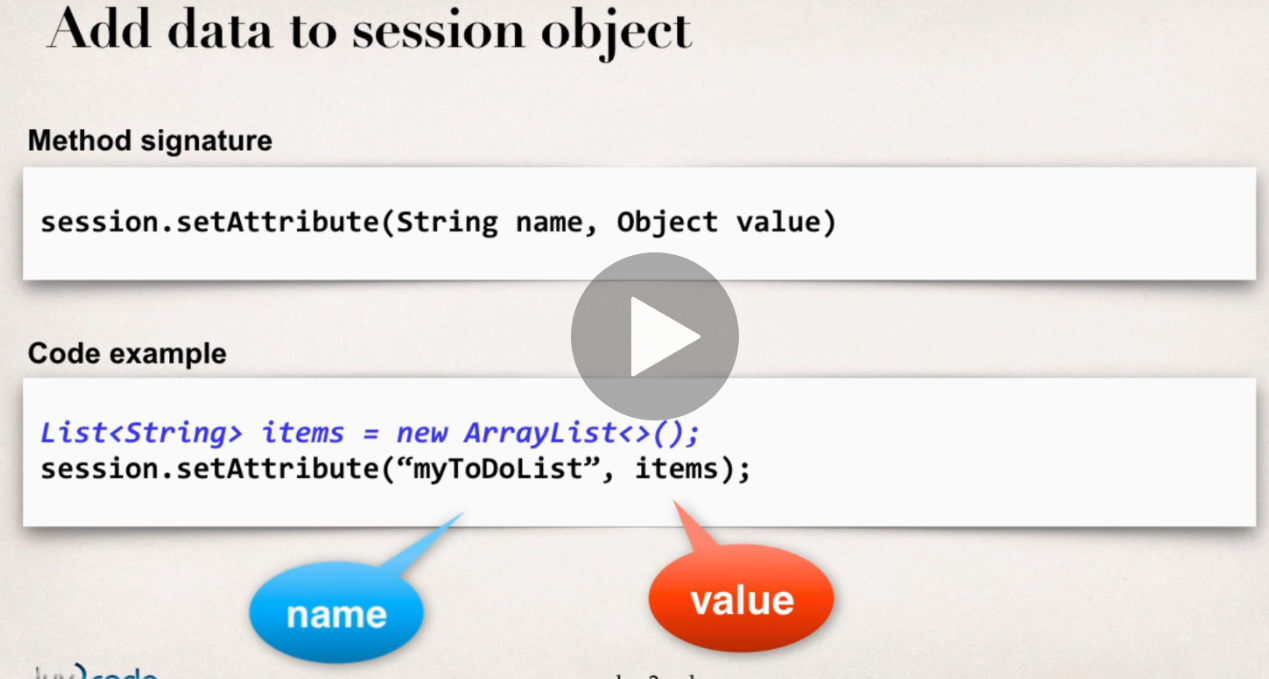
* examples – form.html and student-response.jsp
* in student-response.jsp you access the request data using the request built in
* <%= request.getParameter(“firstname”) %>
* there is also shorthand which is only used for displaying form data - ${pararm.forFieldName} e.g. ${param.firstName}

# Drop Down Lists, Radio buttons and checkboxes

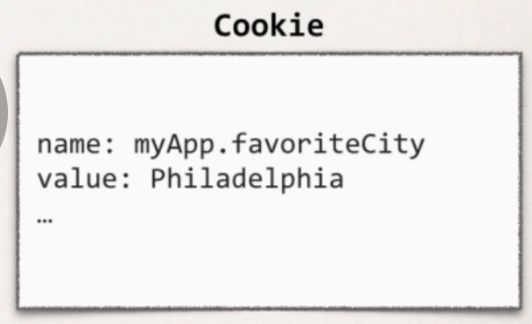
Added to examples from above

Note, for checkboxes, since we receive multiple values – an array of strings, can’t use the shorthand, have to use the request object properly and loop through the array using a JSP scriptlet

# JSP Session Object

* Created once per user’s browser session – **Tomcat server** assigns a session ID
* kept in server’s memory
* setAttribute to set something in the sesssion, getAttribute to get it back
* example in session-demo.jsp – this form points back to itself to process rather than some other jsp page
* import java.util.\* at the top – we make use of the List and ArrayList from this package
* The s**ession.getAttribute** method will always return something of type **java.lang.Object**. We **downcast** this to List<String> because we are making use of strings to keep track of our to do items.
* There is also the PageContext – this provides access to all namespaces associated with a JSP page including the session object so it can be accessed via the PageContext. Attributes set on PageContext are only available for a given page, they are not available to other pages or servlets in the application. The JSP page will always have access to the PageContext

# Cookies

* A cookie is text data exchanged between browser and server
* the site can create cookies and send to browser
* Contents are typically name or key/value pairs
* 
* browser will only send cookies that match the server’s domain
* Cookie class defined in package: javax.servlet.http
  + auto imported in all JSP pages
* Cookie constructor
  + Cookie(String name, String value)
    - I..e give the cookie and name and then a value – below is based on something retrieved directly from the request object



* by default cookies have a lifespan of 0 – expire when the browser is closed
* to get the cookies (the browser will send them for the given domain) use the request object again:
  + if there are cookies then loop through them (for tempCookie in the Cookies)
  + the if condition is looking for a cookie called “myApp.favoriteLanguage”



examples in – **cookies-homepage.jsp → Cookies-personalise-form.html → cookies-personalise-response.jsp** – the idea is when you return to the cookies-homepage.jsp after previously visiting, it will be personalised based on the favourite language you selected. Homepage will default to Java as fave language if not cookie set.

* To handle cookies with whitespace, need to URL encode and decode e.g.
* favLang = URLDecoder.decode(tempCookie.getValue(), "UTF-8");
* favLang = URLEncoder.encode(favLang, "UTF-8");

# JSP Tags

* Examples in tagdemo project – new DynamicWeb Project
* Two categories
  + JSP Custom tags (ones you write yourself)
  + JSP Standard Tag Libraries (JSTL) – standard set from Oracle
    - includes Core, message, functions (string manipulation), xml and SQL for accessing a database (the latter are not good for real world prod apps….)
* help to move business logic into supporting classes and out of the JSP – minimize scriptlet code in JSP and implements logic elsewhere.
* insert a tag into the JSP page to access the class – so keeps presentation and business logic separate e.g. imagine we have a class called weatherReport – access it:
  + <demo:weatherReport city=”Philadelphia”>
* Add JSTL jar files to the project in the WEB-INF lib folder – this is a classpath for the web application
  + https://mvnrepository.com/artifact/org.glassfish.web/javax.servlet.jsp.jstl/1.2.1
  + <https://mvnrepository.com/artifact/javax.servlet.jsp.jstl/javax.servlet.jsp.jstl-api/1.2.1>
* reference them *<%@* **taglib** uri=*"http://java.sun.com/jsp/jstl/core"* prefix=*"c"* %>

prefix=”c” is short for core library

* access the core library in code: <**c:set** var=*"time"* value=*"*<%= new java.util.Date() %>*"* />

# Java Basics

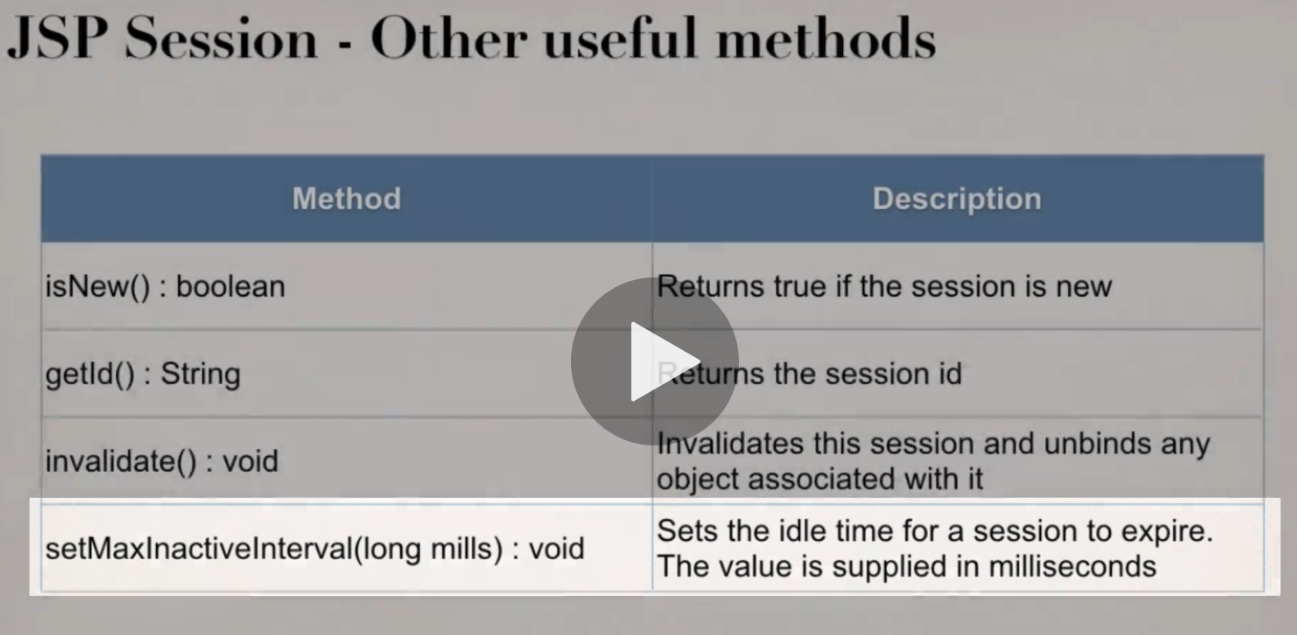
# Method names

Below leading ‘String’ means the method returns a string and accepts a string argument called ‘data’

**String makeItLower(String data) {**

**return data.toLowerCase();**

**}**



# Static and Dynamic Methods

<https://www.geeksforgeeks.org/static-methods-vs-instance-methods-java/>

If you create a static method such as:

**public static String thing() {}** - then you can access the method without instantiating an object (public means its accessible to any other class)

instance methods require an object of the class to be instantiated before accessing the method

note the item after the method() is what the method returns – void means it returns nothing. Invalidate would be used to clear down a user’s session

# Variables

String[] flavours = request.getParameterValues("favouriteIceCream")

above instantiates an array of strings called **langs** from the return value of **request.getParameterValues** from the “favouriteIceCream” form element

# For Loops

**<%**

**String[] flavours = request.getParameterValues("favouriteIceCream");**

**for (String tempFlavour : flavours) {**

**out.println("<li>" + tempFlavour + "</li>");**

**}**

**%>**

Above for loop loops through the array of strings from previous example – it sets tempFlavour as the thing to iterate over like in Python – **for flavour in flavours:** - then prints out each flavour as an html list item.

Since we’re dealing with form data, we should check for null pointer exceptions (i.e. the person didn’t enter a value):

        <%  
            String[] langs = request.getParameterValues("favoriteLanguage");  
          
            if (langs != null) {  
                for (String tempLang : langs) {  
                    out.println("<li>" + tempLang + "</li>");  
                }  
            }  
        %>

**Void keyword**

The void keyword specifies that a method should not have a return value.