Building Dynamic Java Web Applications

Glassfish, JAVA EE, Servlets, JSP, EJB

Java platform

- A Java platform comprises the **JVM** together with **supporting class libraries**.

Java 2 Standard Edition (J2SE)

- (1999) provides core libraries for data structures, xml parsing, security, internationalization, db connectivity, RMI

Java 2 Platform, Enterprise Edition (J2EE)

- provides more class libraries for servlets, JSPs, Enterprise Java Beans, advanced XML

Java Platform, Enterprise Edition (Java EE)

- When Java Platform 5.0 was released (2004) the ‘2’ was dropped from these titles.

Java platform

- A Java platform comprises the **JVM** together with **supporting class libraries**.

Java Micro Edition (Java ME)

  - comprises the necessary core libraries and tools for writing Java for embedded systems and other small footprint platforms, along with some **specialised libraries** for specific types of device such as **mobile phones**.

What is a Java Web application?

Java Web Application

A **Java web application** generates **interactive web pages** containing various types of markup language (**HTML**, **XML**, and so on) and **dynamic content**.

It is typically comprised of web components such as:

- **JavaServer Pages (JSP)**
- **Servlets**
- **JavaBeans**

to **modify** and temporarily **store data**, **interact with databases** and **web services**, and **render content** in response to **client requests**.

https://grizzly.dev.java.net/

What is the Java Enterprise Edition?

Java EE (Enterprise Edition)

Java EE (Enterprise Edition) is a widely used **platform** containing a **set of coordinated technologies** that significantly reduce the cost and complexity of:

- **developing**
- **deploying and**
- **managing**

multitier, server-centric applications.
Java EE builds upon the Java SE platform and **provides a set of APIs** (application programming interfaces) for developing and running portable, robust, scalable, reliable and secure server-side applications.

Java EE 6 is supported only by the GlassFish server v3.x.

http://netbeans.org/kb/trails/java-ee.html

Java EE 6 Platform

- The Java EE platform uses a simplified programming model. **XML deployment descriptors** are **optional**. Instead, a developer can simply enter the information as an **annotation** directly into a Java source file, and the **Java EE server** will configure the component at deployment and runtime
- With **annotations**, you put the specification information in your code next to the program element affected.

http://download.oracle.com/javaee/6/tutorial/doc/bnaaw.html

Java EE application model

- an architecture for implementing **services as multitier applications** that deliver the scalability, accessibility, and manageability needed by enterprise-level applications.
- With this structure you can more easily change one of the tiers without compromising your entire application.
- **Business and presentation logic** - to be implemented by the **developer**
- **Standard system services** – to be provided by the **Java EE platform**

http://download.oracle.com/javaee/6/tutorial/doc/bnaaw.html

What is a Java Servlet?

Java Servlets

- **Servlets are Java classes that dynamically process requests and construct responses.**
- Server side replacement for CGI
- Extensions to Java enabled web-servers
- Inherently **multi-threaded**.
- One thread per request.
- Very efficient.
- Platform independent.

How do Servlets work?

- **Servlets** run inside a **Web Container** - the component of the web server that runs and interacts with Servlets
- **Servlet** is running on the server listening for requests
- When a **request** comes in, a **new thread** is generated by the **web container**.

What is a Java EE Container?

Java EE Containers

Java EE containers

• are the **interface** between a **Java component** and the **low-level platform-specific functionality** (*i.e.* **transaction and state management, multithreading, resource pooling, etc.**) that supports the component.

• provide for the separation of **business logic** from **resource** and **lifecycle management.**

• this allows developers to focus on writing business logic rather than writing **enterprise infrastructure**.

The **Java EE platform** uses "**containers**" to simplify development.

http://download.oracle.com/javaee/6/tutorial/doc/bnabo.html

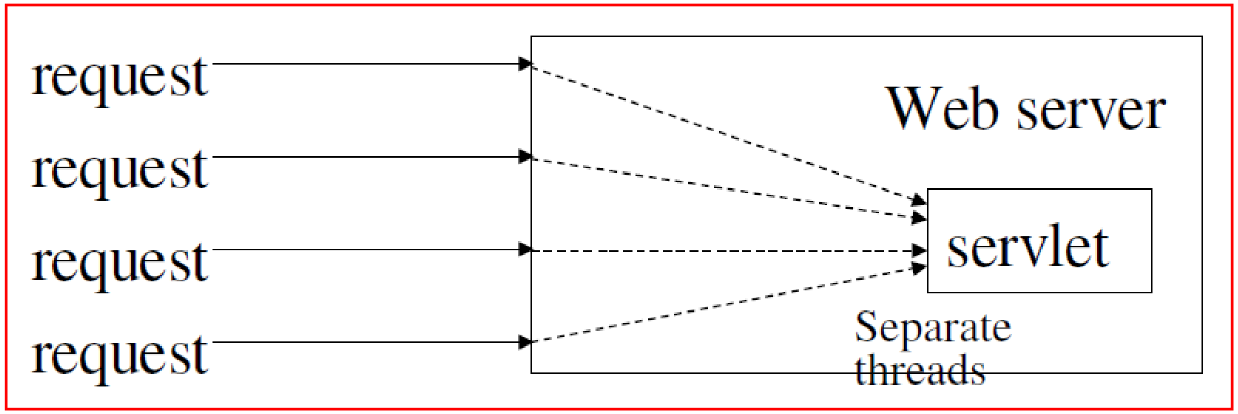
http://www.oracle.com/technetwork/java/javaee/javaee-faq-jsp-135209.html#diff

Java EE Containers

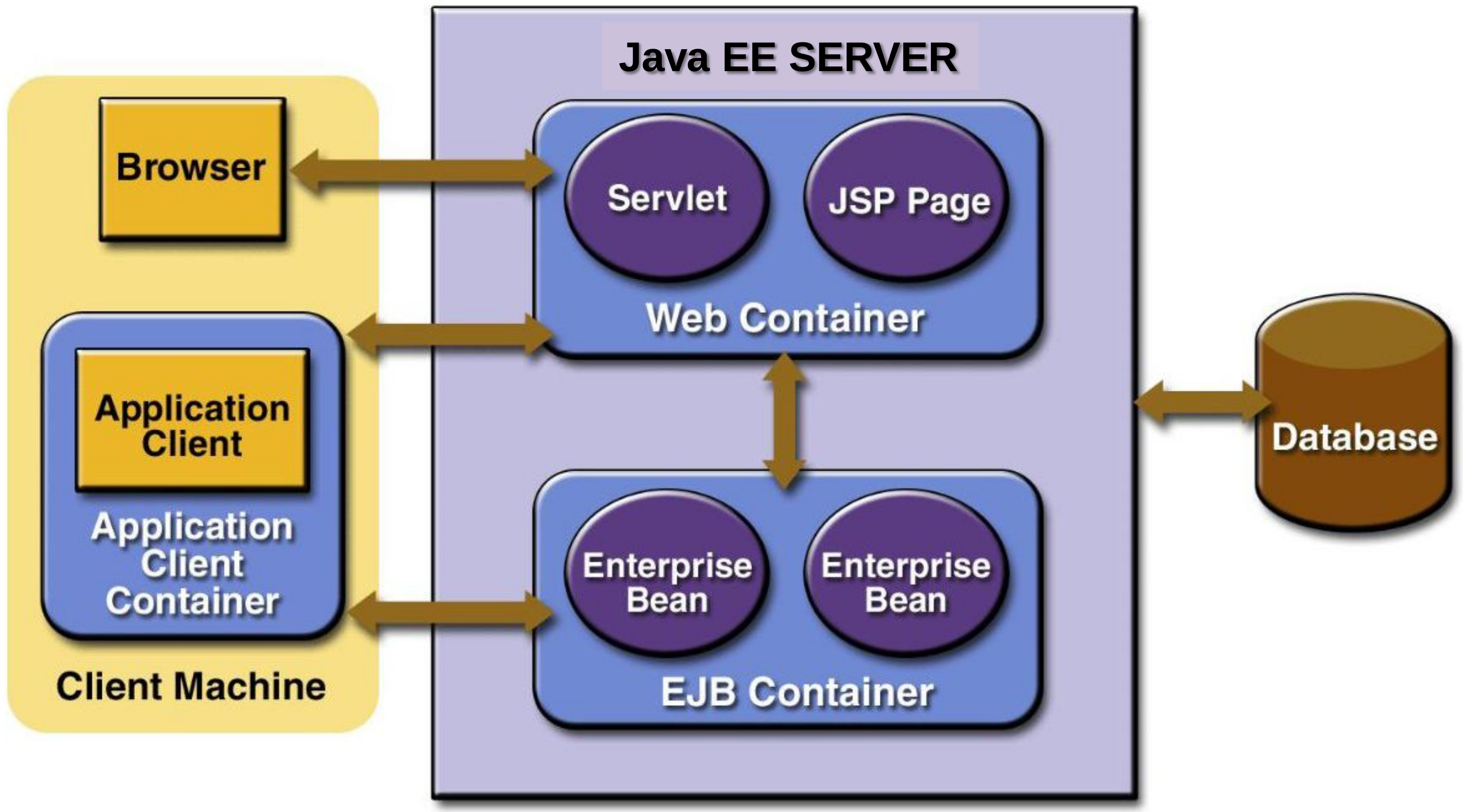
When a request comes in:

- a **Servlet** needs to be **instantiated** and create a **new thread** to handle the request.
- call the **Servlet's doPost()**or **doGet()** method and pass the **HTTP request** and **HTTP response objects**
- get the request and the response to the **Servlet**
- manage the life, death and resources of the **Servlet**

* All of the above are the tasks of the **web container**.



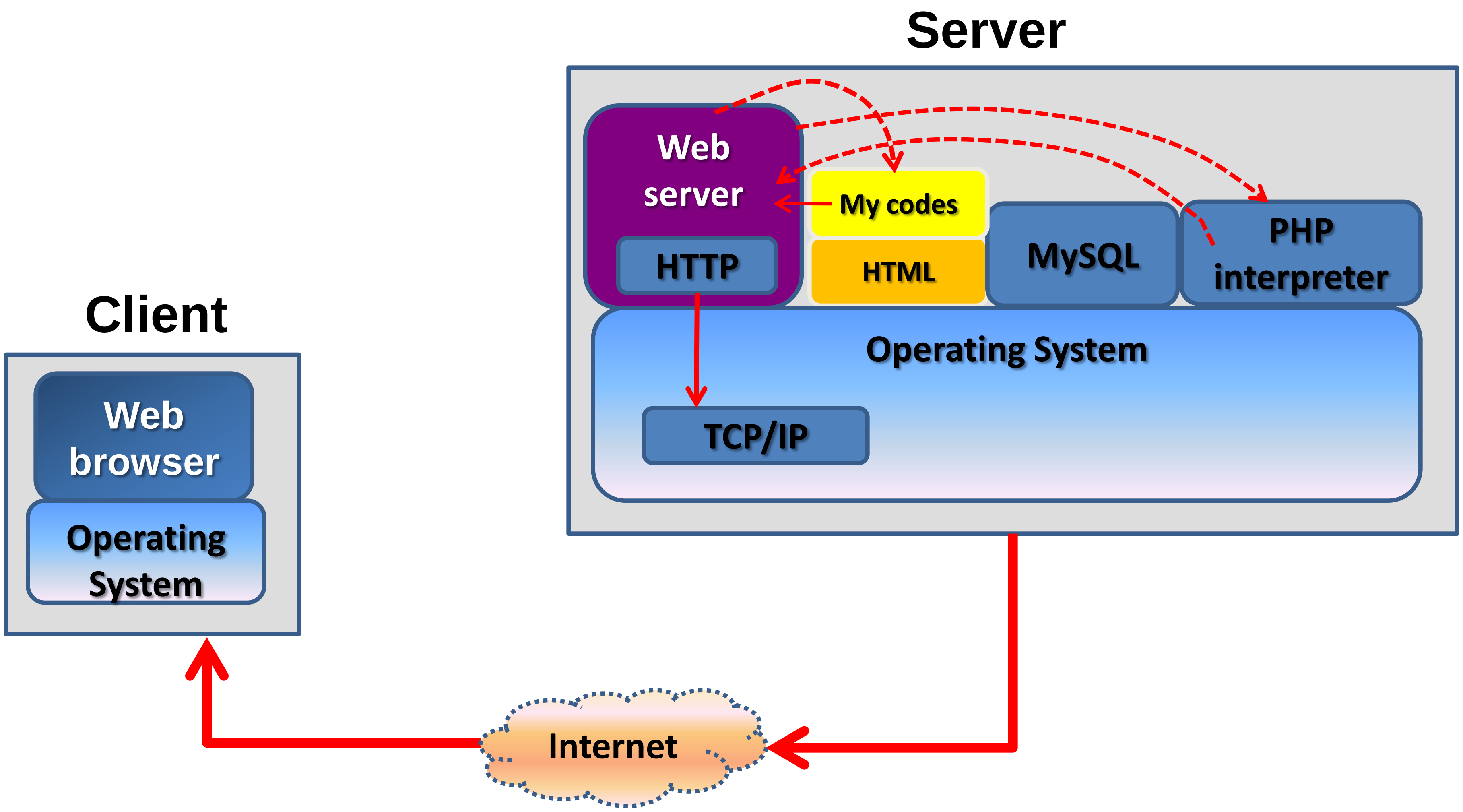
Java EE Containers



From Bodoff et. al. 2005

Recall: (PHP-MySQL) Server: response

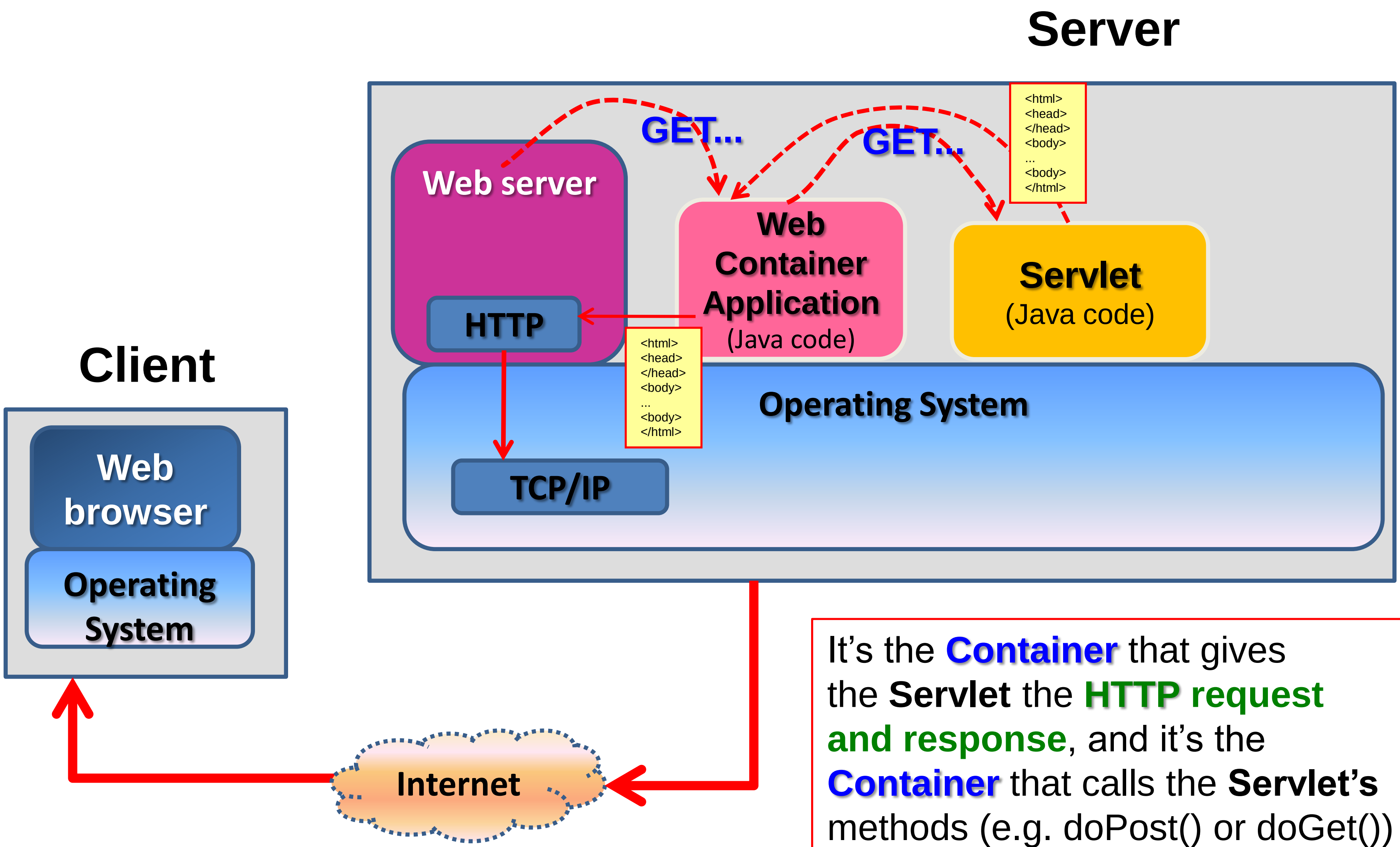
- Webserver supports HTTP.



Historically (Java Web App)

Server: response

- Webserver supports HTTP.

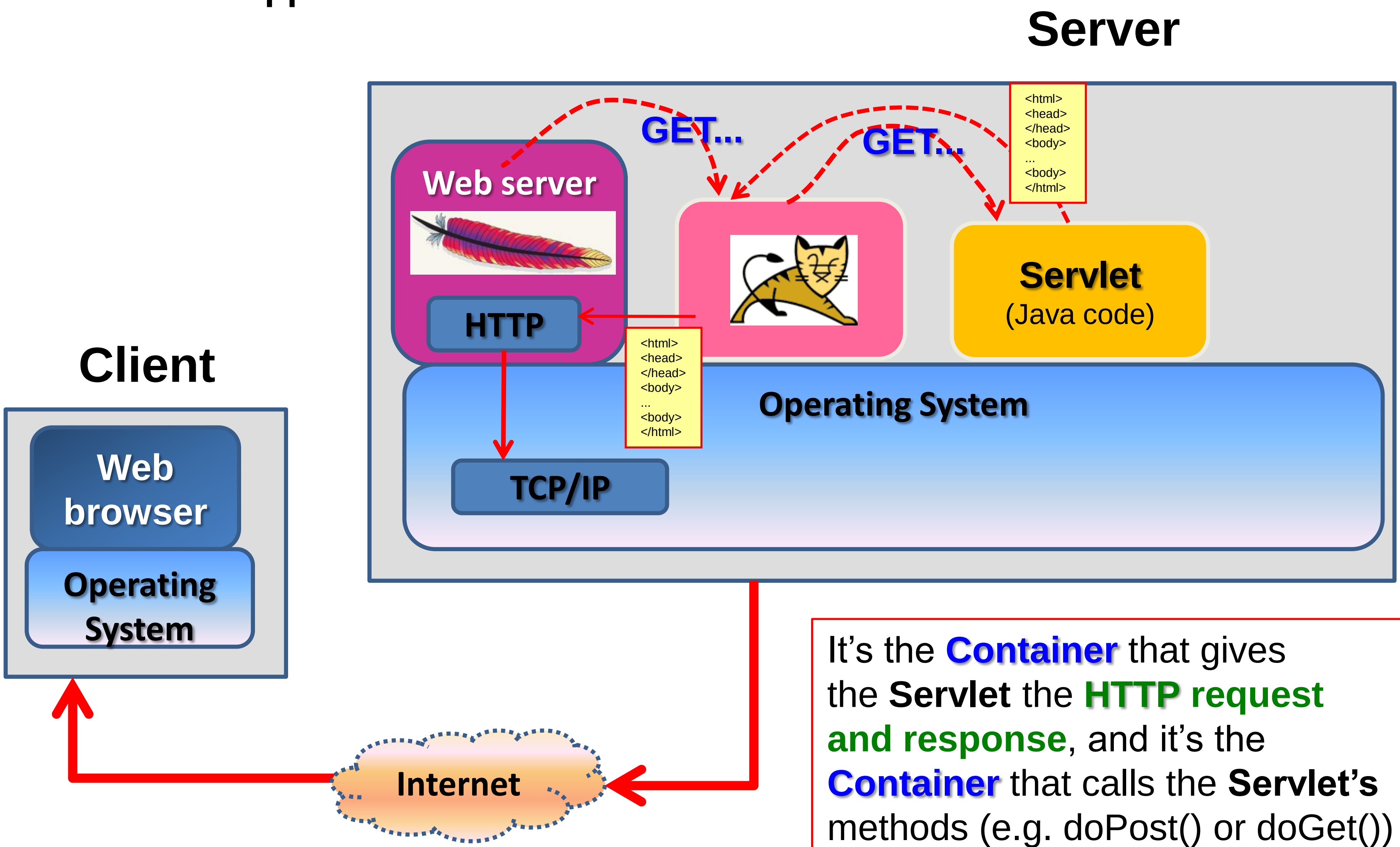


It's the **Container** that gives the **Servlet** the **HTTP request and response**, and it's the **Container** that calls the **Servlet's** methods (e.g. doPost() or doGet())

Historically (Java Web App)

Server: response

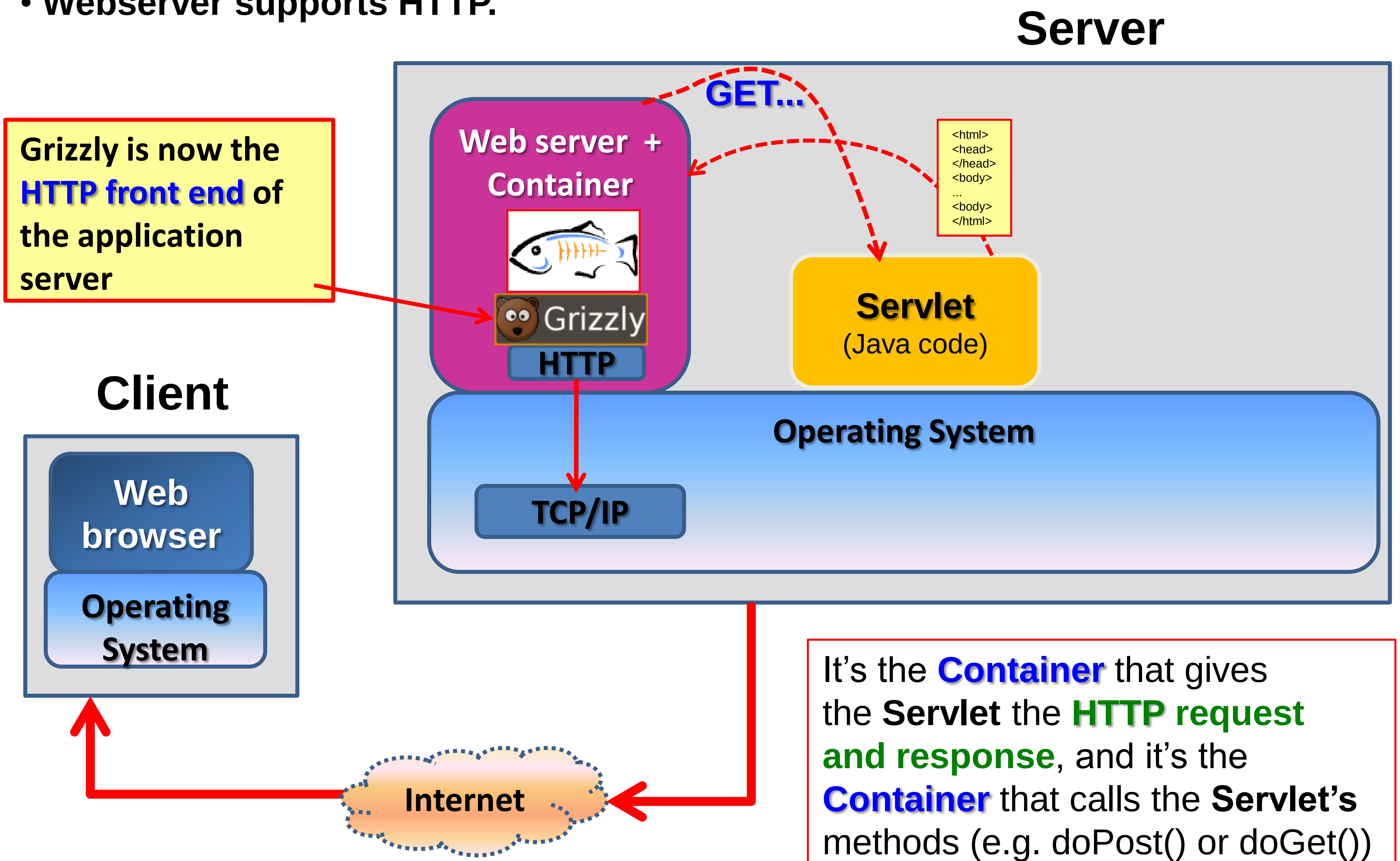
- Webserver supports HTTP.



It's the **Container** that gives the **Servlet** the **HTTP request and response**, and it's the **Container** that calls the **Servlet's** methods (e.g. doPost() or doGet())

(Java Web App) Server: response

- Webserver supports HTTP.



Grizzly is now the **HTTP front end** of the application server

It's the **Container** that gives the **Servlet** the **HTTP request and response**, and it's the **Container** that calls the **Servlet's** methods (e.g. doPost() or doGet())

Java Servlets

Java Servlets simplify web development by providing **infrastructure** for **component**, **communication**, and **session management** in a web container that is integrated with a **web server**.

- Writing **Servlets** is like writing Java codes that place an HTML page inside a Java class (this is the **worst part** of Servlets!)

- (**Historically!**) requires a **deployment descriptor** (**DD**). This is in the form of an **XML file**.

- **Servlets** do not have a **main()** method.
- **Servlets** are under the control of another Java application called a **Container**

http://www.oracle.com/technetwork/java/javaee/javaee-faq-jsp-135209.html#diff

JavaBeans

- **manage the data flow** between the following:

Client/Database	Server
application client or **applet**	components running on the Java EE **server**
database	**Server** components

- **JavaBeans** components are not considered Java EE components by the Java EE specification.

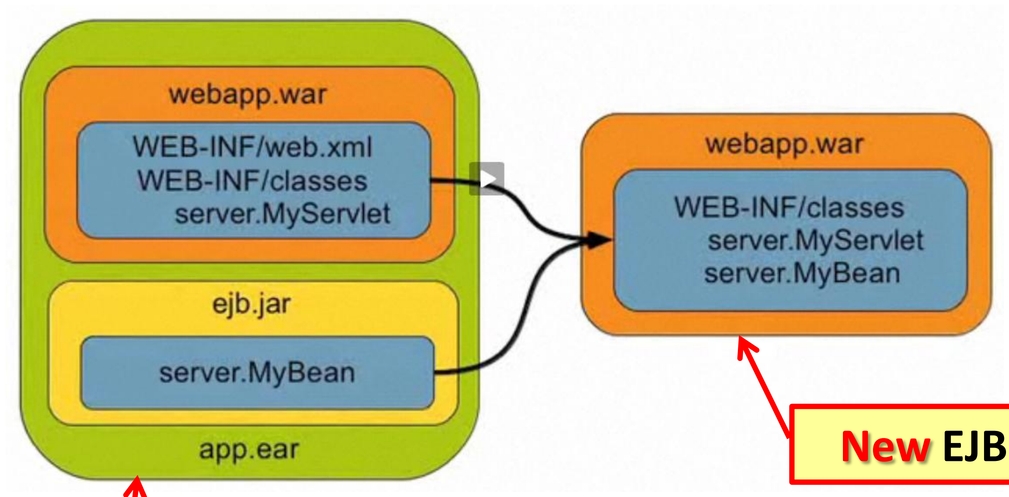
- **JavaBeans** components have properties and have **get** and **set methods** for accessing the **properties**.

Enterprise JavaBeans (EJB)

Enterprise JavaBeans container handles:

- distributed communication
- threading
- scaling
- transaction management, etc.

has a new packaging! (see figure)

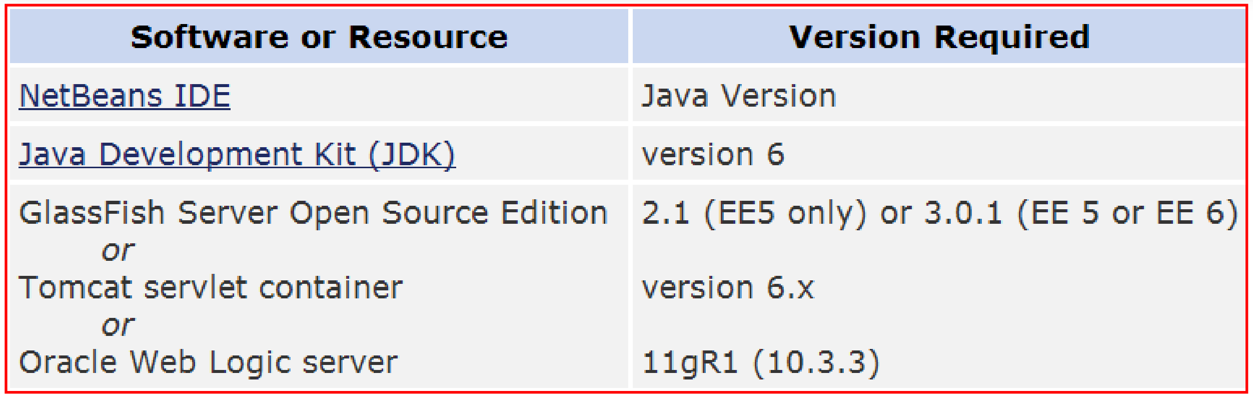


New EJB 3.1 Packaging

Older EJB Packaging

http://www.oracle.com/technetwork/java/deepdivejavaee6glassfishv3-jsp-138230.html

Netbeans IDE



Software or Resource	Version Required
NetBeans IDE	Java Version
Java Development Kit (JDK)	version 6
GlassFish Server Open Source Edition	2.1 (EE5 only) or 3.0.1 (EE 5 or EE 6)
or	
Tomcat servlet container	version 6.x
or	
Oracle Web Logic server	11gR1 (10.3.3)

- **create** a simple web application using **NetBeans IDE**
- **deploy** it to a **server**, and
- **view** its presentation in a **browser**

NetBeans

- A 3rd party **Java Integrated Development Environment** (IDE)
    - Comes with **Java EE class libraries**
    - bundled with GlassFish Sever Open Source Edition
    - Can deploy servlets, JSPs, and web services

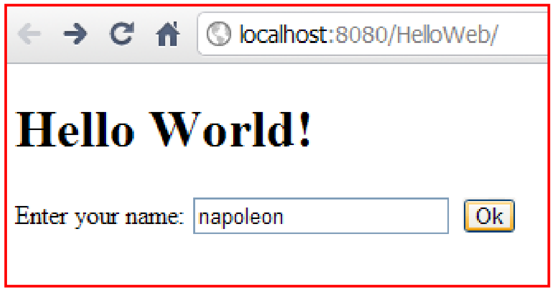
Class libraries for Servlets, JSPs, Enterprise Java Beans, advanced XML

Example: NetBeans Project

A Quick Tour of the IDE (v.6.9)

JSP, Java Bean, User-defined Java Class & Package, Get Method, User Interface

Sample Project


localhost:8080/HelloWeb/

Hello World!

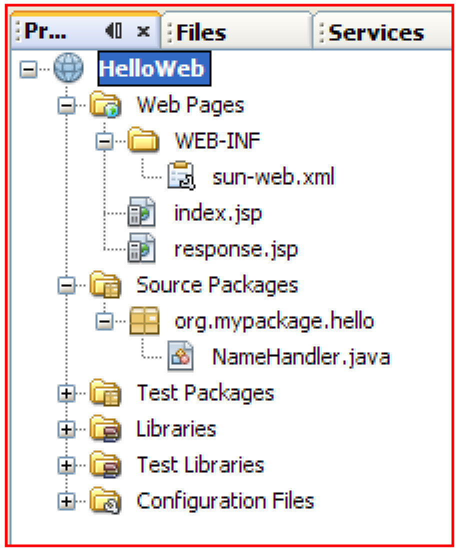
Enter your name: napoleon Ok

Index.jsp

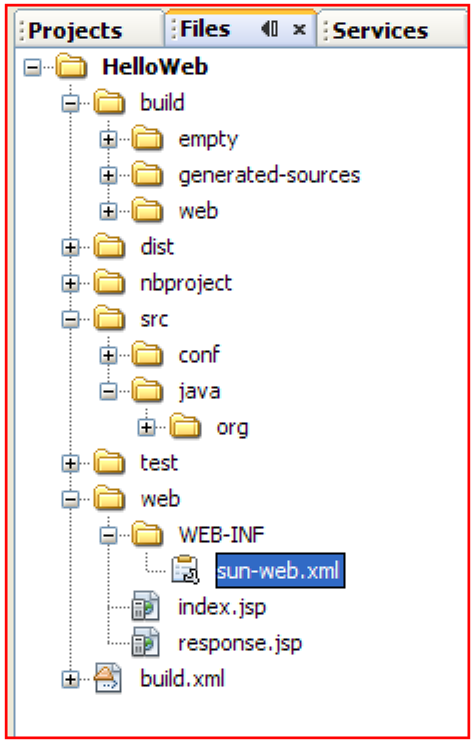
Main interface, Html with form
Invokes **response.jsp** through **form action**.

NameHandler.java

Class NameHandler
containing user data



```
HelloWeb
  Web Pages
    WEB-INF
      sun-web.xml
    index.jsp
    response.jsp
  Source Packages
    org.mypackage.hello
      NameHandler.java
  Test Packages
  Libraries
  Test Libraries
  Configuration Files
```



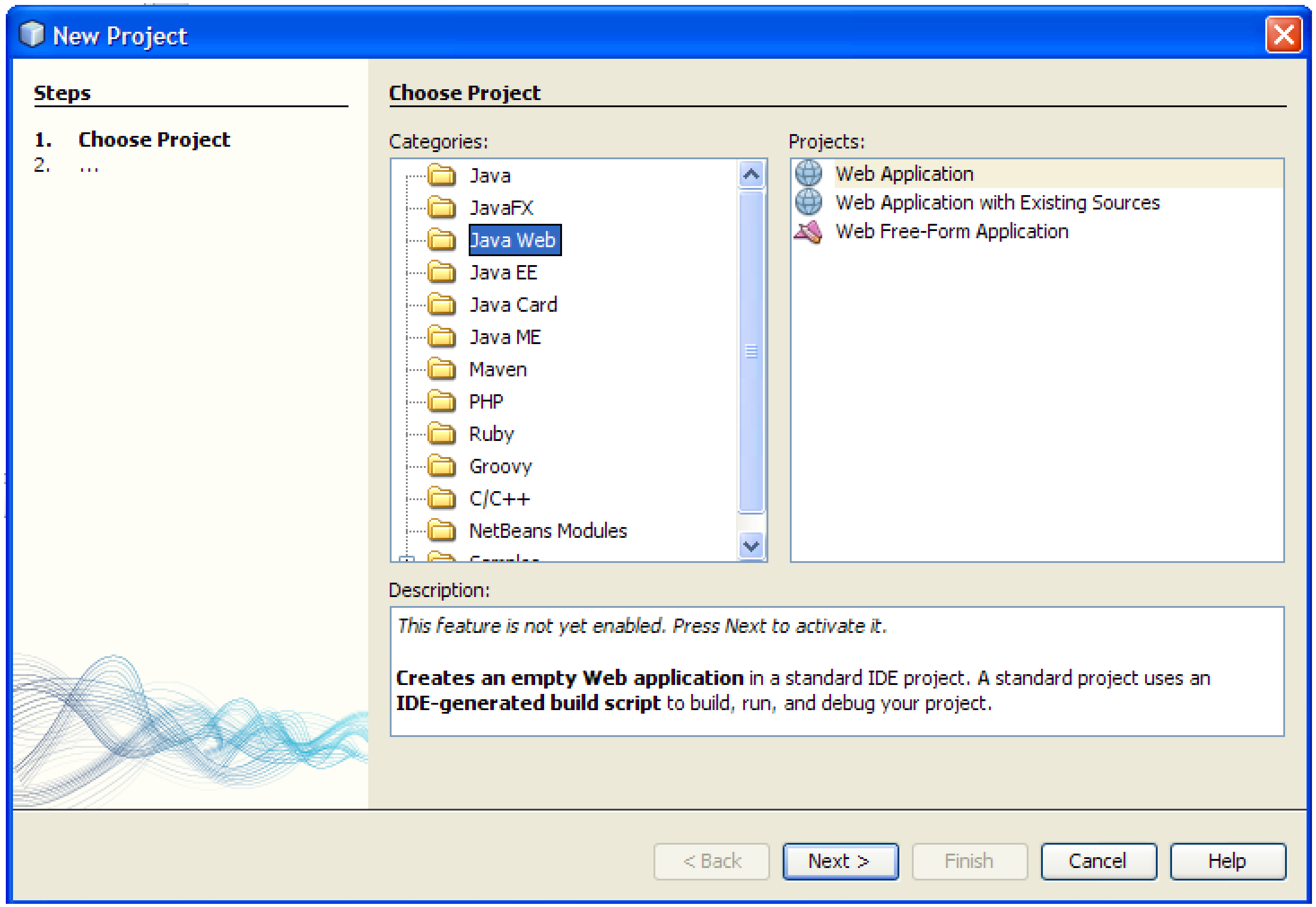
```
HelloWeb
  build
    empty
    generated-sources
    web
  dist
  nbproject
  src
    conf
    java
      org
  test
  web
    WEB-INF
      sun-web.xml
    index.jsp
    response.jsp
  build.xml
```

response.jsp

Generates the server's response
Defines a **JavaBean** to connect the **class NameHandler** to the **user's input** via a **form text field** (name).

Creating a new Web Application

New Project / Java Web

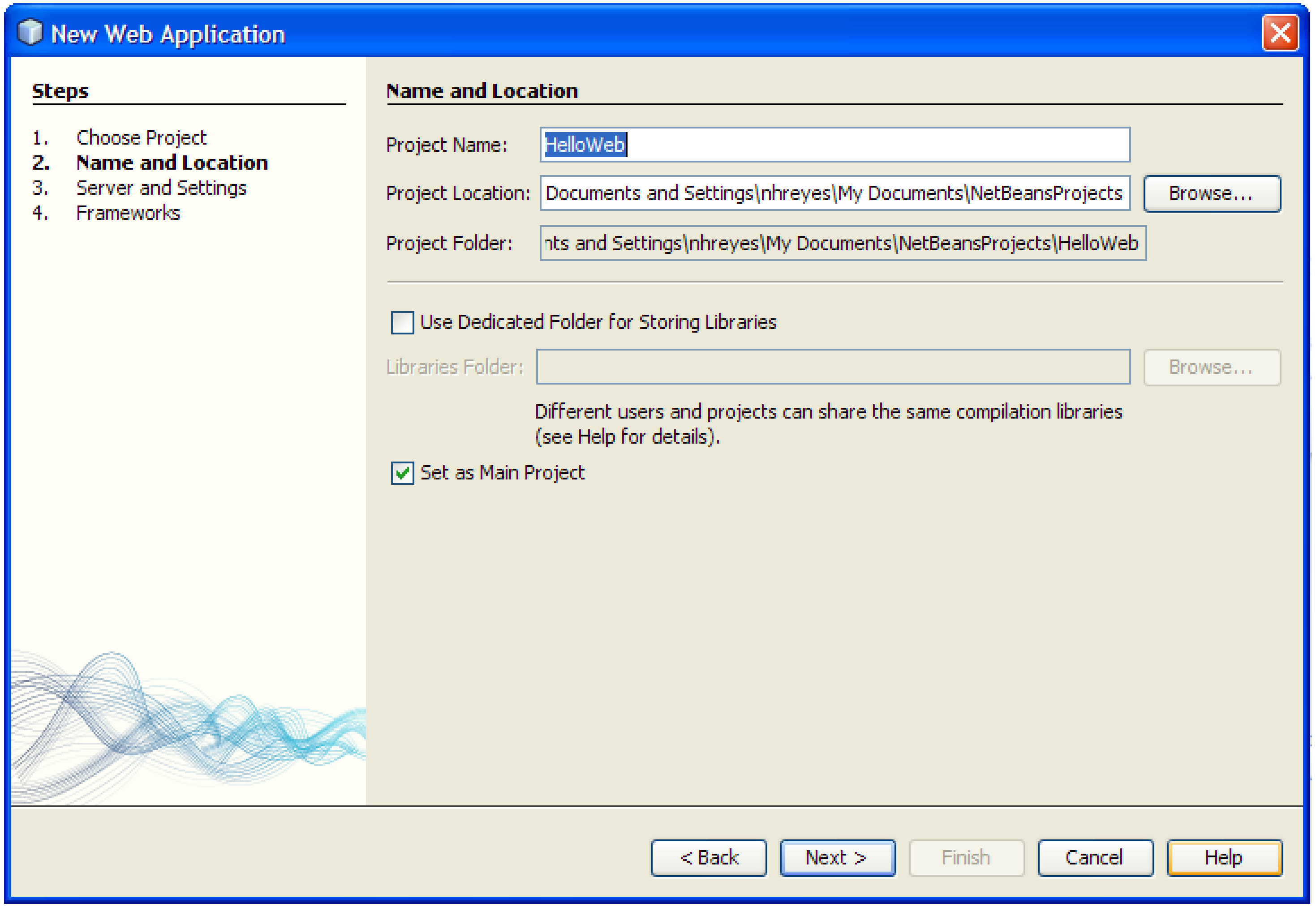


This feature is not yet enabled. Press Next to activate it.

Creates an empty Web application in a standard IDE project. A standard project uses an **IDE-generated build script** to build, run, and debug your project.

Creating a new Web Application

Specify Project Name



New Web Application

Steps

1. Choose Project
2. **Name and Location**
3. Server and Settings
4. Frameworks

Name and Location

Project Name: HelloWeb

Project Location: Documents and Settings\nhreyes\My Documents\NetBeansProjects

Project Folder: nts and Settings\nhreyes\My Documents\NetBeansProjects\HelloWeb

[ ] Use Dedicated Folder for Storing Libraries

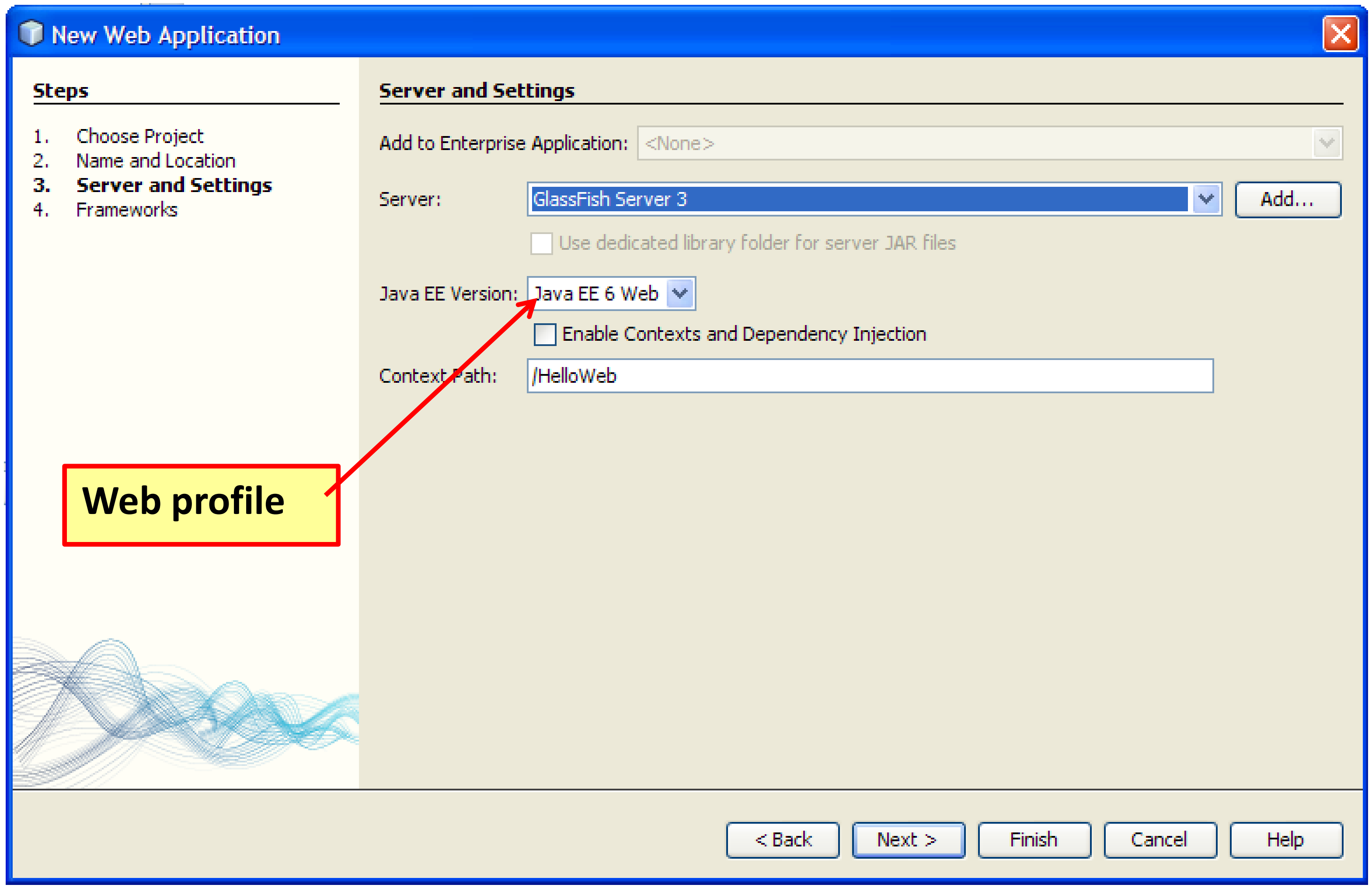
Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

[✓] Set as Main Project

Creating a new Web Application

GlassFish Server



New Web Application

Steps

1. Choose Project
2. Name and Location
3. **Server and Settings**
4. Frameworks

Server and Settings

Add to Enterprise Application: <None>

Server: GlassFish Server 3

Use dedicated library folder for server JAR files

Java EE Version: Java EE 6 Web

Enable Contexts and Dependency Injection

Context Path: /HelloWeb

Web profile

Java Application Server: Glassfish

GlassFish

is an **open source application server** project led by **Sun Microsystems** for the **Java EE** platform. The proprietary version is called Oracle GlassFish Enterprise Server. GlassFish is free software

Sun is the original creator of Tomcat

It uses a **derivative** of **Apache Tomcat** as the **servlet container** for serving Web content, with an added component called **Grizzly** which uses **Java NIO** for scalability and speed.

https://grizzly.dev.java.net/

http://java.dzone.com/articles/glassfish-and-tomcat-whats-the

Before the advent of the Java New I/O API (NIO), thread management issues made it impossible for a server to scale to thousands of users

Java Application Server: Glassfish

GlassFish is an **open source (full) application server project** led by **Sun Microsystems** for the **Java EE** platform. The proprietary version is called Oracle GlassFish Enterprise Server. GlassFish is free software.

It uses a **derivative** of **Apache Tomcat** as the **servlet container** for serving Web content, with an added component called **Grizzly** which uses **Java NIO** for scalability and speed.

On **25 March 2010**, soon **after the acquisition of Sun Microsystems**, **Oracle** issued a Roadmap for versions 3.0.1, 3.1, 3.2 and 4.0 with themes revolving around **clustering**, **virtualization** and **integration** with **Coherence** and other Oracle technologies.

http://en.wikipedia.org/wiki/GlassFish

Glassfish vs. Tomcat

<u>Not</u> a full-application server

Sun is the original creator of Tomcat

Historically, if you wanted to get good HTTP performance from **Tomcat** you really needed to have an **Apache web server** to sit in front of Tomcat which involved more setting up and extra administrative work.

Since **GlassFish v1** (May 2006), **Grizzly** is the **HTTP frontend** of the application server.

It's a 100% **Java NIO framework** that provides the same performance as Apache, only it's written in **Java** and **integrated** straight into the application server.

http://java.dzone.com/articles/glassfish-and-tomcat-whats-the

Other Java web application-capable Servers

- Blazix from Desiderata Software (*1.5 Megabytes, JSP, Servlets and EJBs*)
- TomCat from Apache (*Approx 6 Megabytes*)
- WebLogic from BEA Systems (*Approx 40 Megabytes, JSP, Servlets and EJBs*)
- WebSphere from IBM (*Approx 100 Megabytes, JSP, Servlets and EJBs*)

Commercial Deployment

- **Oracle GlassFish Server**

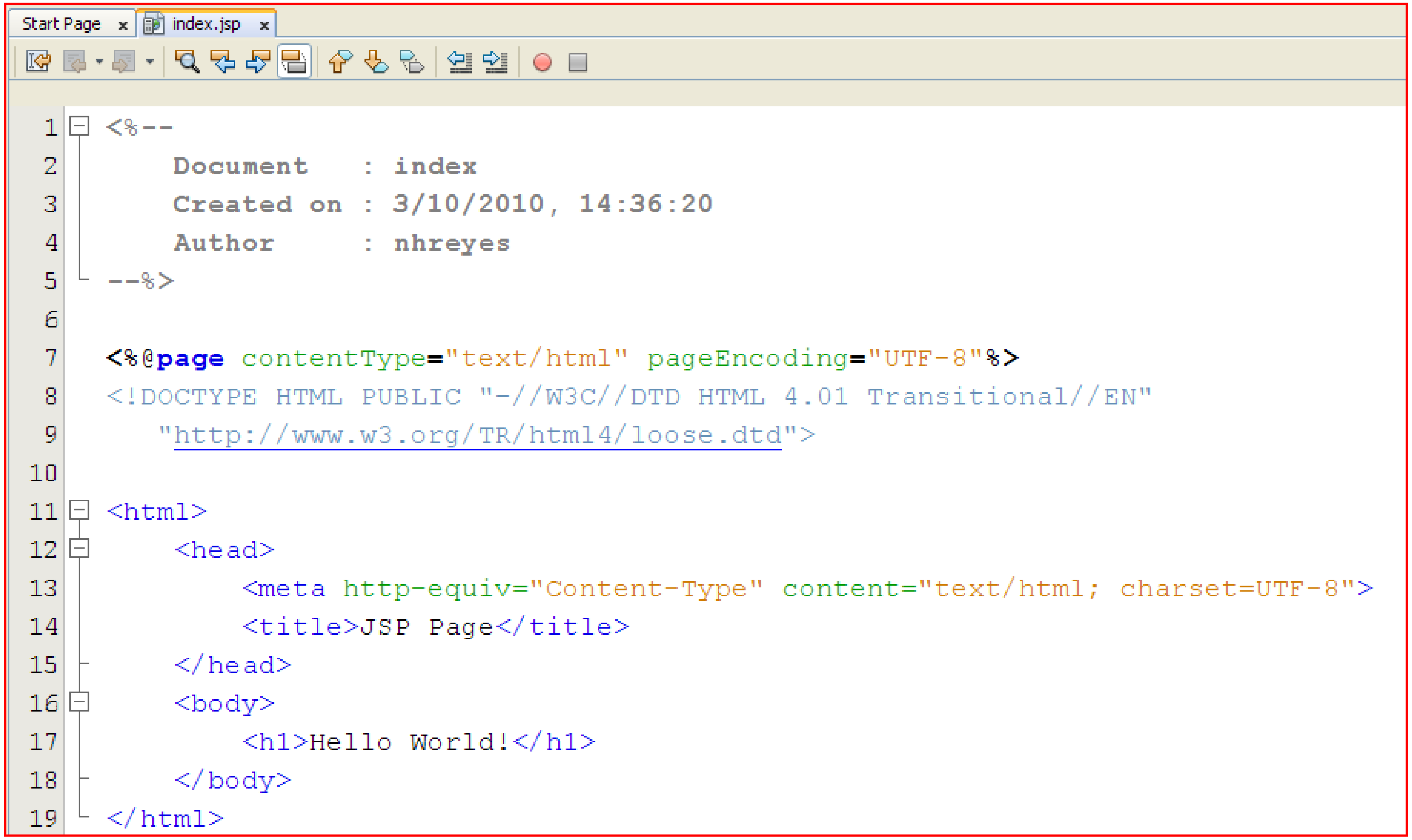
Oracle provides software support <u>only</u> for Oracle GlassFish Server, not for GlassFish Server Open Source Edition

  - delivers a flexible, lightweight and extensible Java EE 6 platform. It provides a small footprint, fully featured Java EE application server that is completely supported for commercial deployment and is available as a standalone offering.
- **Oracle WebLogic Server**
  - designed to run the broader portfolio of Oracle Fusion Middleware and large-scale enterprise applications.
  - industry's most comprehensive Java platform for developing, deploying, and integrating enterprise applications.

http://docs.sun.com/app/docs/doc/821-1751/gkbtb?l=en&a=view

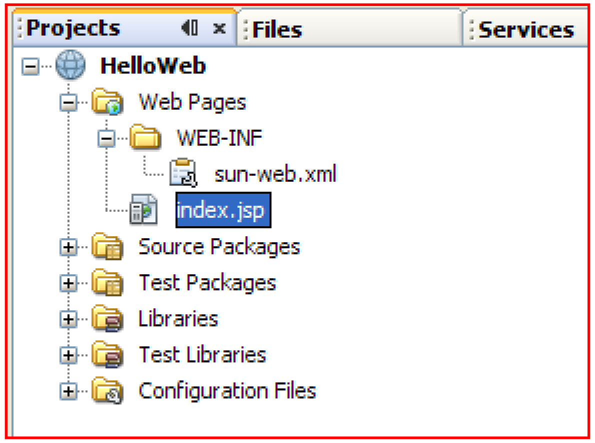
Creating a new Web Application

JSP File

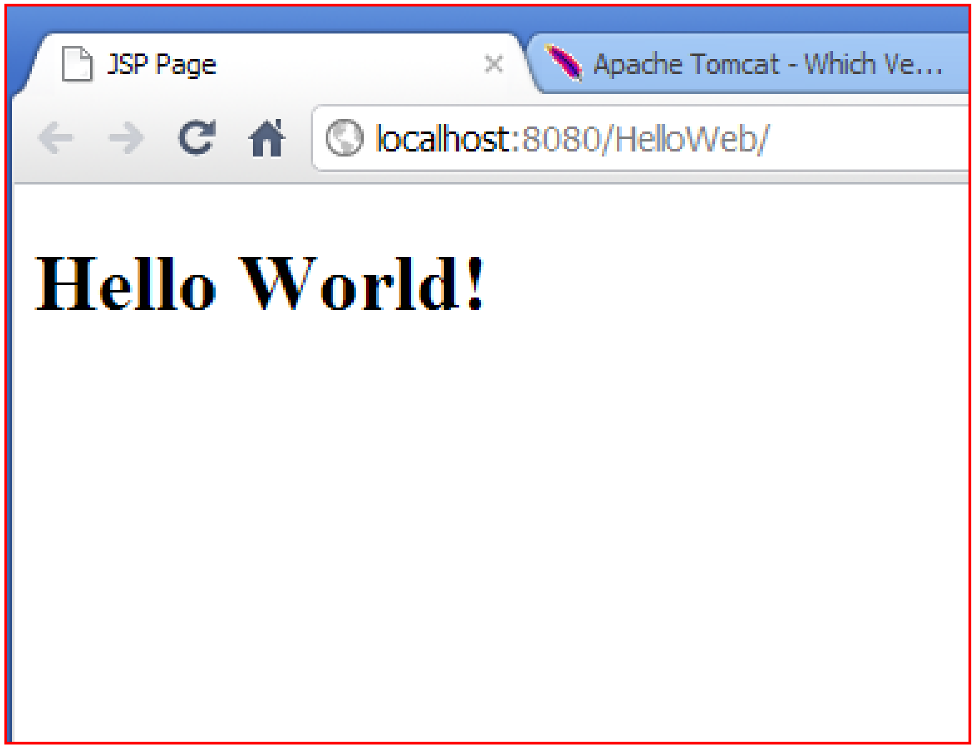


```
<%--
    Document   : index
    Created on : 3/10/2010, 14:36:20
    Author     : nhreyes
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello World!</h1>
    </body>
</html>
```

Creating a new Web Application



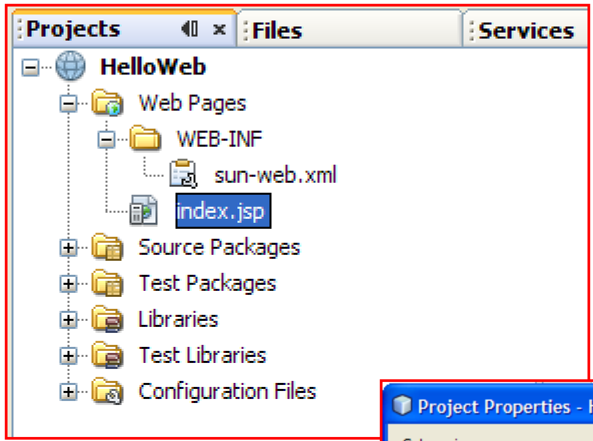
```
HelloWeb
  Web Pages
    WEB-INF
      sun-web.xml
    index.jsp
  Source Packages
  Test Packages
  Libraries
  Test Libraries
  Configuration Files
```



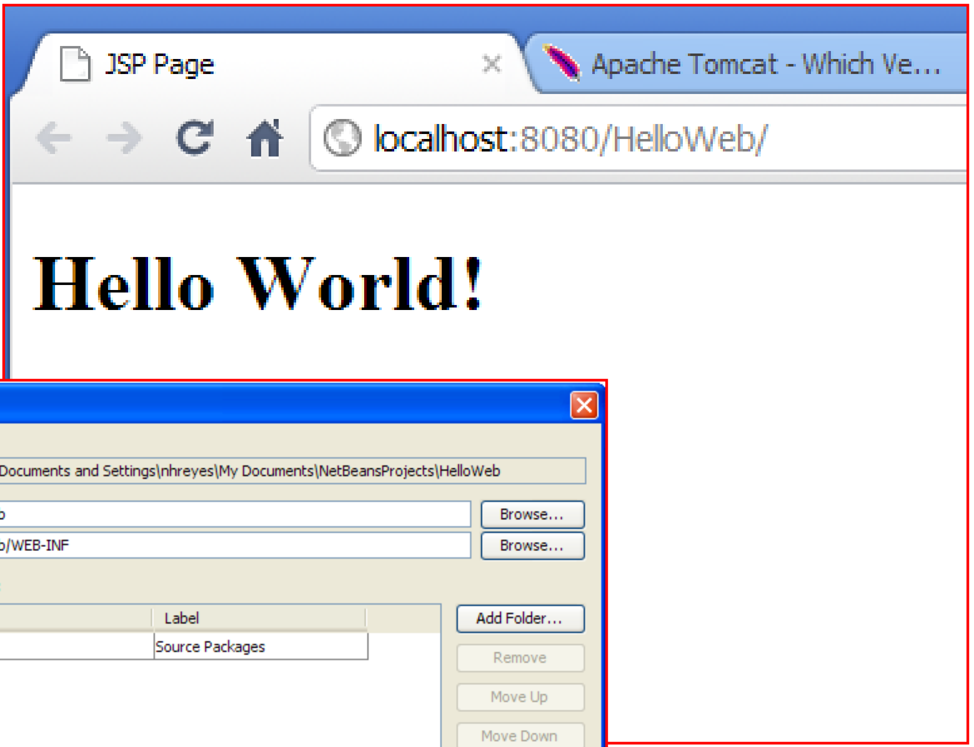
Hello World!

Sample Run

Project: HelloWeb

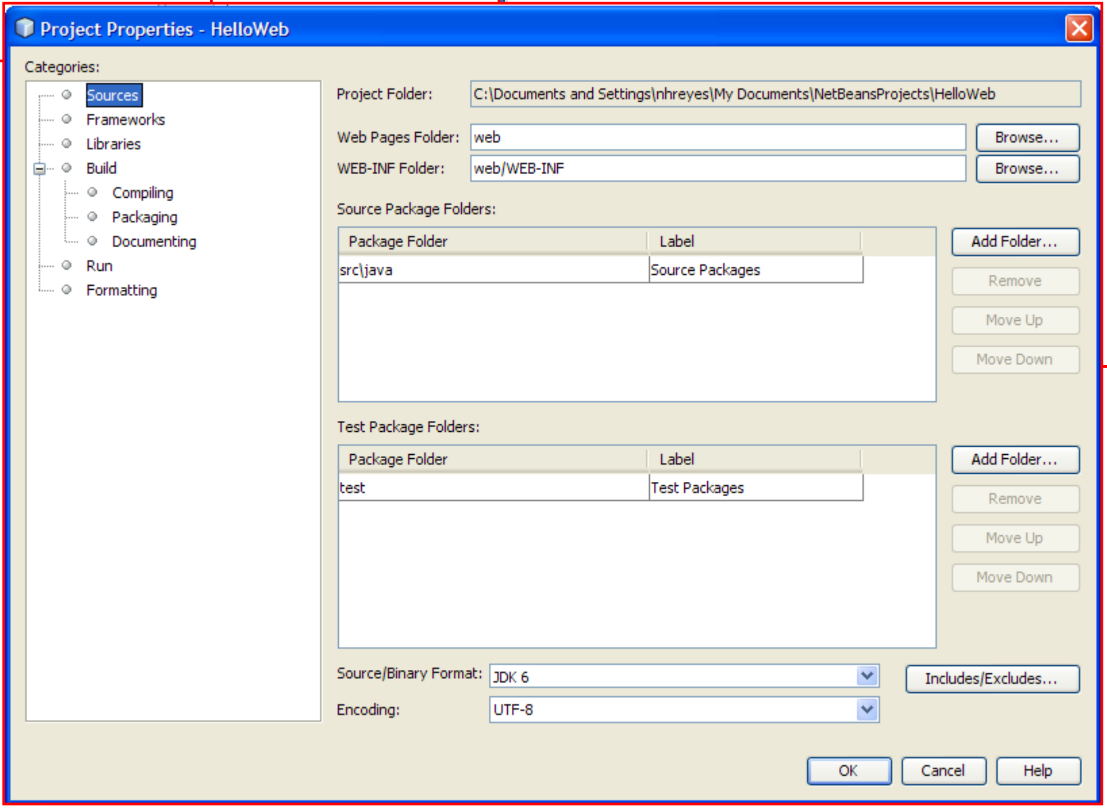


- HelloWeb
  - Web Pages
    - WEB-INF
      - sun-web.xml
    - index.jsp
  - Source Packages
  - Test Packages
  - Libraries
  - Test Libraries
  - Configuration Files



localhost:8080/HelloWeb/

Hello World!



Project Properties - HelloWeb

Categories:

- Sources
- Frameworks
- Libraries
- Build
  - Compiling
  - Packaging
  - Documenting
- Run
- Formatting

Project Folder: C:\Documents and Settings\nhreyes\My Documents\NetBeansProjects\HelloWeb

Web Pages Folder: web

WEB-INF Folder: web/WEB-INF

Source Package Folders:

Package Folder	Label
src\java	Source Packages

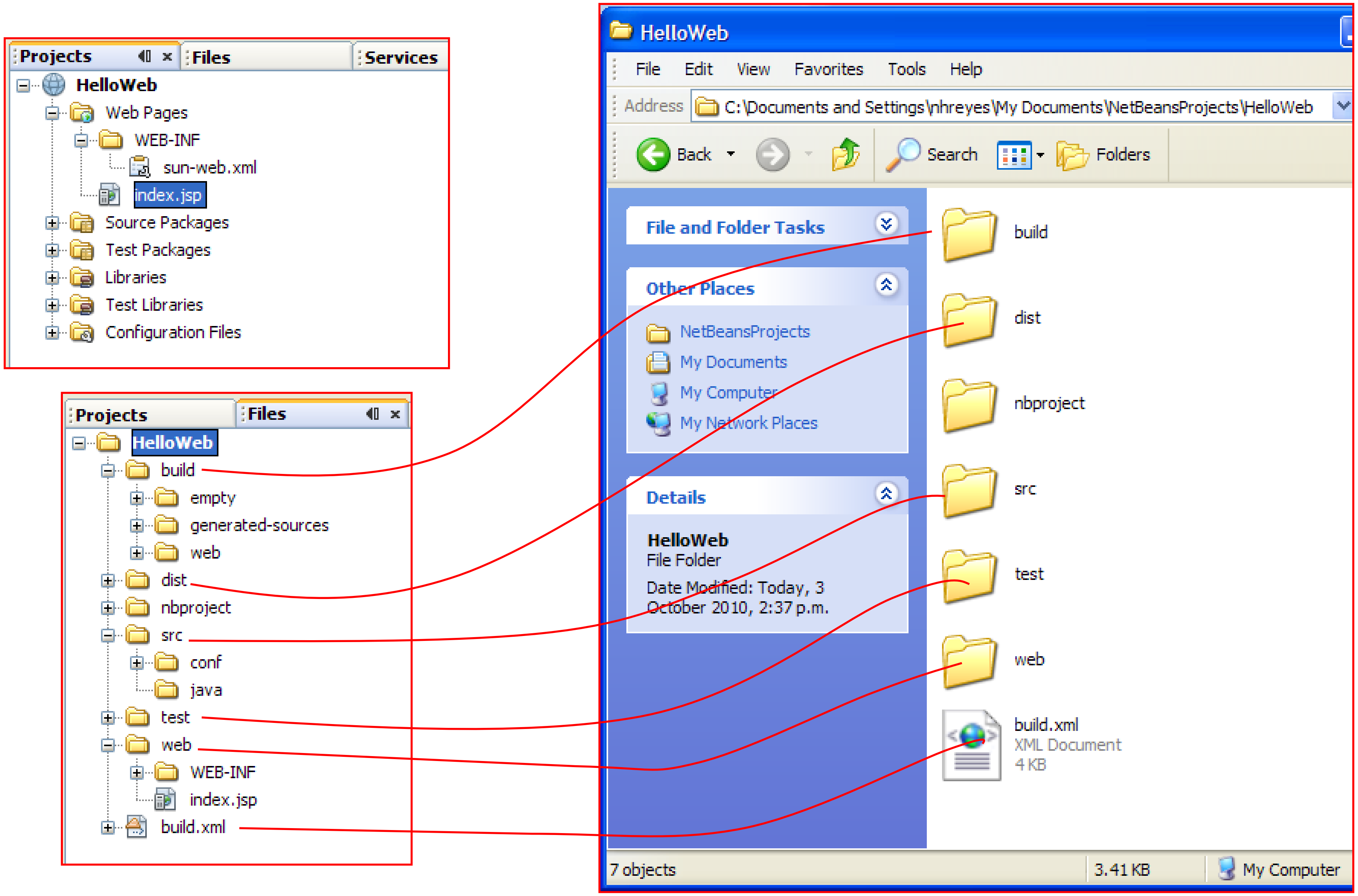
Test Package Folders:

Package Folder	Label
test	Test Packages

Source/Binary Format: JDK 6

Encoding: UTF-8

HelloWeb: Directories and Files



Projects | Files | Services

```
HelloWeb
  Web Pages
    WEB-INF
      sun-web.xml
    index.jsp
  Source Packages
  Test Packages
  Libraries
  Test Libraries
  Configuration Files
```

Projects | Files

```
HelloWeb
  build
    empty
    generated-sources
    web
  dist
  nbproject
  src
    conf
    java
  test
  web
    WEB-INF
    index.jsp
  build.xml
```

HelloWeb

File Edit View Favorites Tools Help

Address C:\Documents and Settings\nhreyes\My Documents\NetBeansProjects\HelloWeb

Back Search Folders

File and Folder Tasks

Other Places

- NetBeansProjects
- My Documents
- My Computer
- My Network Places

Details

HelloWeb
File Folder
Date Modified: Today, 3 October 2010, 2:37 p.m.

- build
- dist
- nbproject
- src
- test
- web
- build.xml
  XML Document
  4 KB

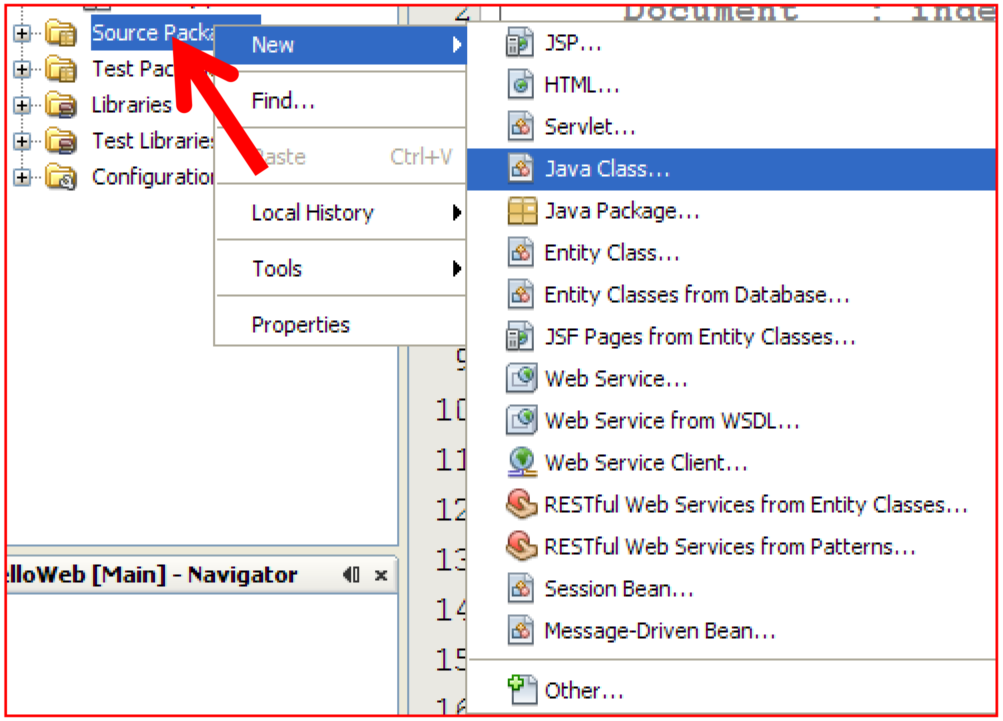
7 objects | 3.41 KB | My Computer

Adding a Java source package and a source file

NameHandler.java

Java Package

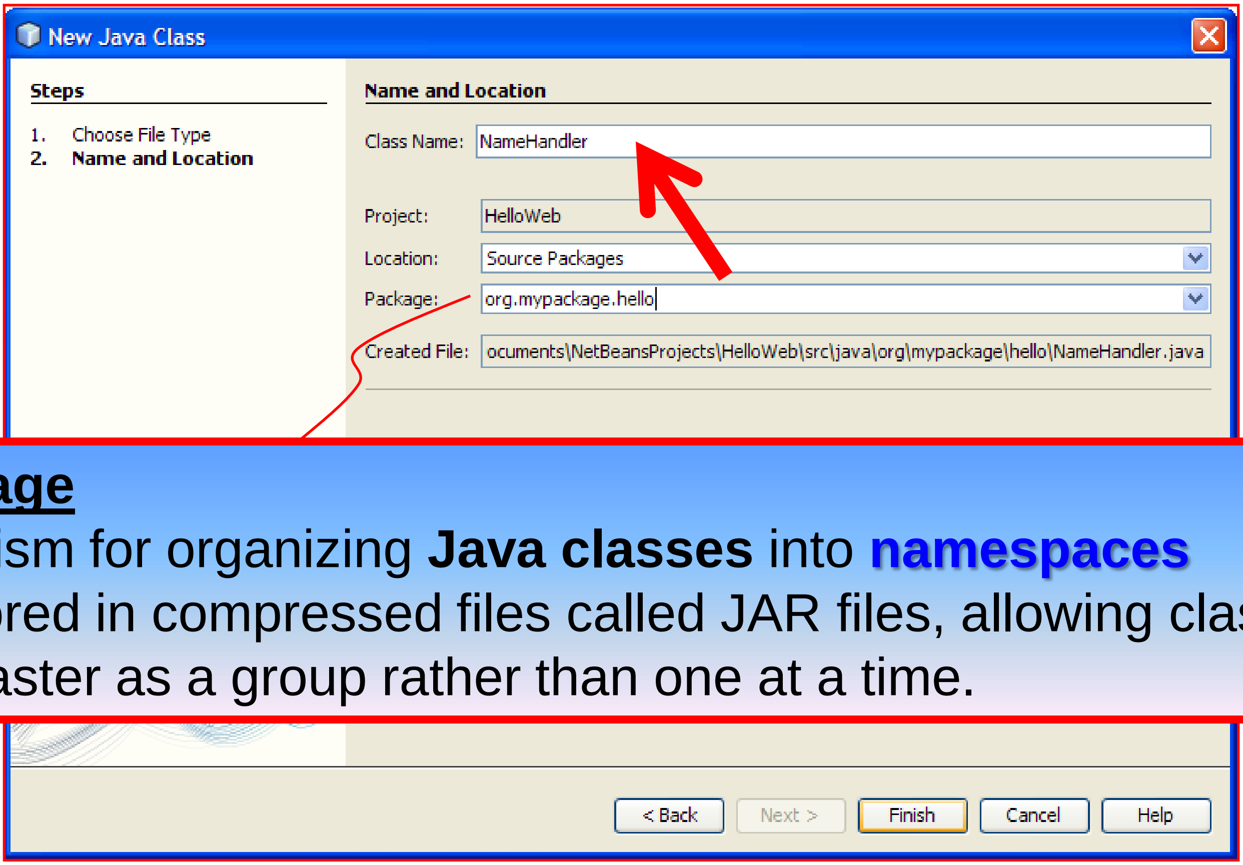
Right-click Source Packages



http://en.wikipedia.org/wiki/GlassFish

Java Package

Add a Java Class, specify Package name



Name and Location

Class Name: NameHandler

Project: HelloWeb

Location: Source Packages

Package: org.mypackage.hello

Created File: ocuments\NetBeansProjects\HelloWeb\src\java\org\mypackage\hello\NameHandler.java

Java Package

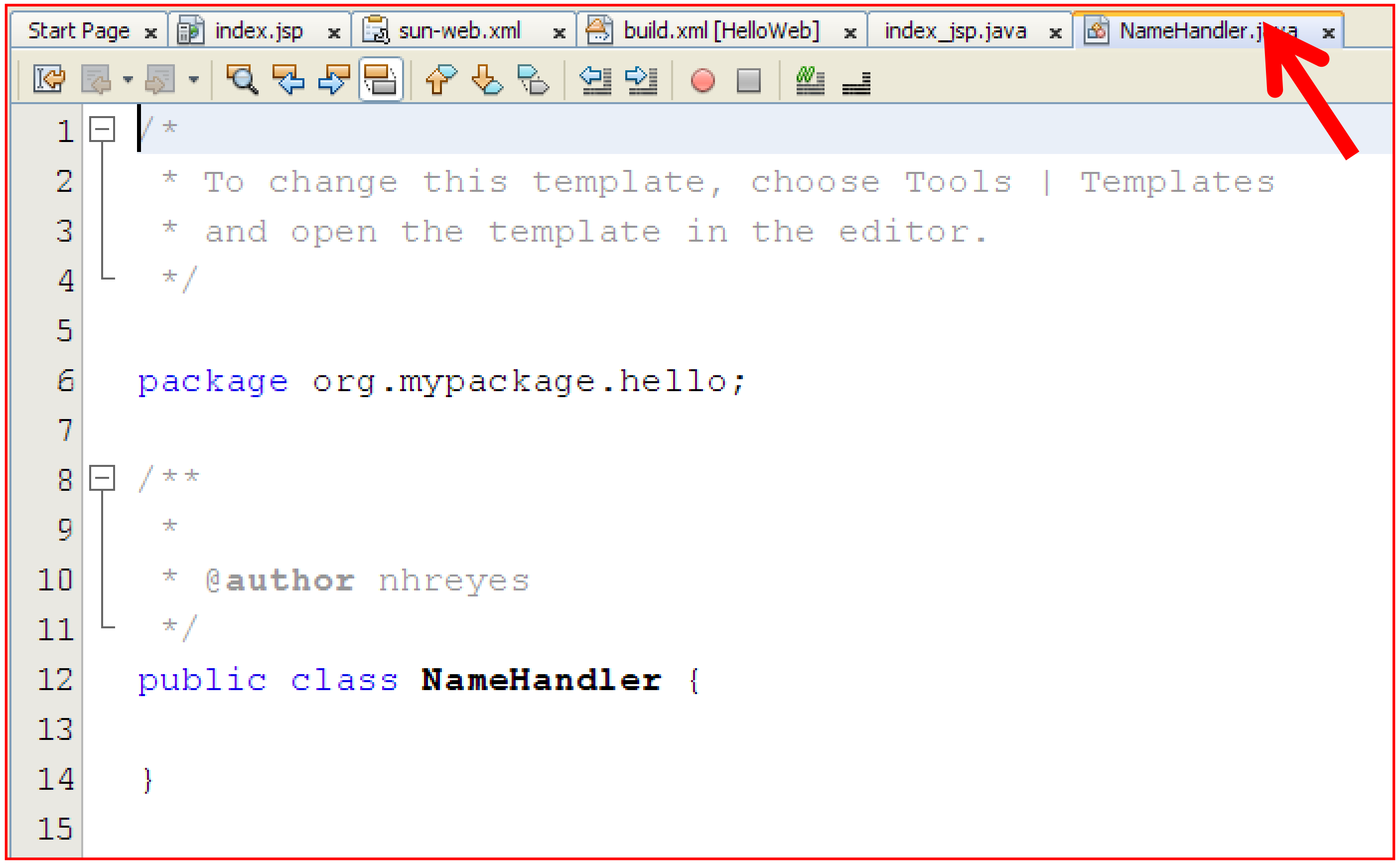
• a mechanism for organizing **Java classes** into **namespaces**

• can be stored in compressed files called JAR files, allowing classes to download faster as a group rather than one at a time.

http://en.wikipedia.org/wiki/GlassFish

Java Package

Add a Java Class

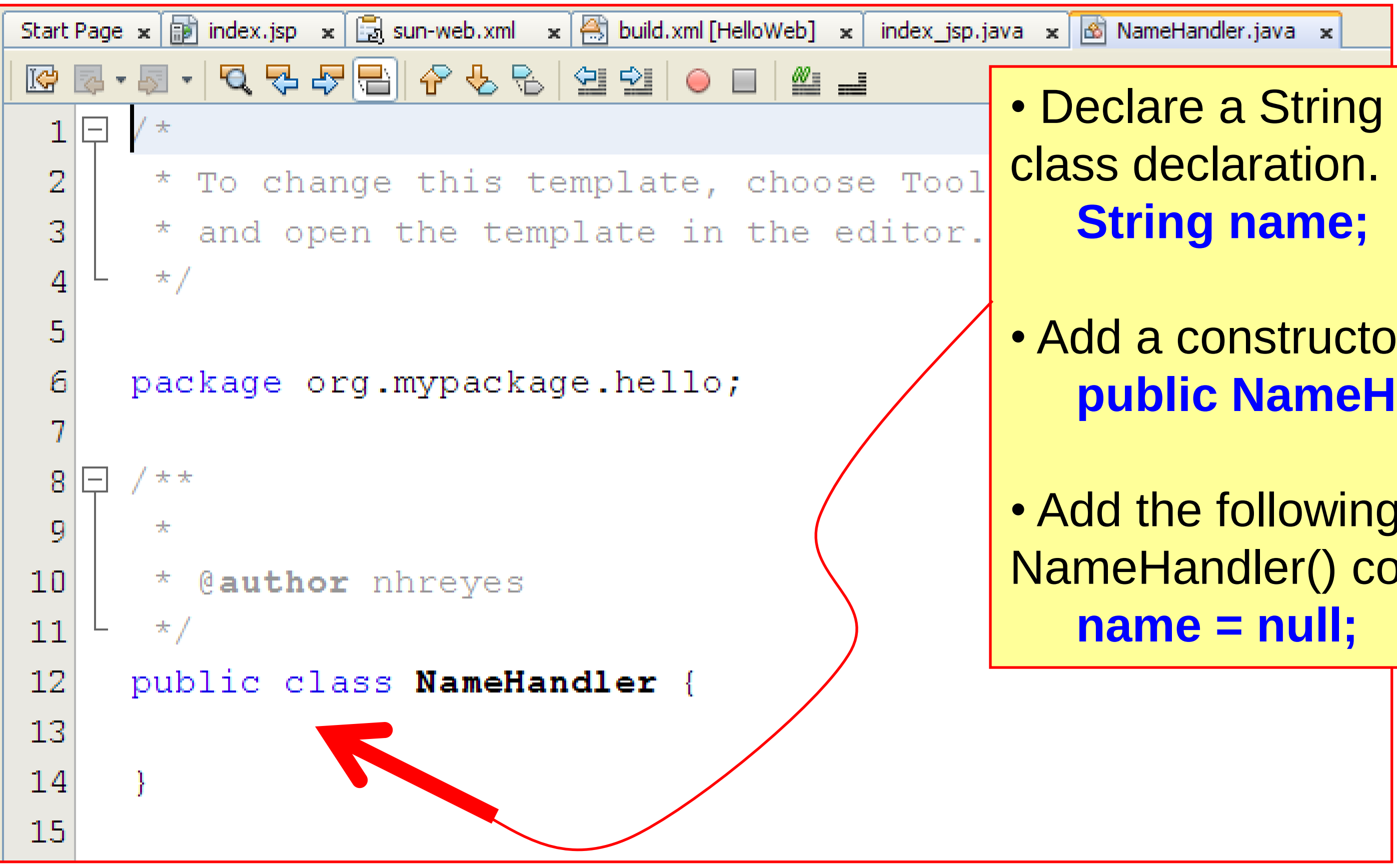


```java
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package org.mypackage.hello;

/**
 *
 * @author nhreyes
 */
public class NameHandler {

}

```

http://en.wikipedia.org/wiki/GlassFish

Java Package

Edit the Java Class



```
/*
 * To change this template, choose Tool
 * and open the template in the editor.
 */

package org.mypackage.hello;

/**
 *
 * @author nhreyes
 */
public class NameHandler {

}

```

• Declare a String variable inside the class declaration.

String name;

• Add a constructor to the class:

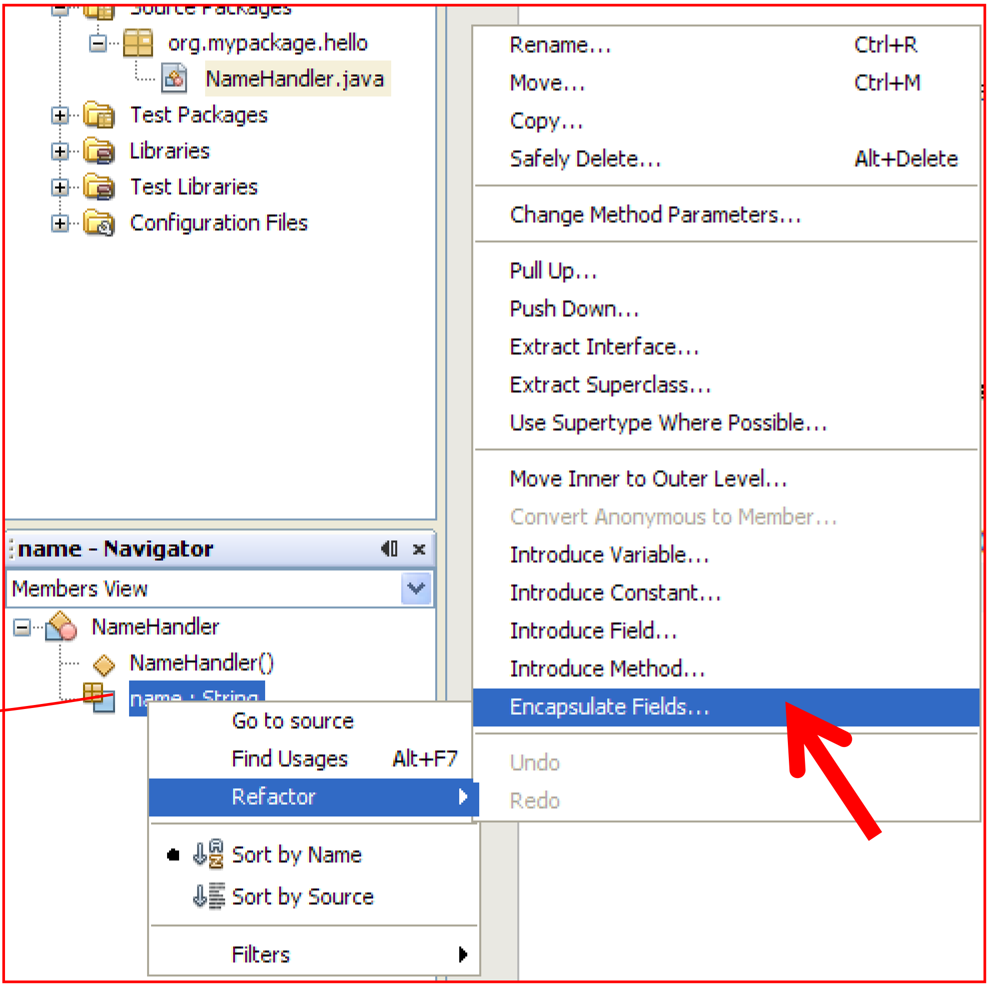
public NameHandler()

• Add the following line in the NameHandler() constructor:

name = null;

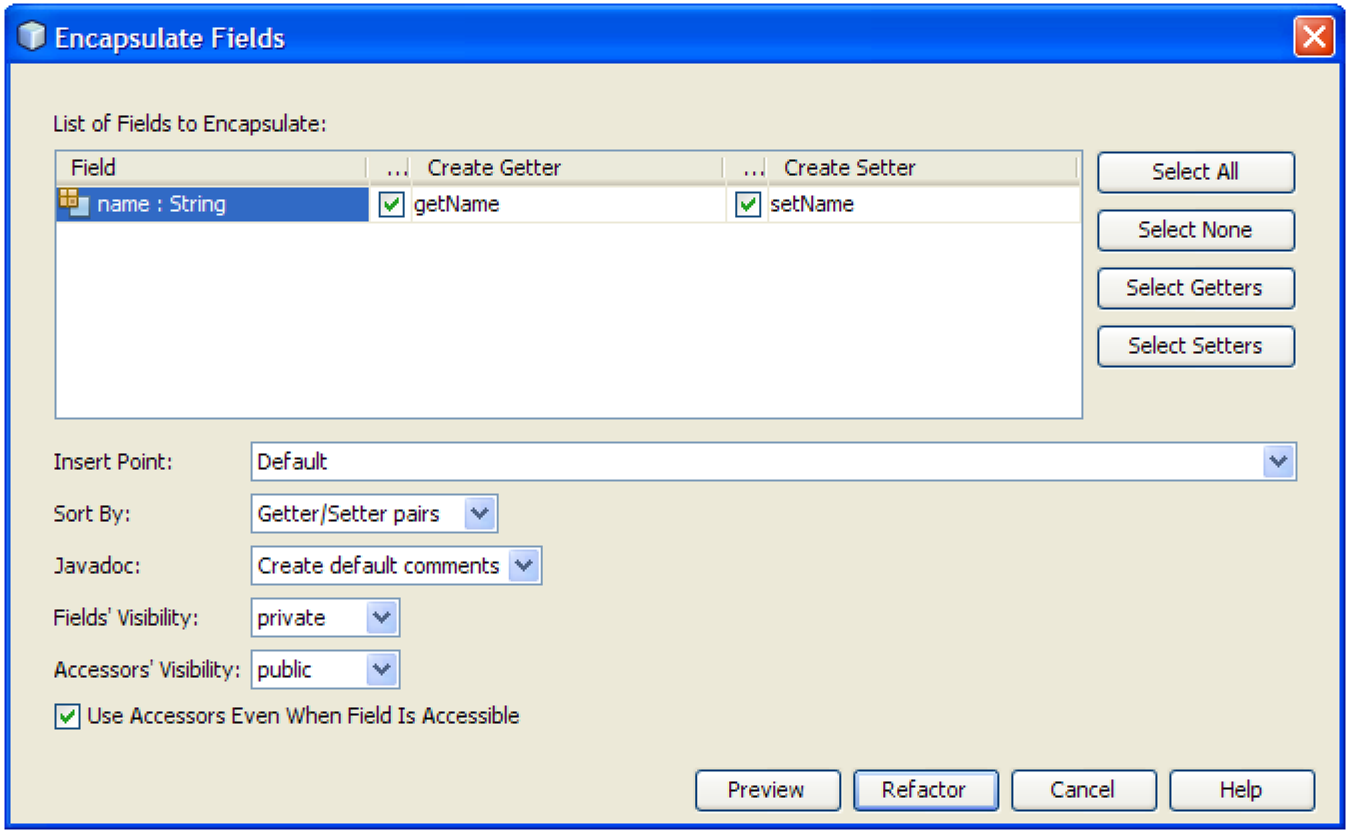
Generating Getter and Setter Methods

Right-click **name field** in the Source editor



Selection: **Name Field / Refactor / Encapsulate Fields**

Generating Getter and Setter Methods



List of Fields to Encapsulate:

Field	...	Create Getter	...	Create Setter
name : String	✓	getName	✓	setName

Insert Point: Default

Sort By: Getter/Setter pairs

Javadoc: Create default comments

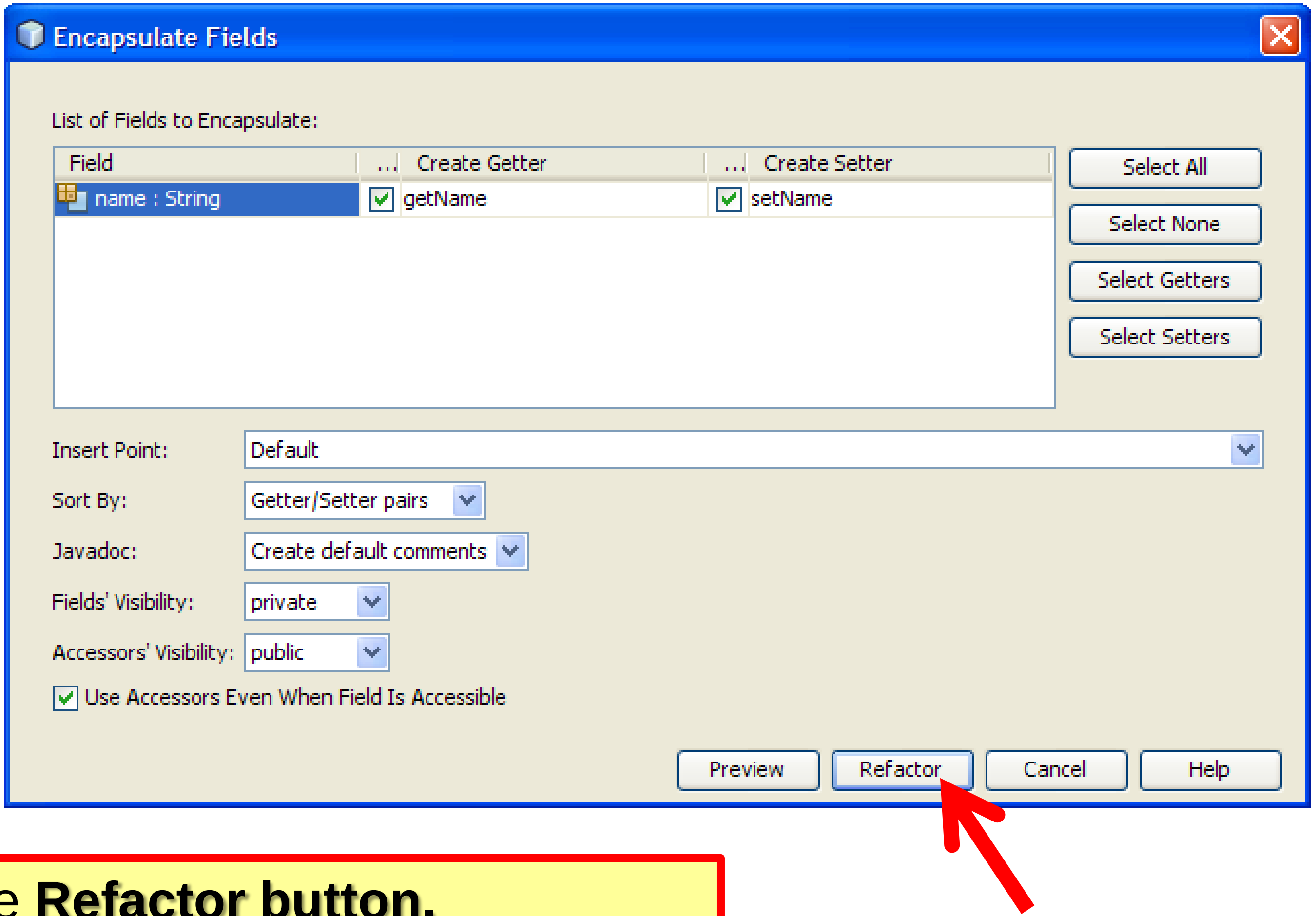
Fields' Visibility: private

Accessors' Visibility: public

✓ Use Accessors Even When Field Is Accessible

Notice that Fields' **Visibility** is by default set to **private**, and Accessors' Visibility to public, indicating that the **access modifier** for class variable declaration will be specified as **private**, whereas **getter** and **setter methods** will be generated with **public** and **private** modifiers, respectively.

Generating Getter and Setter Methods



Encapsulate Fields

List of Fields to Encapsulate:

Field	... Create Getter	... Create Setter
name : String	✓ getName	✓ setName

Insert Point: Default

Sort By: Getter/Setter pairs

Javadoc: Create default comments

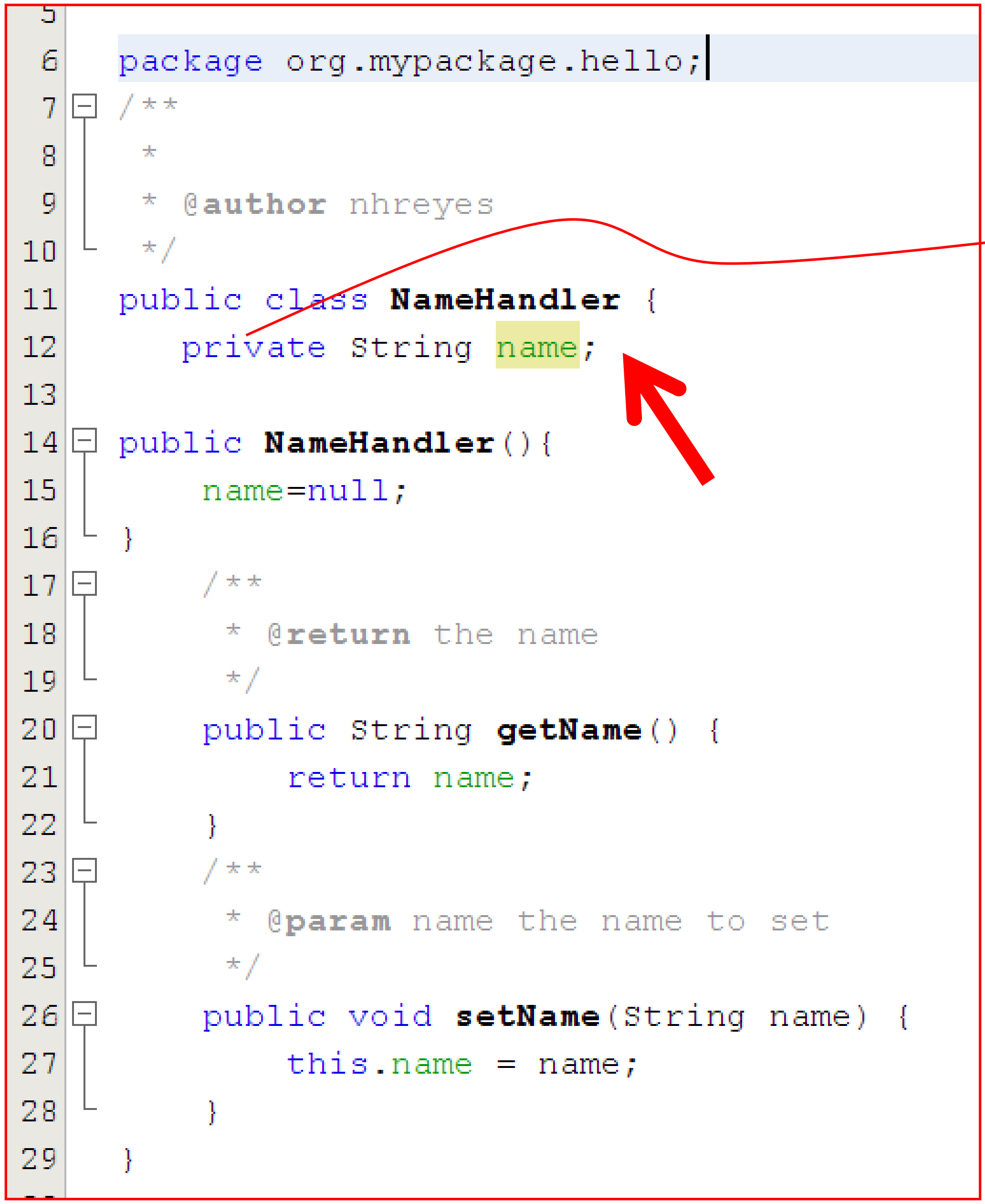
Fields' Visibility: private

Accessors' Visibility: public

✓ Use Accessors Even When Field Is Accessible

Select the **Refactor button.**

Results of Refactoring



```
package org.mypackage.hello;
/**
 *
 * @author nhreyes
 */
public class NameHandler {
   private String name;

public NameHandler(){
    name=null;
}
    /**
     * @return the name
     */
    public String getName() {
        return name;
    }
    /**
     * @param name the name to set
     */
    public void setName(String name) {
        this.name = name;
    }
}
```

Notice that the variable declaration has changed.
- set to **private**

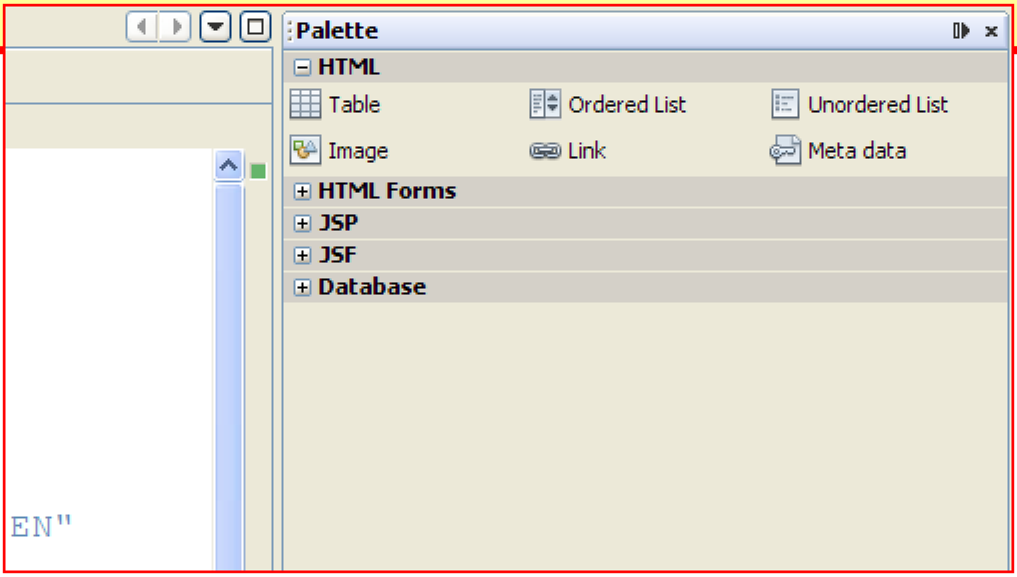
Get and **set functions** with implementation have been added as well.
- access modifier: **public**

Editing the Default JSP file

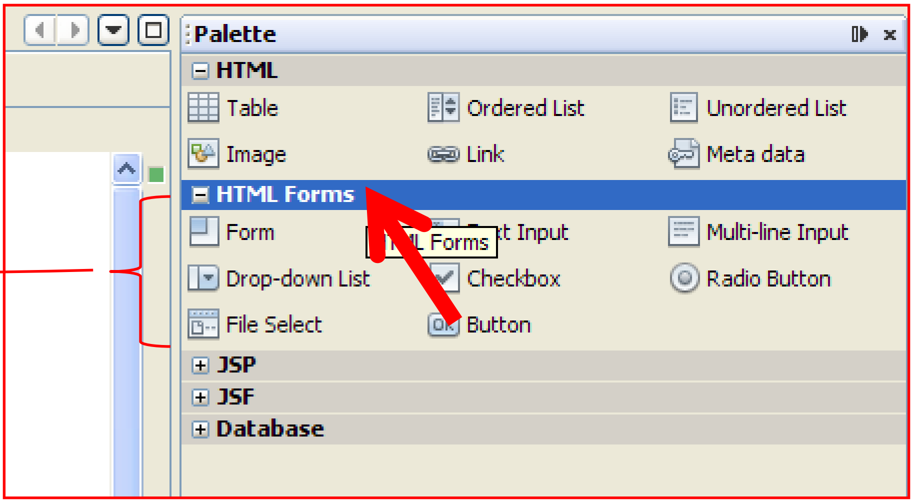
Adding and Customising a Form, input text field, submit button

Inserting a Form

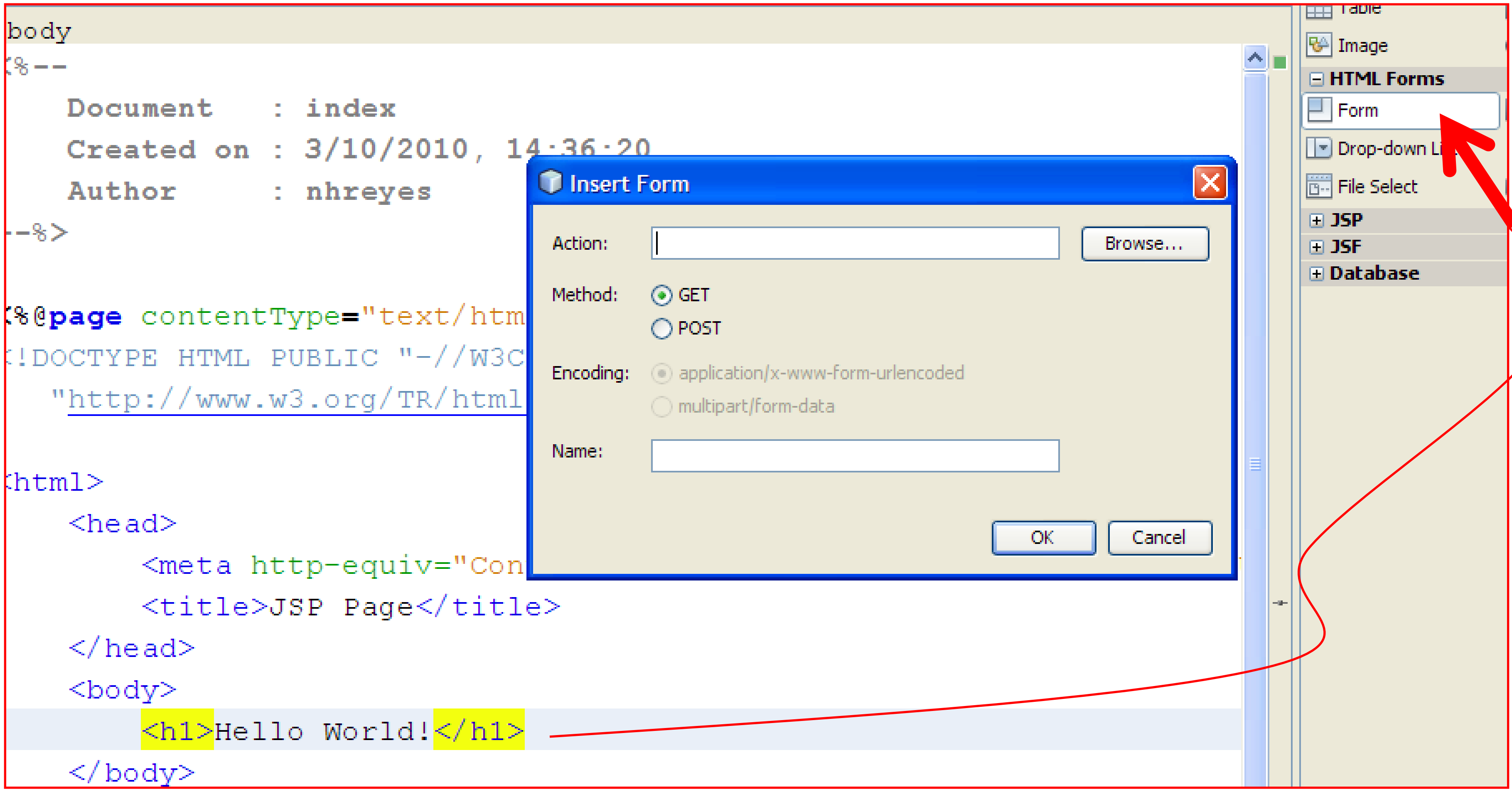
Invoke the **palette**: from the menu, select (Window/Palette): or press Ctrl+Shift+8



expand HTML Forms



Inserting a Form



```
body
<%--
    Document   : index
    Created on : 3/10/2010, 14:36:20
    Author     : nhreyes
--%>

<%@page contentType="text/htm
<!DOCTYPE HTML PUBLIC "-//W3C
   "http://www.w3.org/TR/html

<html>
    <head>
        <meta http-equiv="Con
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello World!</h1>
    </body>
```

Insert Form

Action:

Method: [✓ GET] POST

Encoding: [✓ application/x-www-form-urlencoded] multipart/form-data

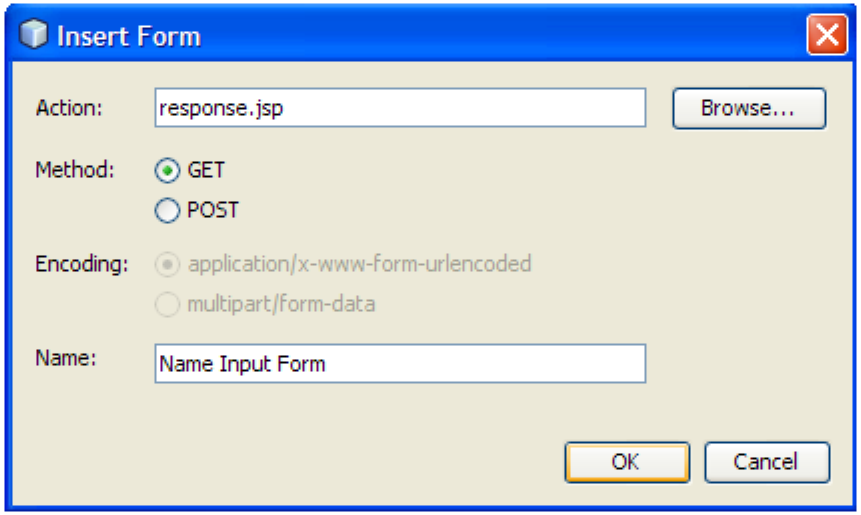
Name:

expand HTML Forms and drag a **Form item** to a point after the **<h1>** tags in the Source Editor.

The Insert Form dialog box displays.

Specifying an action

Specify the following values:



Action: response.jsp

Method: [✓ GET] POST

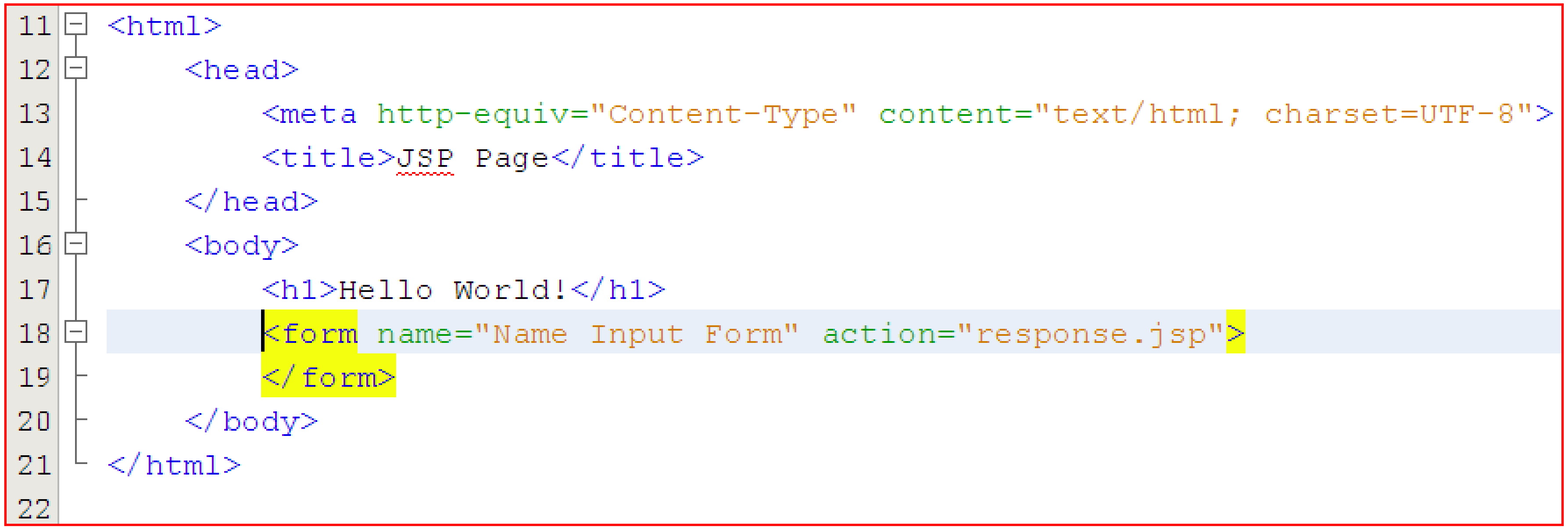
Encoding: [✓ application/x-www-form-urlencoded] multipart/form-data

Name: Name Input Form

Click OK.

Source Generated

An HTML form is automatically added to the index.jsp file.

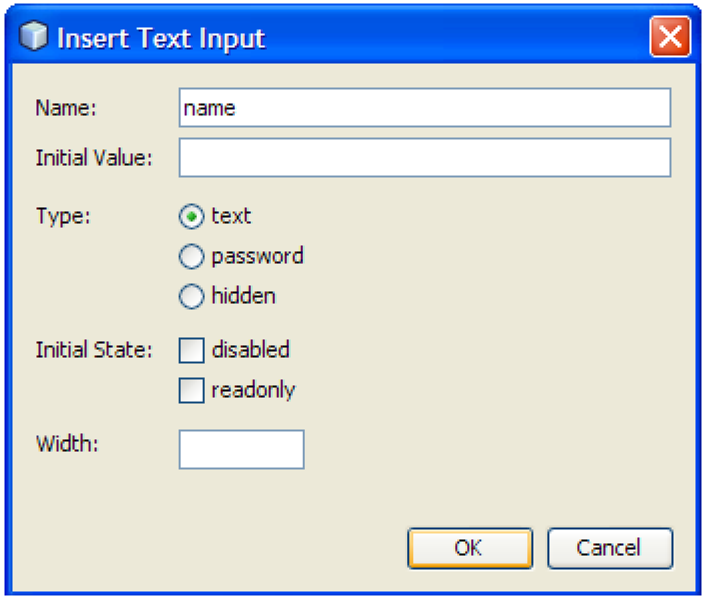


```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello World!</h1>
        <form name="Name Input Form" action="response.jsp">
        </form>
    </body>
</html>
```

Adding an Input Text Field

Drag a Text Input item to a point just before the </form> tag, then specify the following values:

- **Name**: name
- **Type**: text



Name: name

Initial Value:

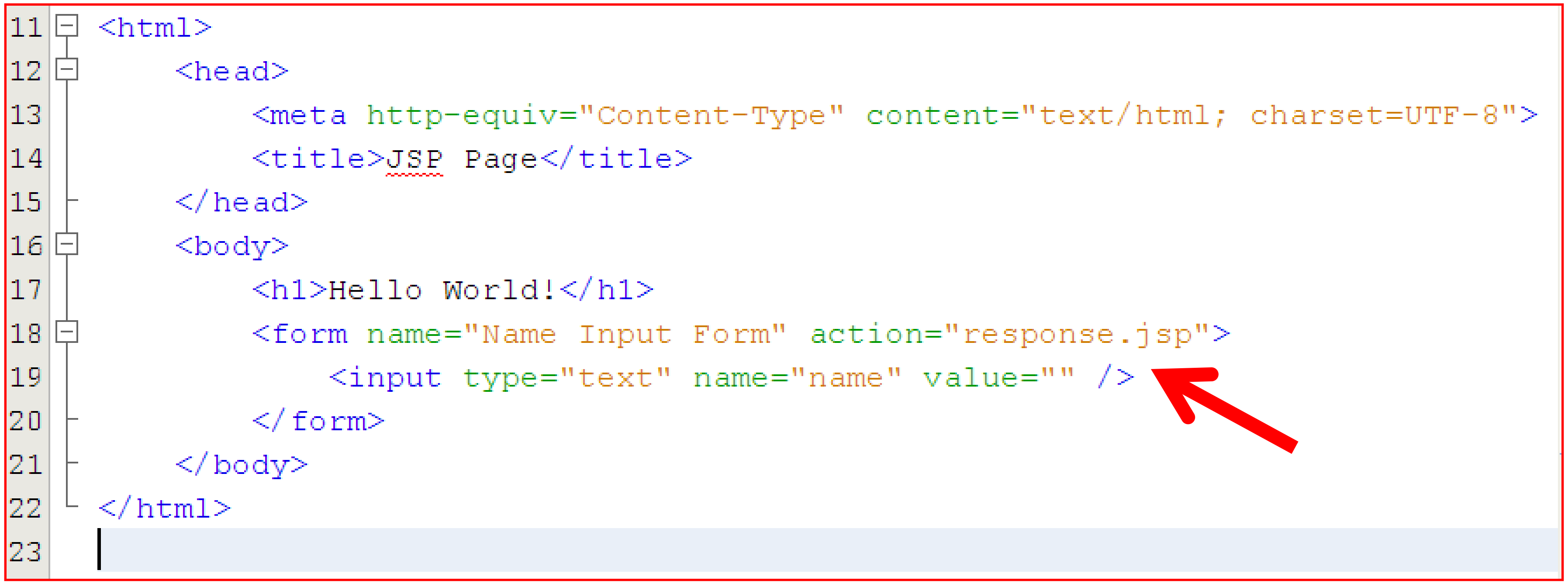
Type: [✓ text] password hidden

Initial State: disabled readonly

Width:

Source Generated

Input Text Field



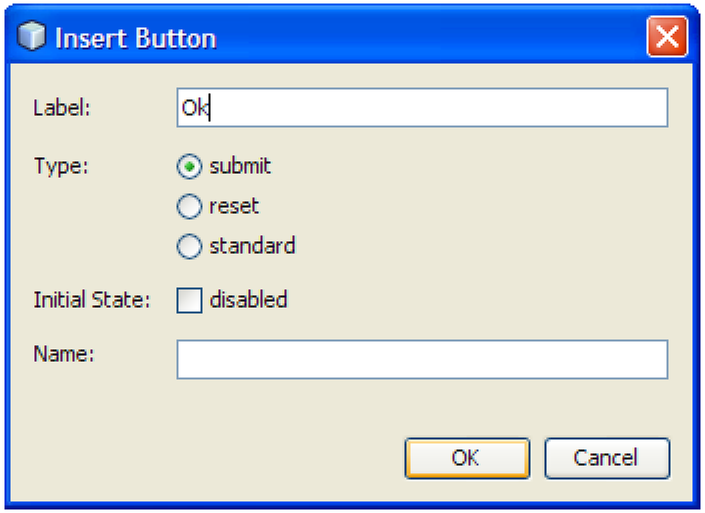
```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello World!</h1>
        <form name="Name Input Form" action="response.jsp">
            <input type="text" name="name" value="" />
        </form>
    </body>
</html>
```

Adding a Submit Button

Drag a Button item to a point just before the </form> tag. Specify the following values:

- **Label: OK**
- **Type: submit**

Click OK. An HTML button is added between the <form> tags.


Insert Button

Label: Ok

Type: [✓ submit] reset standard

Initial State: ☐ disabled

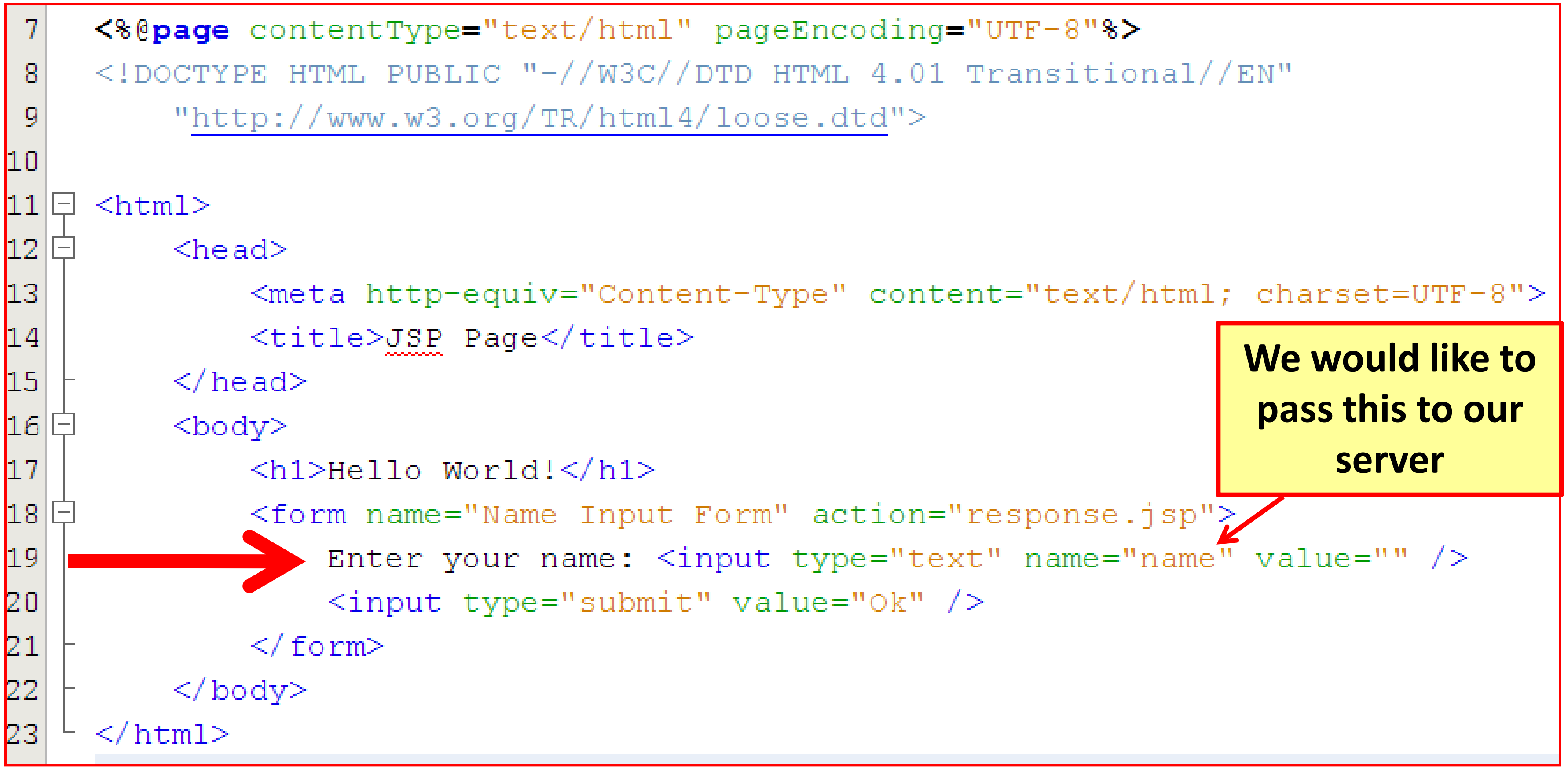
Name:

Adding some extra labels, tidying up your code

Type Enter your name: just before the first <input> tag, then change the default Hello World! text between the <h1> tags to Entry Form.

Right-click within the Source Editor and choose Format (Alt-Shift-F) to tidy the format of your code.

index.jsp: Source Generated



```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello World!</h1>
        <form name="Name Input Form" action="response.jsp">
            Enter your name: <input type="text" name="name" value="" />
            <input type="submit" value="Ok" />
        </form>
    </body>
</html>
```

We would like to pass this to our server

Creating a JSP file that generates the server’s response

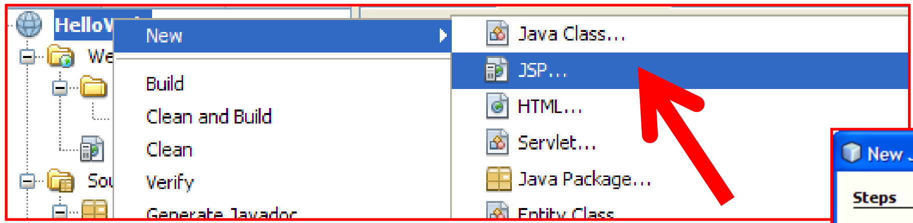
response.jsp

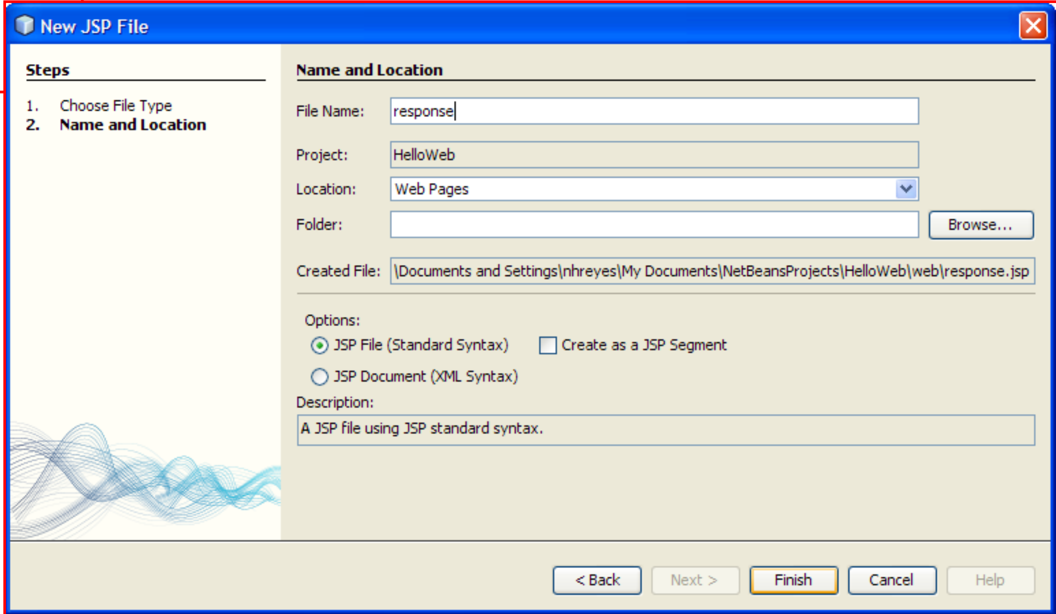
Adding a JSP File

In the Projects window, right-click the **HelloWeb** project node and choose **New > JSP**. The New JSP File wizard opens.

Name the **file response**, and **click Finish**.

Notice that a response.jsp file node displays in the Projects window beneath index.jsp, and the **new file opens in the Source Editor**.




New JSP File

Steps
1. Choose File Type
2. **Name and Location**

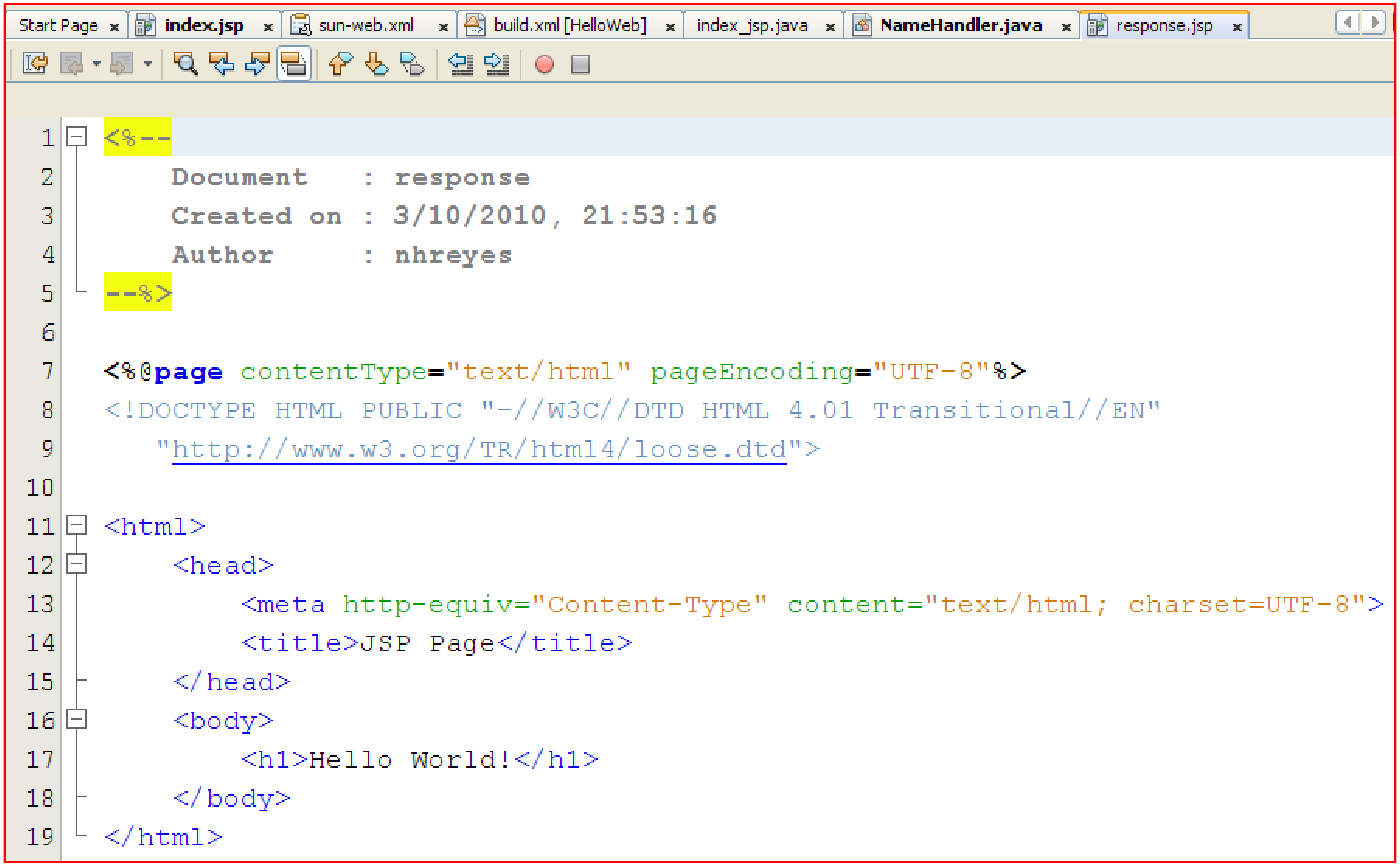
Name and Location

File Name: response
Project: HelloWeb
Location: Web Pages
Folder:
Created File: \Documents and Settings\nhreyes\My Documents\NetBeansProjects\HelloWeb\web\response.jsp

Options:
[✓ JSP File (Standard Syntax)] Create as a JSP Segment
JSP Document (XML Syntax)

Description:
A JSP file using JSP standard syntax.

JSP Source File Generated: response.jsp



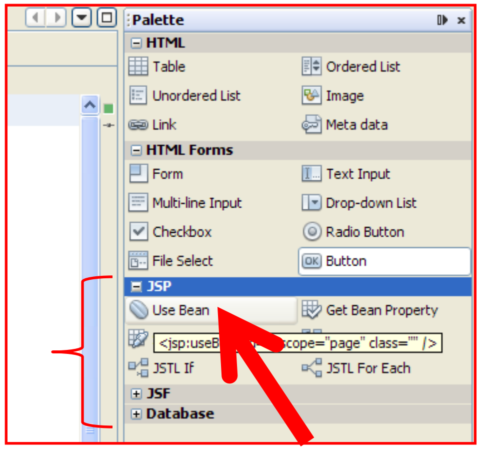
```
<%--
    Document   : response
    Created on : 3/10/2010, 21:53:16
    Author     : nhreyes
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello World!</h1>
    </body>
</html>
```

Adding a Use Bean item

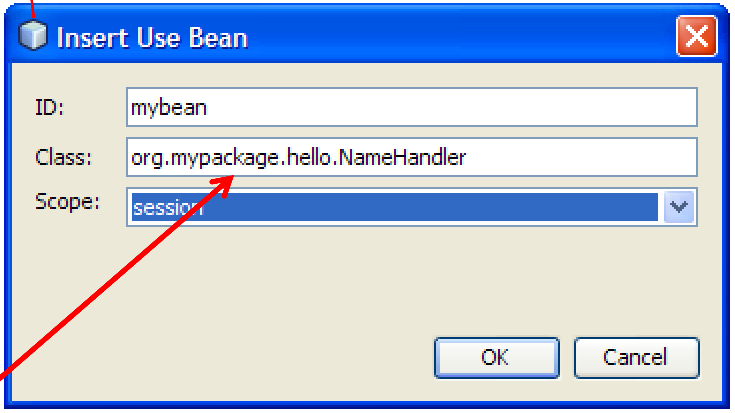
In the **Palette** to the right of the Source Editor, expand **JSP** and drag a **Use Bean** item to a point just below the **<body> tag** in the Source Editor.

The **Insert Use Bean dialog** opens.

Specify the values shown in the following figure.



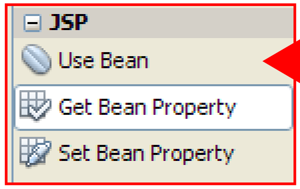
```
<jsp:useB... scope="page" class="" />
```

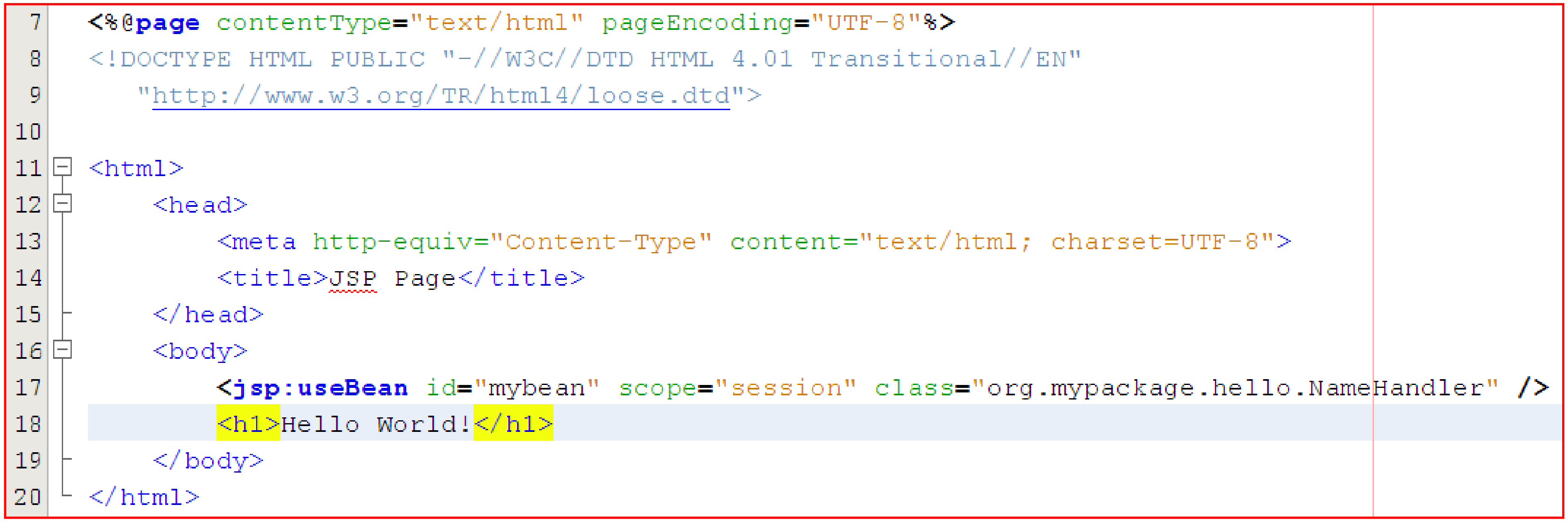


ID: mybean
Class: org.mypackage.hello.NameHandler
Scope: session

The class NameHandler belongs to the package we have set earlier

JSP Source File Generated: response.jsp





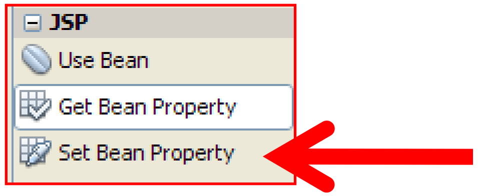
```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <jsp:useBean id="mybean" scope="session" class="org.mypackage.hello.NameHandler" />
        <h1>Hello World!</h1>
    </body>
</html>
```

Notice that the **<jsp:useBean> tag** is added beneath the **<body> tag**.

Adding a Set Bean property item

Drag **a Set Bean Property item** from the Palette to a point just before the <h1> tag and click OK.

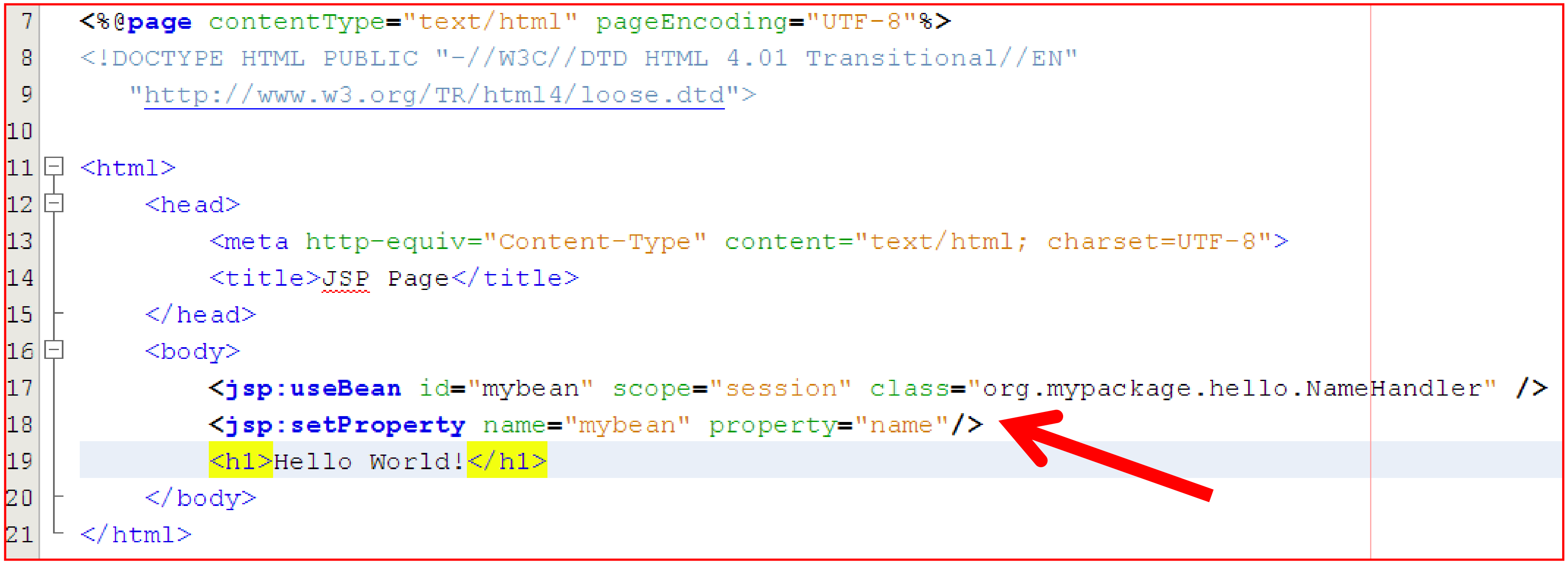
In the <jsp:setProperty> tag that appears, delete the empty value attribute and edit as follows. Delete the **value = ""** attribute if the IDE created it! Otherwise, it overwrites the value for name that you pass in **index.jsp**.



Adding a Set Bean property item

Drag **a Set Bean Property item** from the Palette to a point just before the <h1> tag and click OK.

In the **<jsp:setProperty>** tag that appears, delete the empty value attribute and edit as follows. Delete the **value = ""** attribute if the IDE created it! Otherwise, it overwrites the value for name that you pass in **index.jsp**.

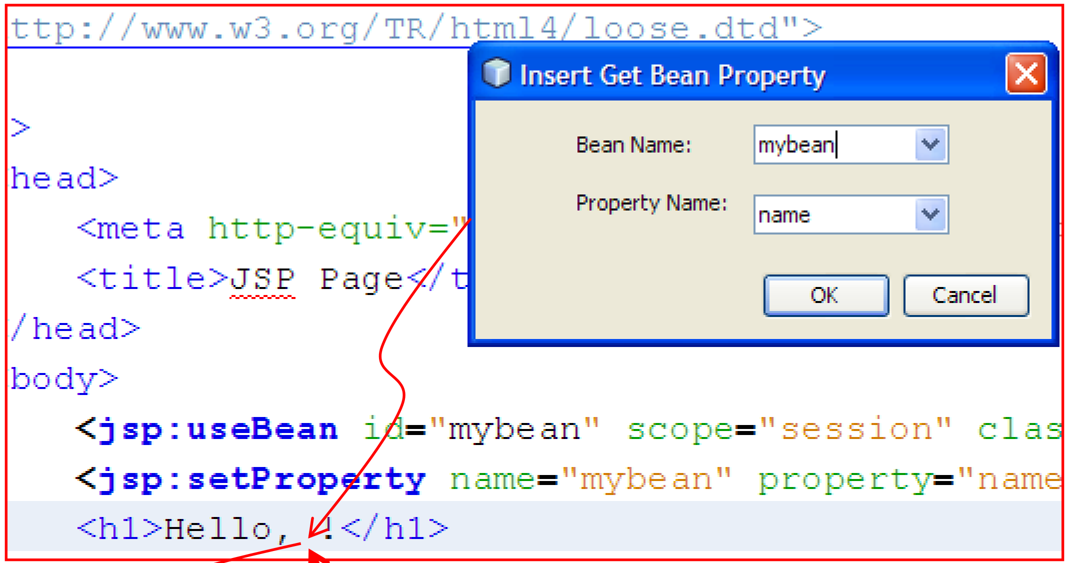


```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <jsp:useBean id="mybean" scope="session" class="org.mypackage.hello.NameHandler" />
        <jsp:setProperty name="mybean" property="name"/>
        <h1>Hello World!</h1>
    </body>
</html>
```

Adding a Get Bean property item

Drag a **Get Bean Property** item from the **Palette** and drop it after the comma between the **<h1>** tags.
Specify the following values in the Insert Get Bean Property dialog:

- **Bean Name: mybean**
- **Property Name: name**

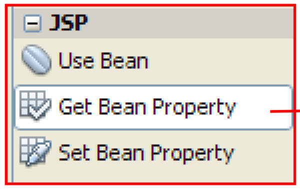


```
ttp://www.w3.org/TR/html4/loose.dtd">

>
head>
    <meta http-equiv="
    <title>JSP Page</t
/head>
body>
    <jsp:useBean id="mybean" scope="session" clas
    <jsp:setProperty name="mybean" property="name
    <h1>Hello, !</h1>
```

Insert Get Bean Property

Bean Name: mybean

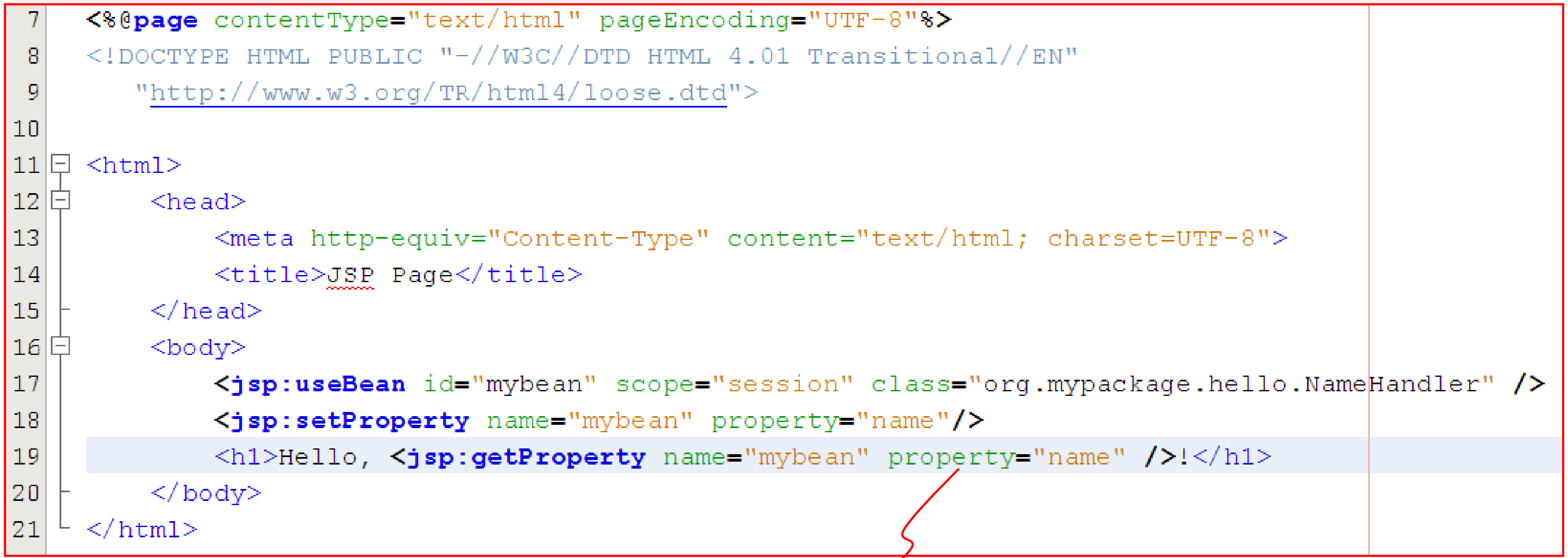
Property Name: name



Insert a Get Bean Property item here!

JSP Source Code Generated

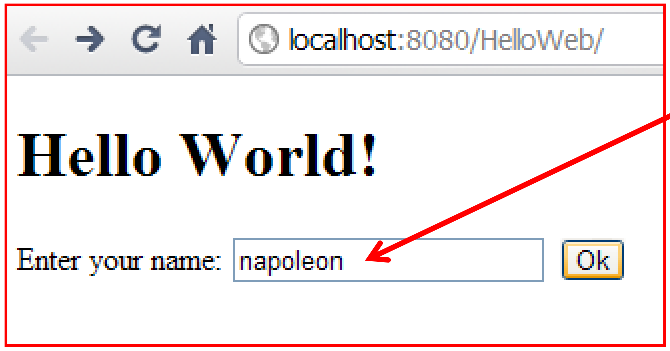
the user input coming from **index.jsp** becomes a **name/value pair** that is passed to the **request object**.
When you set a property using the **<jsp:setProperty> tag**, you can specify the value according to the **name of a property** contained in the **request object**.



```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <jsp:useBean id="mybean" scope="session" class="org.mypackage.hello.NameHandler" />
        <jsp:setProperty name="mybean" property="name"/>
        <h1>Hello, <jsp:getProperty name="mybean" property="name" />!</h1>
    </body>
</html>
```

Therefore, by setting **property** to **name**, you can retrieve the value specified by **user input**.

Sample Run

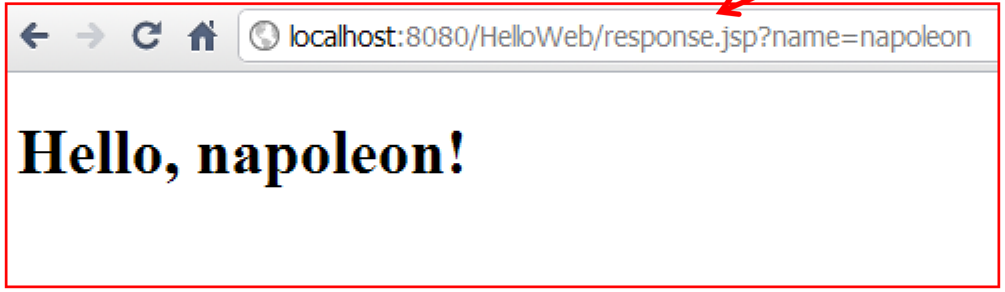

localhost:8080/HelloWeb/

Hello World!

Enter your name: napoleon

User input

Response from the JSP file

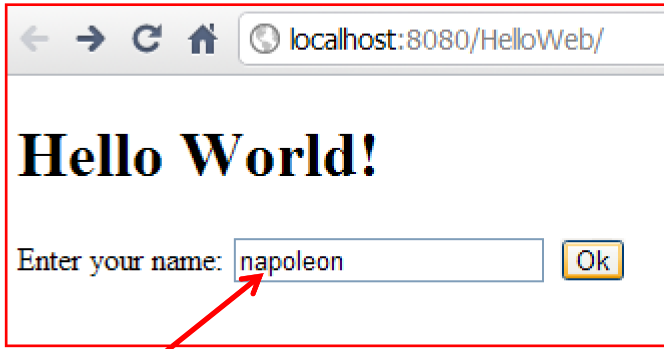

localhost:8080/HelloWeb/response.jsp?name=napoleon

Hello, napoleon!

Sample Run

Index.jsp

Main interface, Html with form
Invokes **response.jsp** through **form action**.



localhost:8080/HelloWeb/

Hello World!

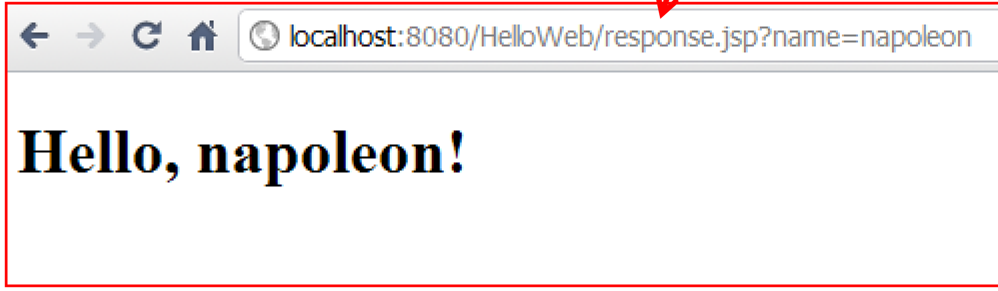
Enter your name: napoleon Ok

User input

Response from the JSP file

NameHandler.java

Class NameHandler
containing user data, get and set methods



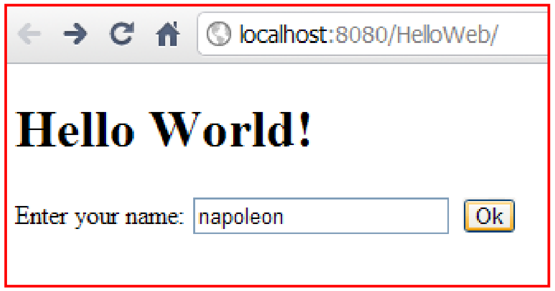
localhost:8080/HelloWeb/response.jsp?name=napoleon

Hello, napoleon!

response.jsp

Generates the server's response
Defines a **JavaBean** to connect the **class NameHandler** to the **user's input** via a **form text field** (name).

Project


localhost:8080/HelloWeb/

Hello World!

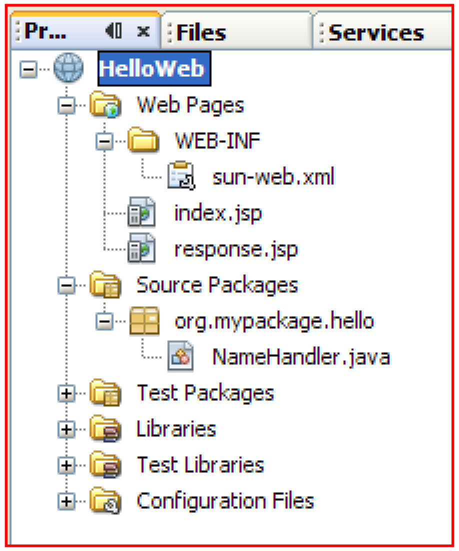
Enter your name: napoleon [Ok]

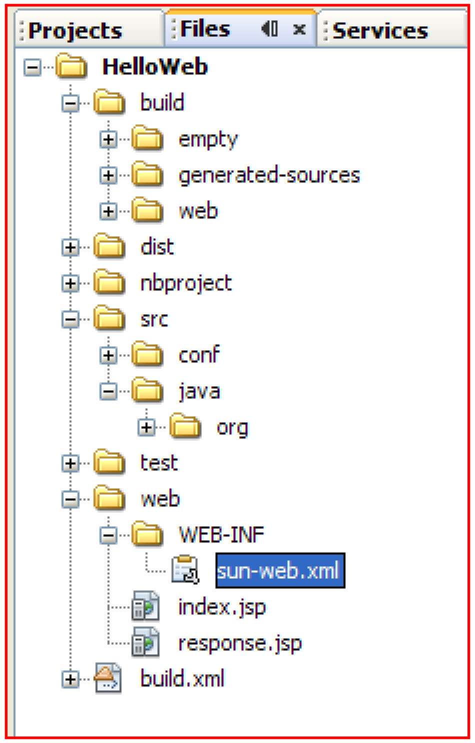
Index.jsp

Main interface, Html with form
Invokes **response.jsp** through **form action**.

NameHandler.java

Class NameHandler
containing user data,
get and set methods


```
Pr...  | Files | Services
HelloWeb
  Web Pages
    WEB-INF
      sun-web.xml
    index.jsp
    response.jsp
  Source Packages
    org.mypackage.hello
      NameHandler.java
  Test Packages
  Libraries
  Test Libraries
  Configuration Files
```


```
Projects | Files | Services
HelloWeb
  build
    empty
    generated-sources
    web
  dist
  nbproject
  src
    conf
    java
      org
  test
  web
    WEB-INF
      sun-web.xml
    index.jsp
    response.jsp
  build.xml
```

http://java.sun.com/blueprints/code/projectconventions.html

response.jsp

Generates the server's response
Defines a **JavaBean** to connect the **class NameHandler** to the **user's input** via a **form text field** (name).

Packaging Web Applications

The Java EE specification defines how the web application can be archived into a **web application archive** (**WAR**)

• **WAR files** are
– Java archives with a **.war extension**
– Packaged using the same specification as zip files
– Understood by all Java EE compliant application servers

• WAR files can be directly deployed in servlet containers such as Tomcat

NetBeans WAR files

- To make a WAR for your NetBeans project, right click on the project node and select **Build Project**.

- The WAR file will be placed in the **“dist” sub-directory** of your project folder

Project

Java EE 6

http://download.oracle.com/javaee/6/tutorial/doc/

Recommended Directory Structure for Projects

http://java.sun.com/blueprints/code/projectconventions.html

NetBeans

http://netbeans.org/kb/docs/web/quickstart-webapps.html

http://www.oracle.com/technetwork/java/javaee/documentation/index.html

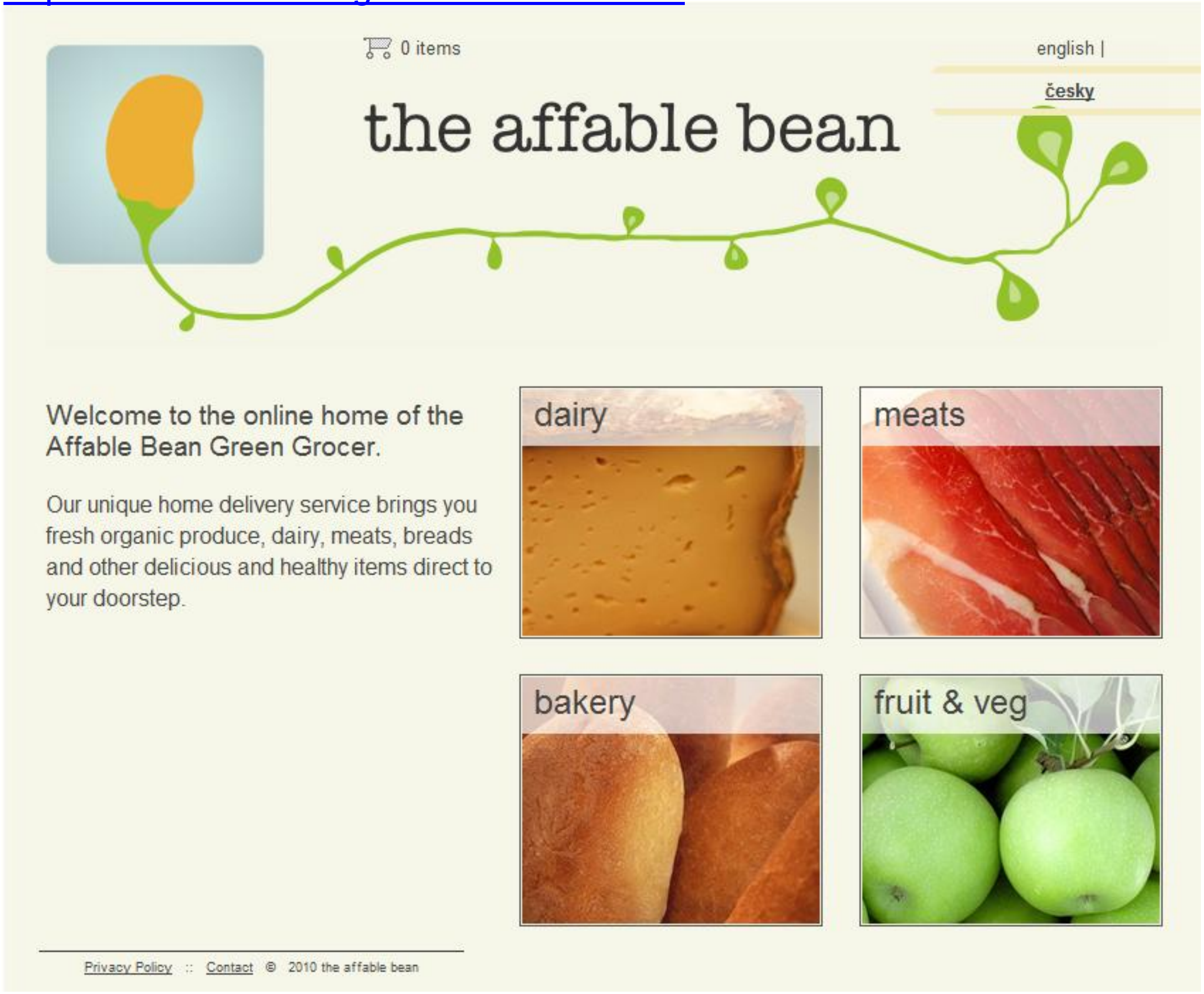
Simple Database Example

http://netbeans.org/kb/docs/web/mysql-webapp.html

E-Commerce Example

http://netbeans.org/kb/docs/javaee/ecommerce/design.html

http://netbeans.org/kb/docs/javaee/ecommerce/data-model.html#createERDiagram http://dot.netbeans.org:8080/AffableBean/



0 items

english |

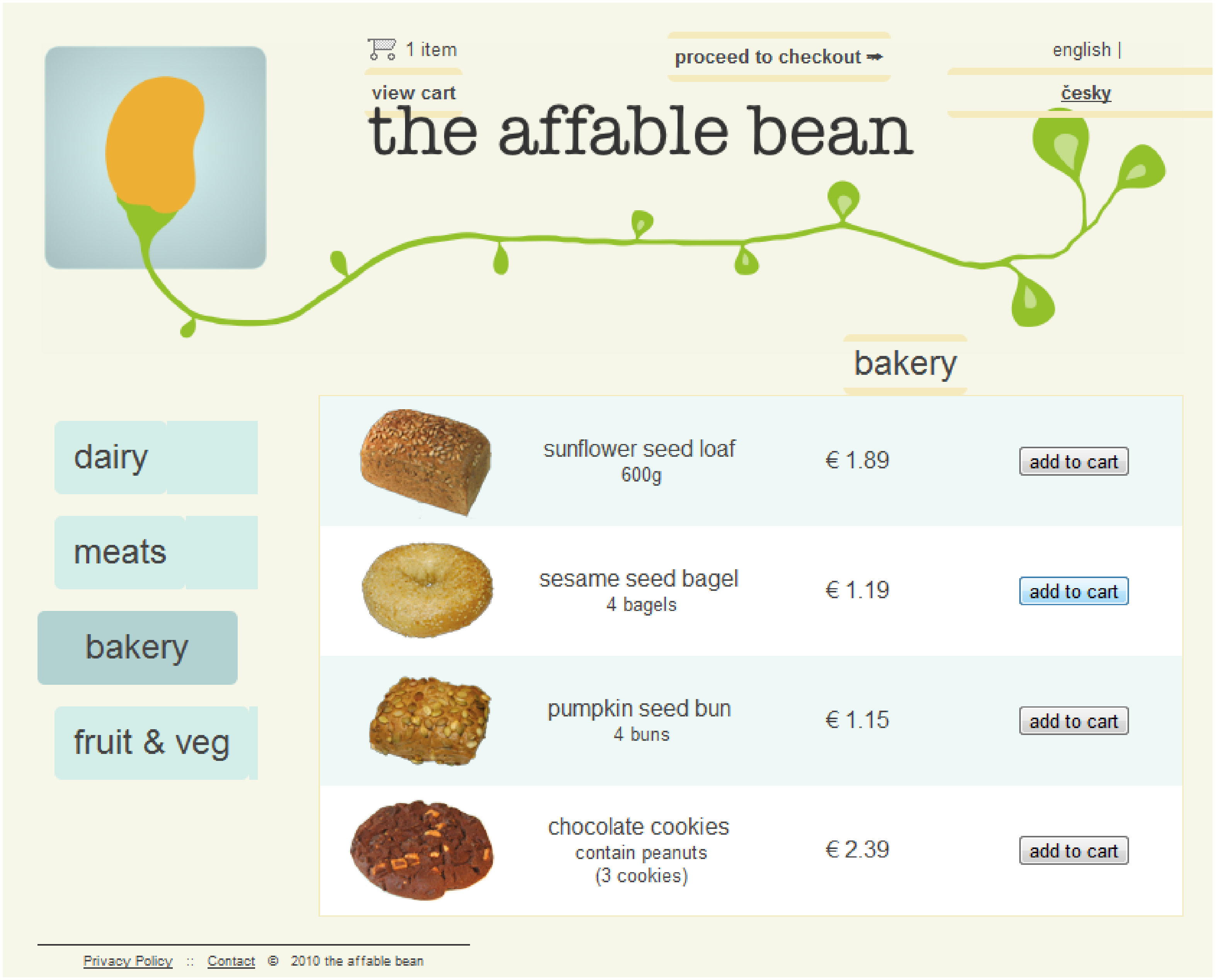
česky

the affable bean

Welcome to the online home of the Affable Bean Green Grocer.

Our unique home delivery service brings you fresh organic produce, dairy, meats, breads and other delicious and healthy items direct to your doorstep.

Privacy Policy :: Contact © 2010 the affable bean



1 item

view cart

proceed to checkout ➠

english |

česky

the affable bean

bakery

dairy

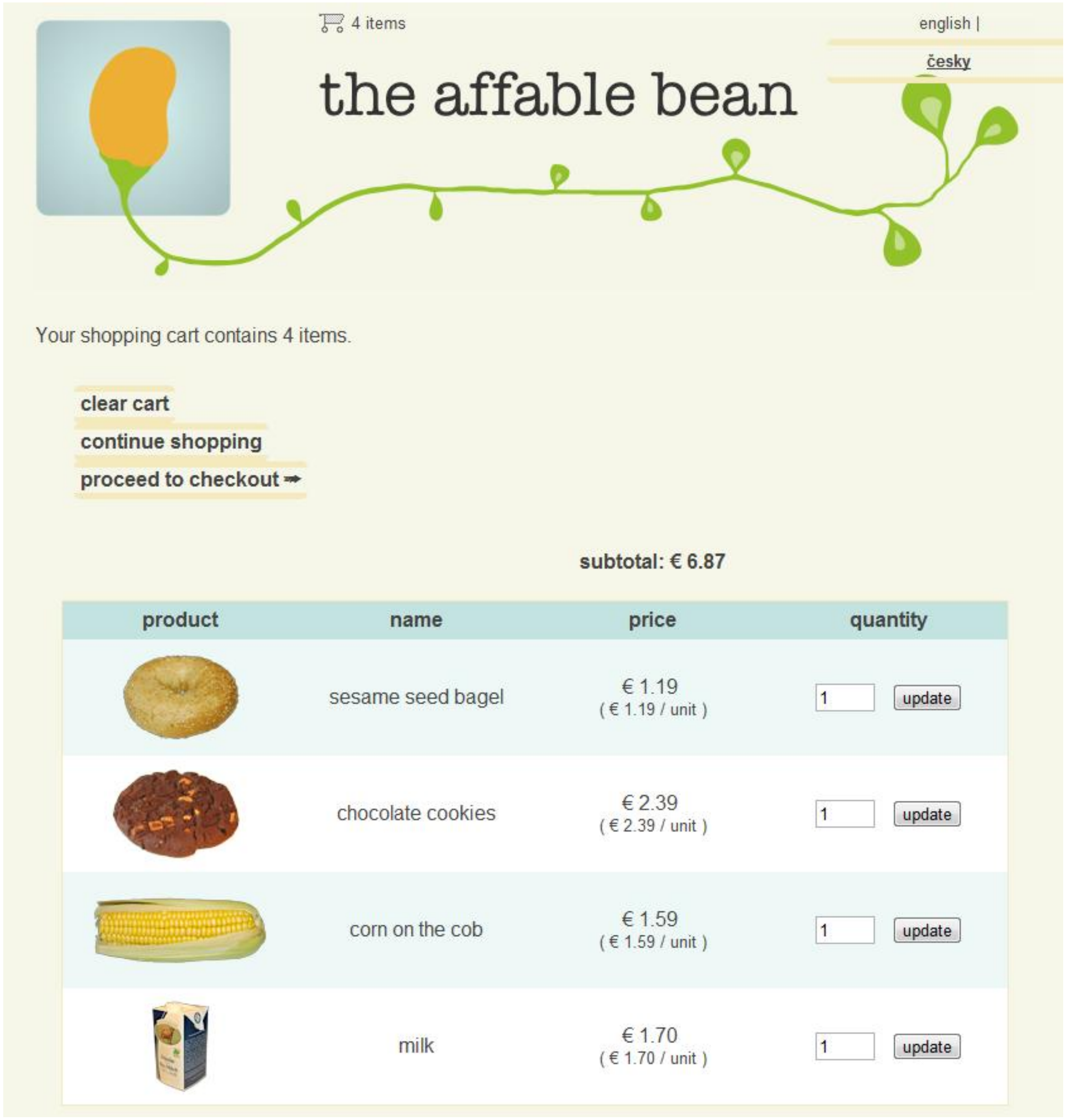
meats

bakery

fruit & veg

	sunflower seed loaf 600g	€ 1.89	add to cart
	sesame seed bagel 4 bagels	€ 1.19	add to cart
	pumpkin seed bun 4 buns	€ 1.15	add to cart
	chocolate cookies contain peanuts (3 cookies)	€ 2.39	add to cart

Privacy Policy :: Contact © 2010 the affable bean

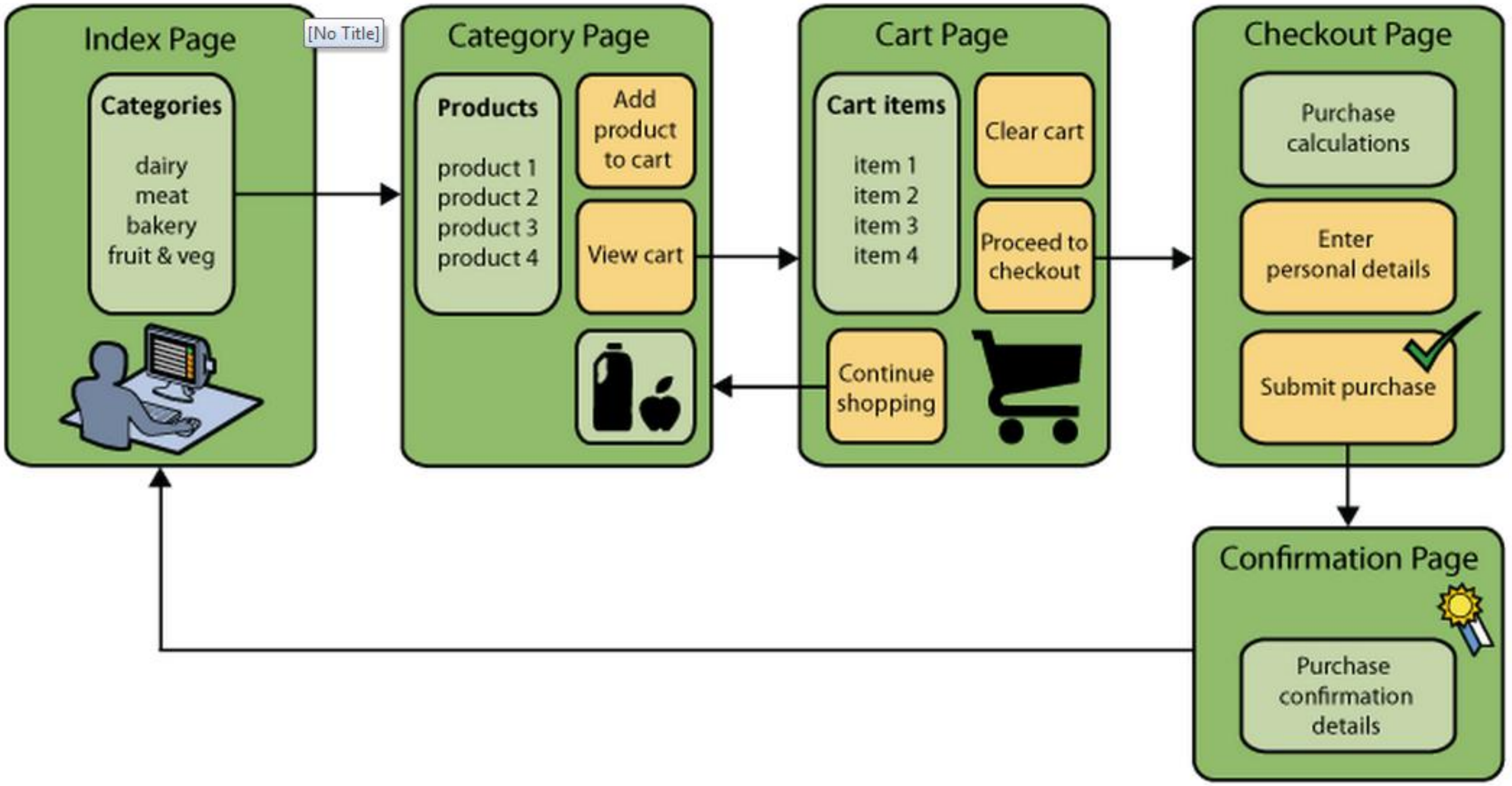


Your shopping cart contains 4 items.

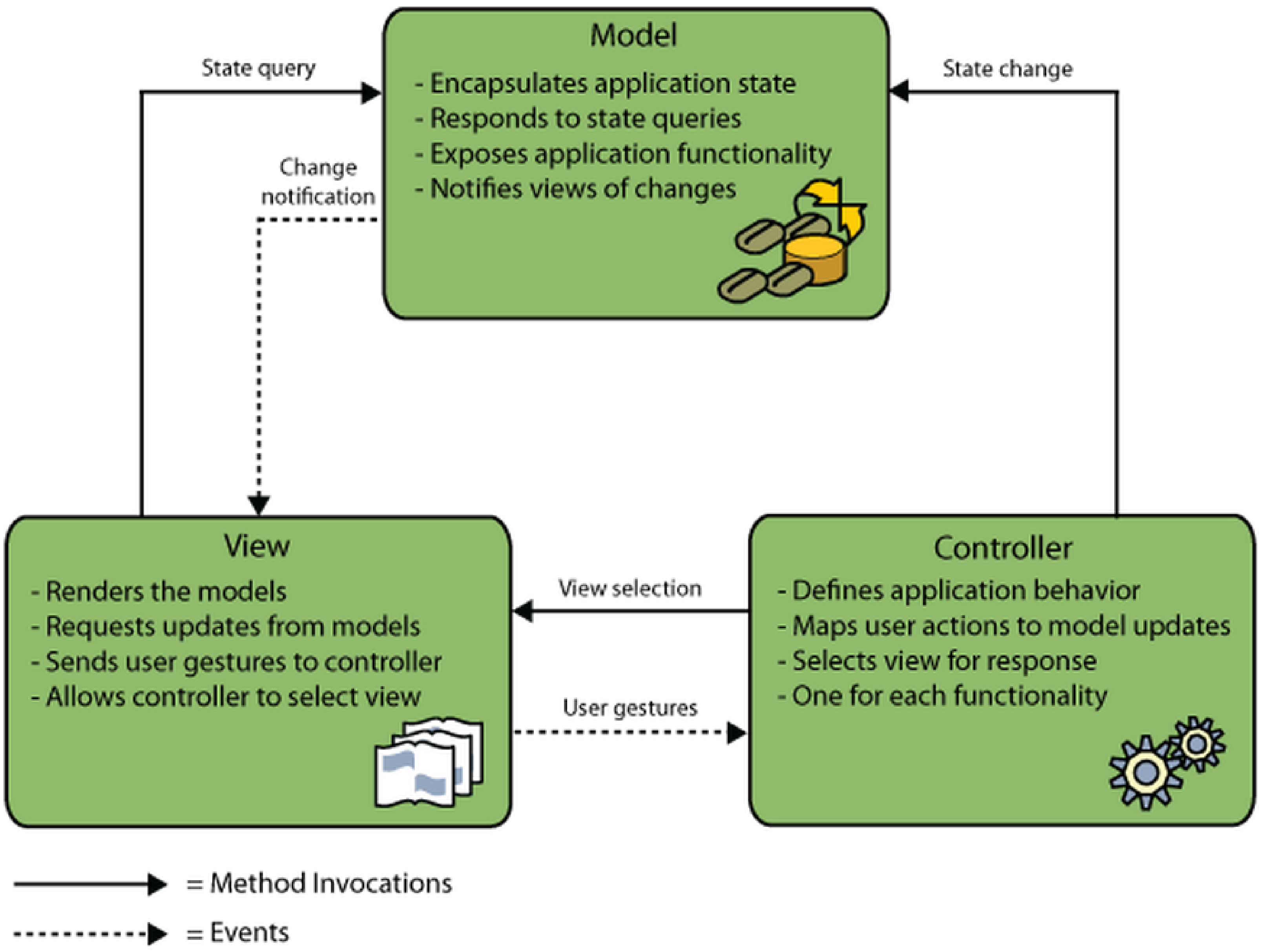
clear cart
continue shopping
proceed to checkout ➠

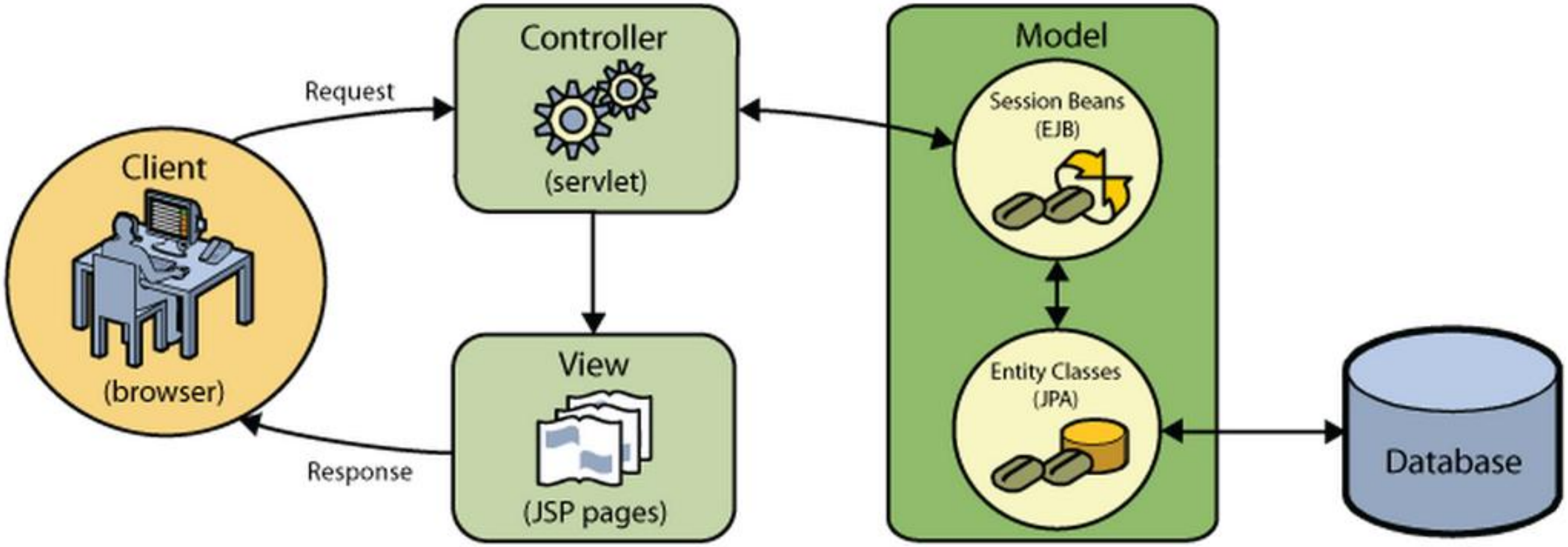
subtotal: € 6.87

product	name	price	quantity
	sesame seed bagel	€ 1.19 (€ 1.19 / unit)	1 update
	chocolate cookies	€ 2.39 (€ 2.39 / unit)	1 update
	corn on the cob	€ 1.59 (€ 1.59 / unit)	1 update
	milk	€ 1.70 (€ 1.70 / unit)	1 update


Index Page
[No Title]
Categories
dairy
meat
bakery
fruit & veg
Category Page
Products
product 1
product 2
product 3
product 4
Add product to cart
View cart
Cart Page
Cart items
item 1
item 2
item 3
item 4
Clear cart
Proceed to checkout
Continue shopping
Checkout Page
Purchase calculations
Enter personal details
Submit purchase
Confirmation Page
Purchase confirmation details

Model-View-Controller Paradigm


Model
- Encapsulates application state
- Responds to state queries
- Exposes application functionality
- Notifies views of changes
State query
State change
Change notification
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
(servlet)
Model
Session Beans
(EJB)
Request
Client
(browser)
View
(JSP pages)
Response
Entity Classes
(JPA)
Database