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

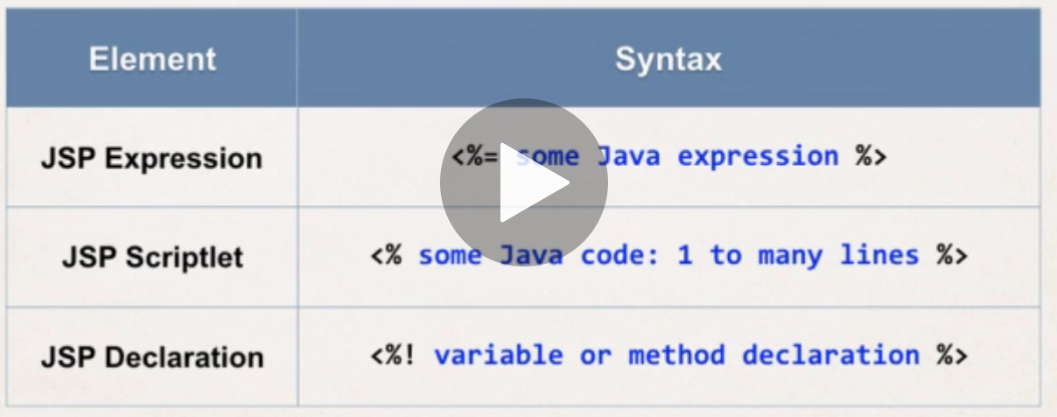
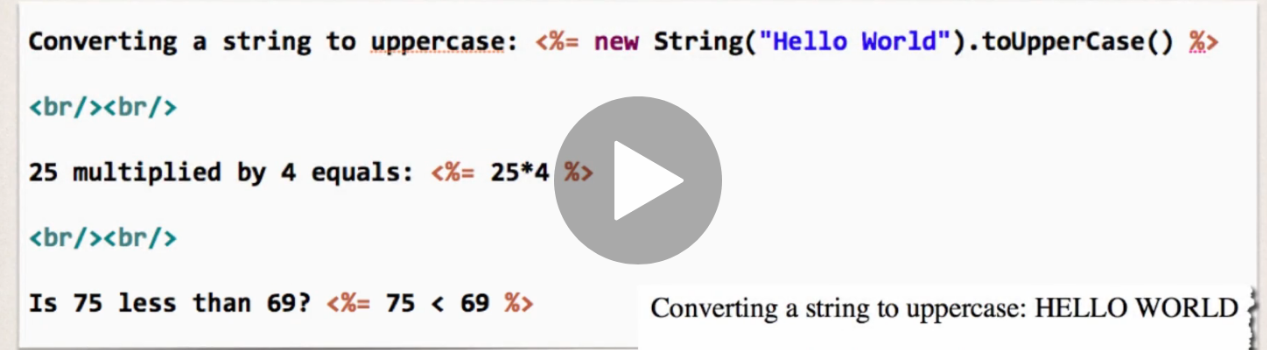
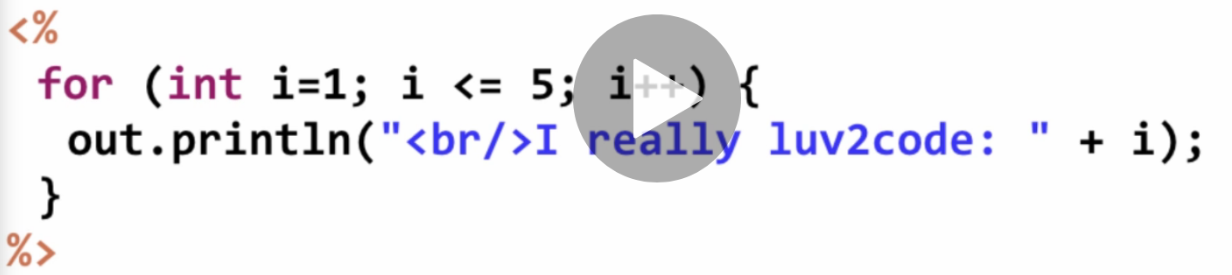
tagdemo project

servletdemo project (useful – this starts to tie things together with Tomcat Admin course)

# 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

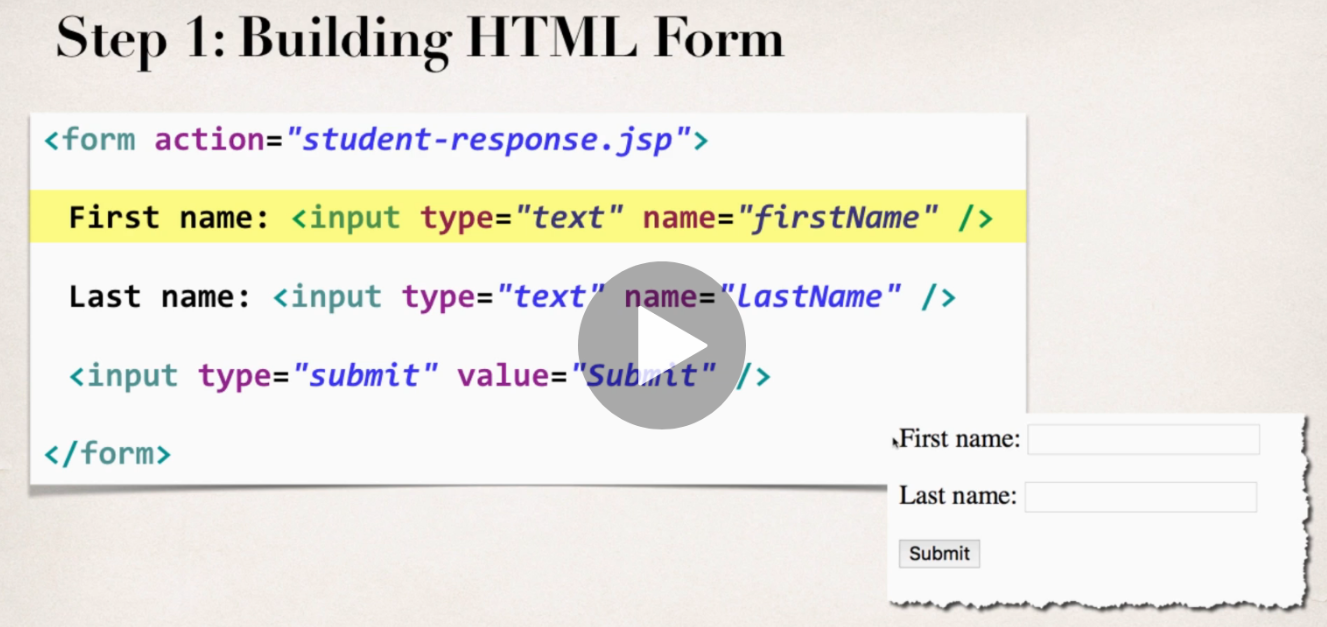
# JSP Built-in Server Objects

* 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



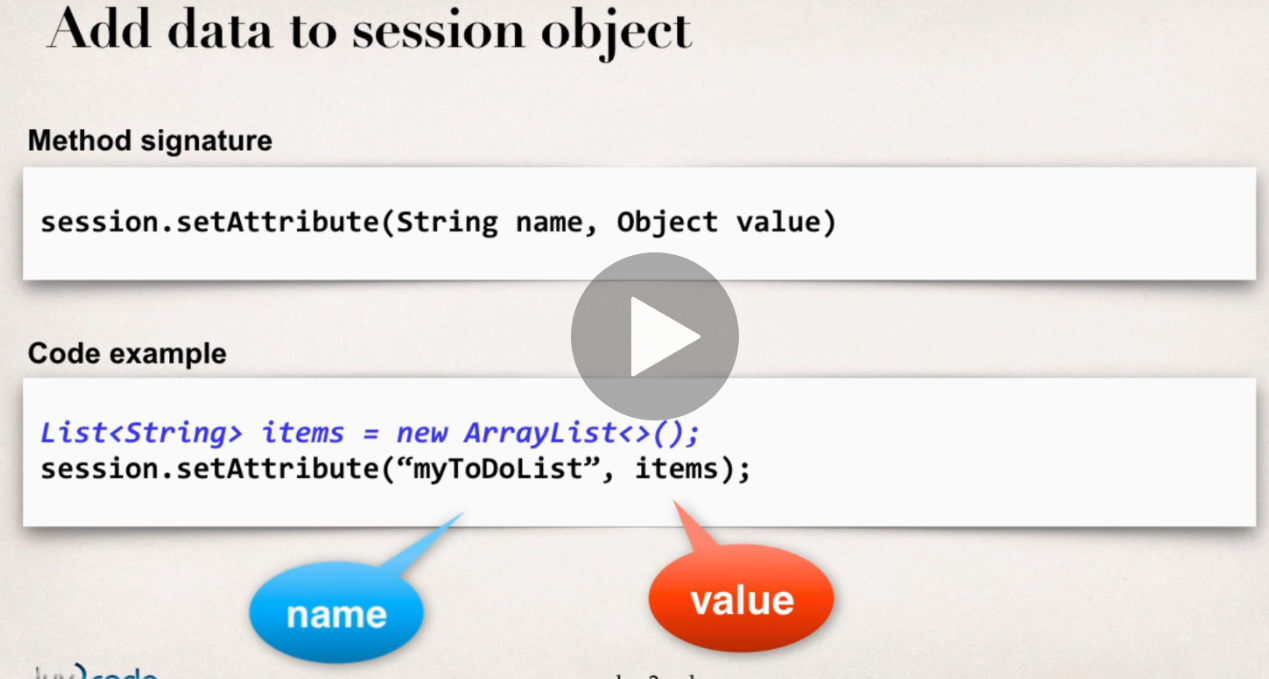
* examples – form.html and student-response.jsp
* in student-response.jsp you access the request data using the request built in
* <%= request.getParameter(“firstname”) %>
* there is also shorthand which is only used for displaying form data - ${pararm.forFieldName} e.g. ${param.firstName}

# Drop Down Lists, Radio buttons and checkboxes

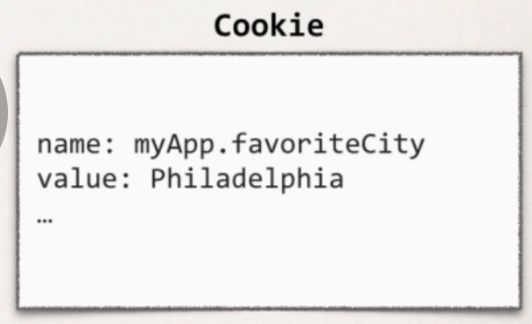
Added to examples from above

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
* libraries of functions and resources that you can access by putting <%@ taglib uri=xyz %> in your code
* 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>

## JSP Core Taglib

example – we create a new Java Package first to hold the Student class – Java Resources/src - new package

examples – **tagdemo project. Foreach-student-test.jsp**

* Every page that uses the Core tags must include reference *<%@* **taglib** uri=*"http://java.sun.com/jsp/jstl/core"* prefix=*"c"* %>
* prefix=”c” is short for core library
* the uri is used to associate to the .jar file you’ve dropped locally
* access the core library in code: <**c:set** var=*"time"* value=*"*<%= new java.util.Date() %>*"* />
* when using JSTL tags they have to be part of an attribute of one of the scopes – page, request, session – below is being set to the pageContext



* then use the forEach tag
* core tags have **if** condition:
* <c:if test=”$tempStudent.goldCustomer}”>

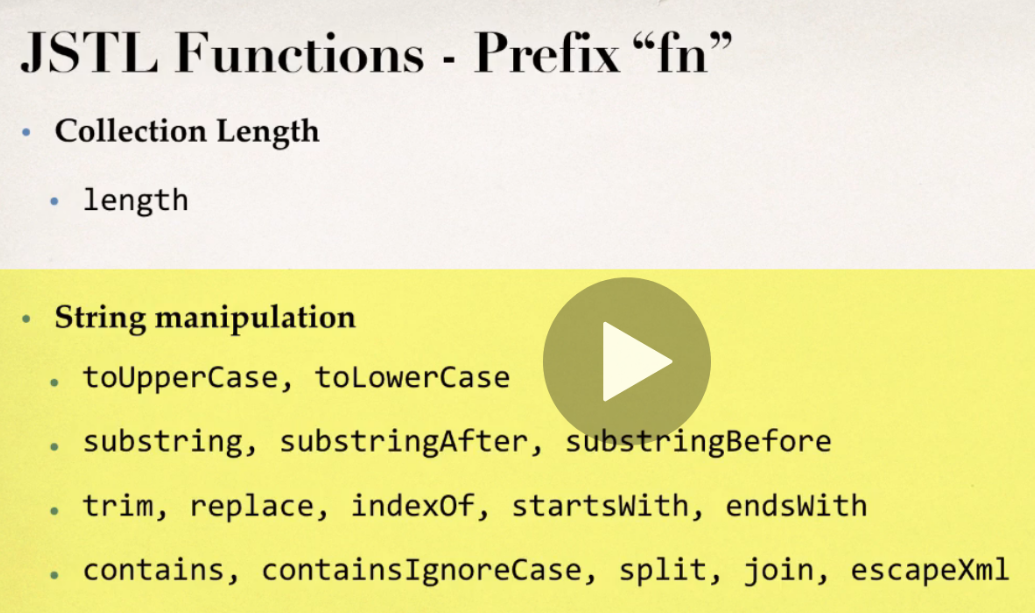
Special Discount

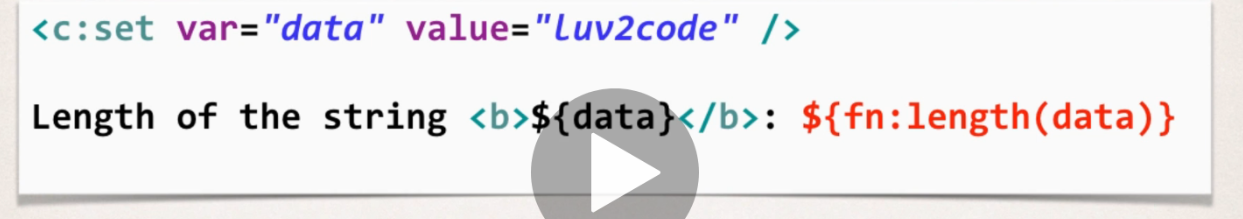
</c:if>

* JSP will call the isGoldCustomer() method for you automatically in the background when you do $tempStudent.goldCustomer above and return a boolean
  + above will display “Special Discount” if true and nothing if false
* **choose tag**
* - like a switch clause

## Function tags

* Prefix “fn”
* examples **function-test.demo**
* lightweight function routines
* Every page that uses the Core tags must include reference *<%@* **taglib** uri=*"http://java.sun.com/jsp/jstl/functions"* prefix=*"fn"* %>
* .length – pass in array list and get length



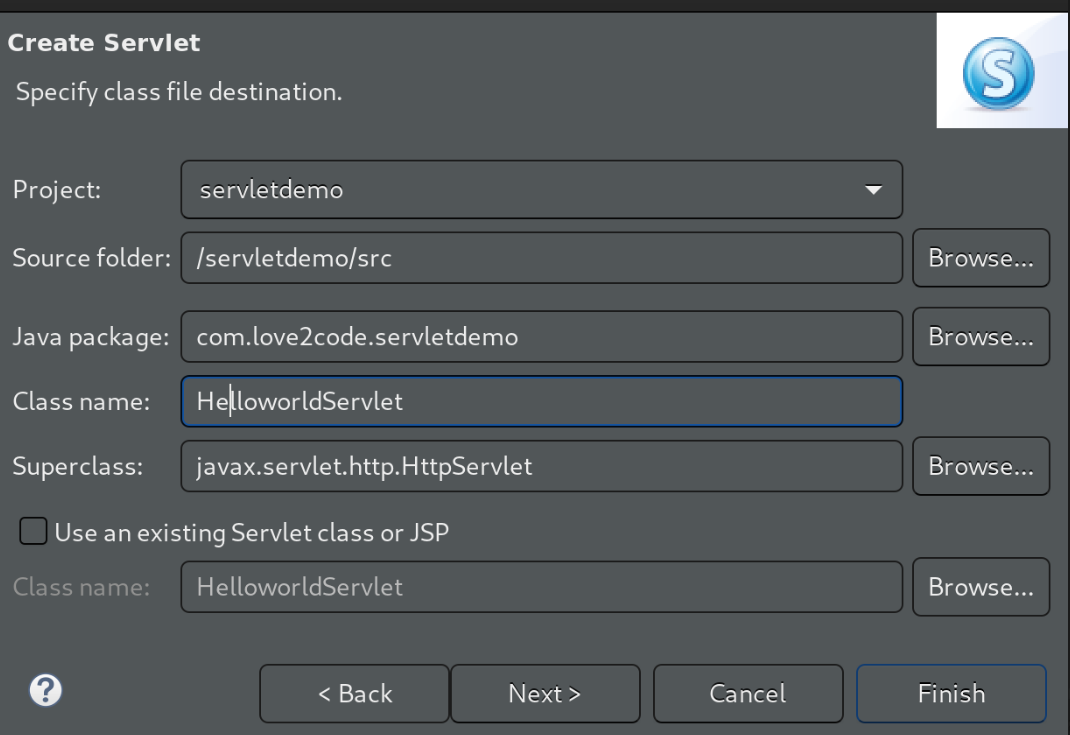


* fn:split() function splits a String into an array of substrings based on a delimeter
* 
* fn:join() function concatenates a String array into single string
* 

## JSTL Internationalisation

* Adapt app to different languages without making changes to source code

# Servlets (vs JSP)

* Examples – **servletdemo** project **HelloWorldServlet.java** (in Java resources)– create dynamic web project as usual but hit next and get Eclipse to generate web.xml deployment descriptor
  + this is stored in /WEB-INF dir – the servlet itself contains the context root on which to access the servlet e.g. /helloworldservlet (Tomcat Admin course also said it could be set in web.xml)
  + this screen also sets the context root (servletdemo in this case)
* A servlet is a **java class** that is processed on the server
* Java class generates HTML that is returned to the browser using out.println
* Can read HTML form data, use cookies and sessions etc.
* create a new servlet – right click on the project – new/servlet – give the package a suitable name, accept other defaults. Eclipse generates a lot of default stuff in the servlet – and creates the HelloworldServlet.java file under Java resources
* 
* Eclipse can create boilerplate code for the various methods you can use on the server, we just used **doGet** and **doPost** (what happens when browser GETS and POSTS)
* 
* can use either one for building Java web apps
* best practice – integrate them both together
  + servlet does the business logic
  + JSP handles the presentation view
  + this is the MVC Design pattern (used by Spring and Struts)

# Java Basics

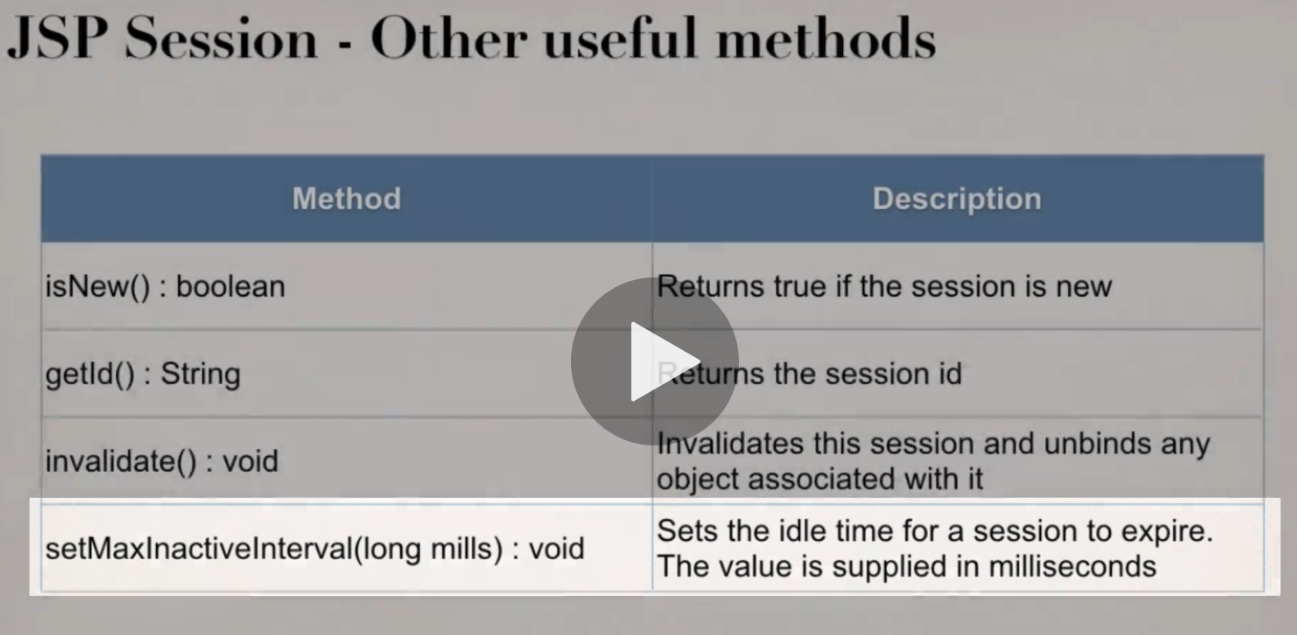
# Method names

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.

## User Defined classes

the classpath for user defined classes seems to be under Java Resources/src – this is where we put the love2code examples package (come.love2code.jsp.tagdemo) - we define as a package, then add classes under it – the are then to import into jsp files – i.e. new package, then new class

package com.love2code.jsp.tagdemo;

public class Student {

private String firstName;

private String lastName;

private boolean goldCustomer;

}

.jar files such as the JSTL go in the WebContent/WEB-INF/lib folder

## Constructor

A block of code inside a class that initialises the newly created object – it resembles an instance method but it doesn’t have a return type.

<https://beginnersbook.com/2013/03/constructors-in-java/>

the text in bold below is the constructor (Eclipse can create these for you when you have defined the vars that make up the class – right click source/Generate Constructors using fields)

**this** keyword refers to the current object

package com.love2code.jsp.tagdemo;

public class Student {

private String firstName;

private String lastName;

private boolean goldCustomer;

**public Student(String firstName, String lastName, boolean goldCustomer) {**

**super();**

**this.firstName = firstName;**

**this.lastName = lastName;**

**this.goldCustomer = goldCustomer;**

}

}

## Getters and Setters

Use Eclipse to generate these automatically – right click source/Gnerate getters and setters

## What are getter and setter?

In Java, getter and setter are two conventional methods that are used for retrieving and updating value of a variable.

The following code is an example of simple class with a private variable and a couple of getter/setter methods:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | public class SimpleGetterAndSetter {      private int number;        public int getNumber() {          return this.number;      }        public void setNumber(int num) {          this.number = num;      }  } |

The class declares a private variable, number. Since number is private, code from outside this class cannot access the variable directly, like this:

|  |  |
| --- | --- |
| 1  2  3 | SimpleGetterAndSetter obj = new SimpleGetterAndSetter();  obj.number = 10;    // compile error, since number is private  int num = obj.number; // same as above |

Instead, the outside code have to invoke the getter, getNumber() and the setter, setNumber() in order to read or update the variable, for example:

|  |  |
| --- | --- |
| 1  2  3  4 | SimpleGetterAndSetter obj = new SimpleGetterAndSetter();    obj.setNumber(10);  // OK  int num = obj.getNumber();  // fine |

So, a setter is a method that updates value of a variable. And a getter is a method that reads value of a variable.