

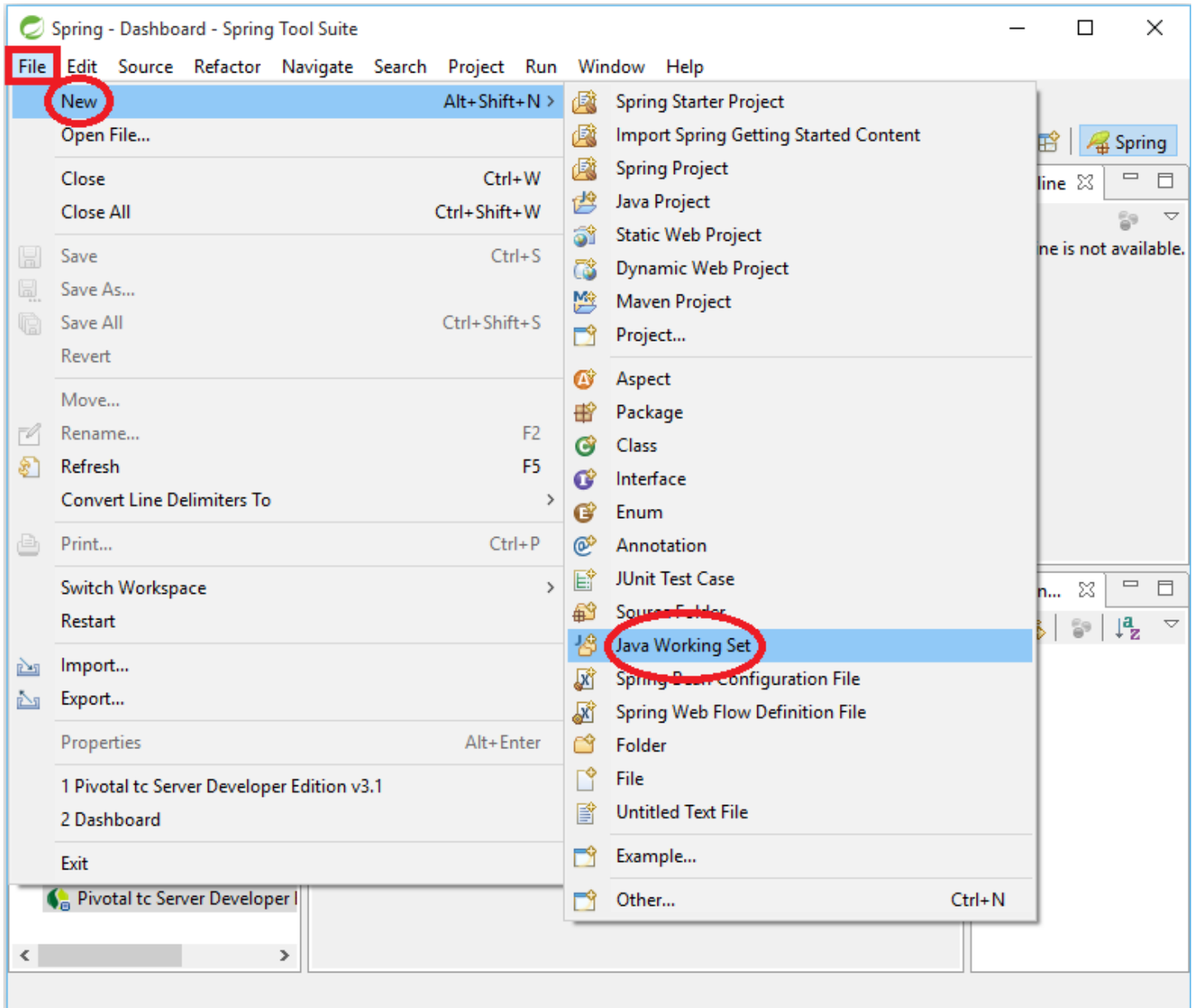
## ***“Spring MVC – Básico parte II”***

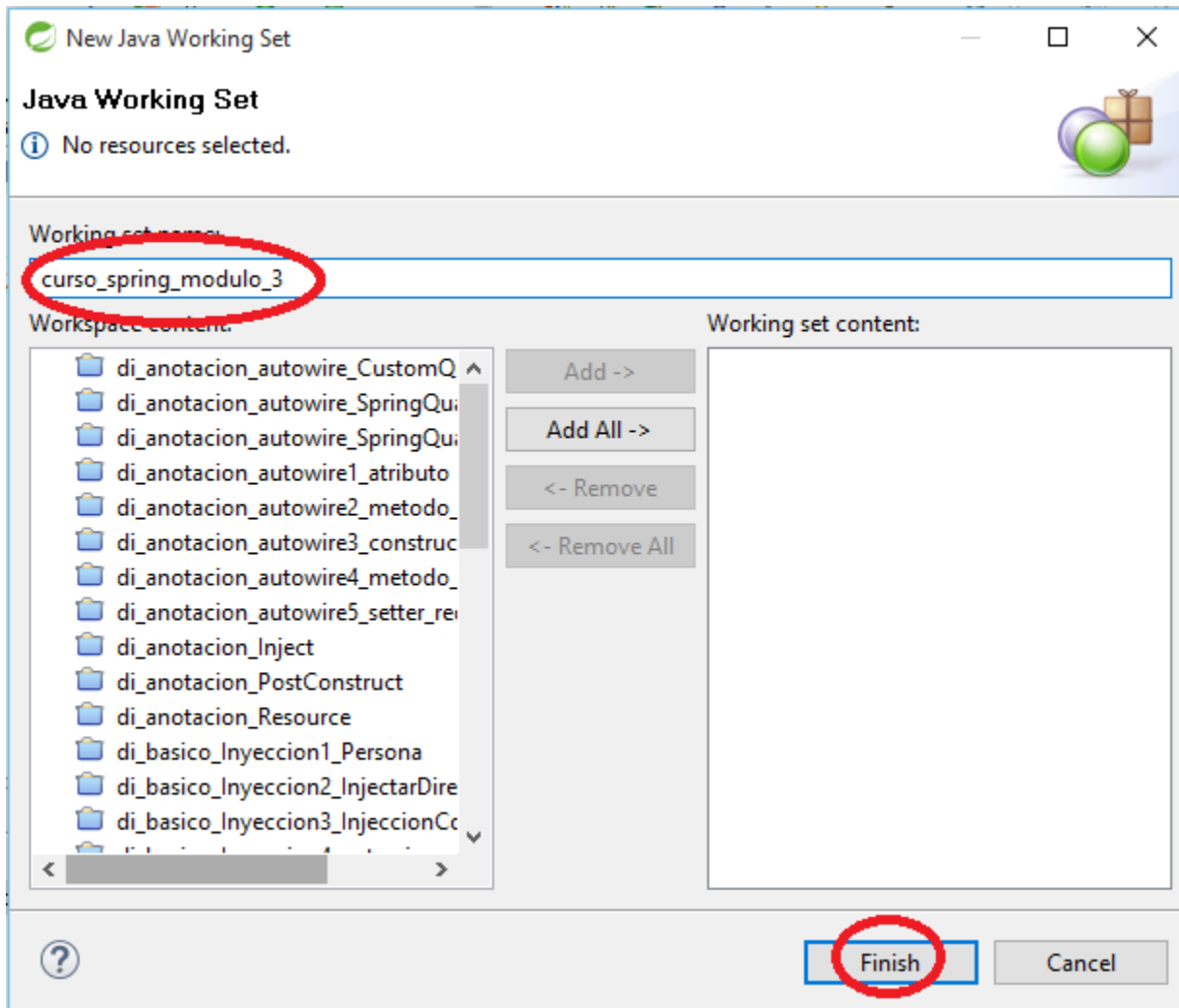
### **Módulo 3 / 2**

© *Todos los logos y marcas utilizados en este documento, están registrados y pertenecen a sus respectivos dueños.*

## Ejercicio 0: Importar todos los proyectos de ejemplo

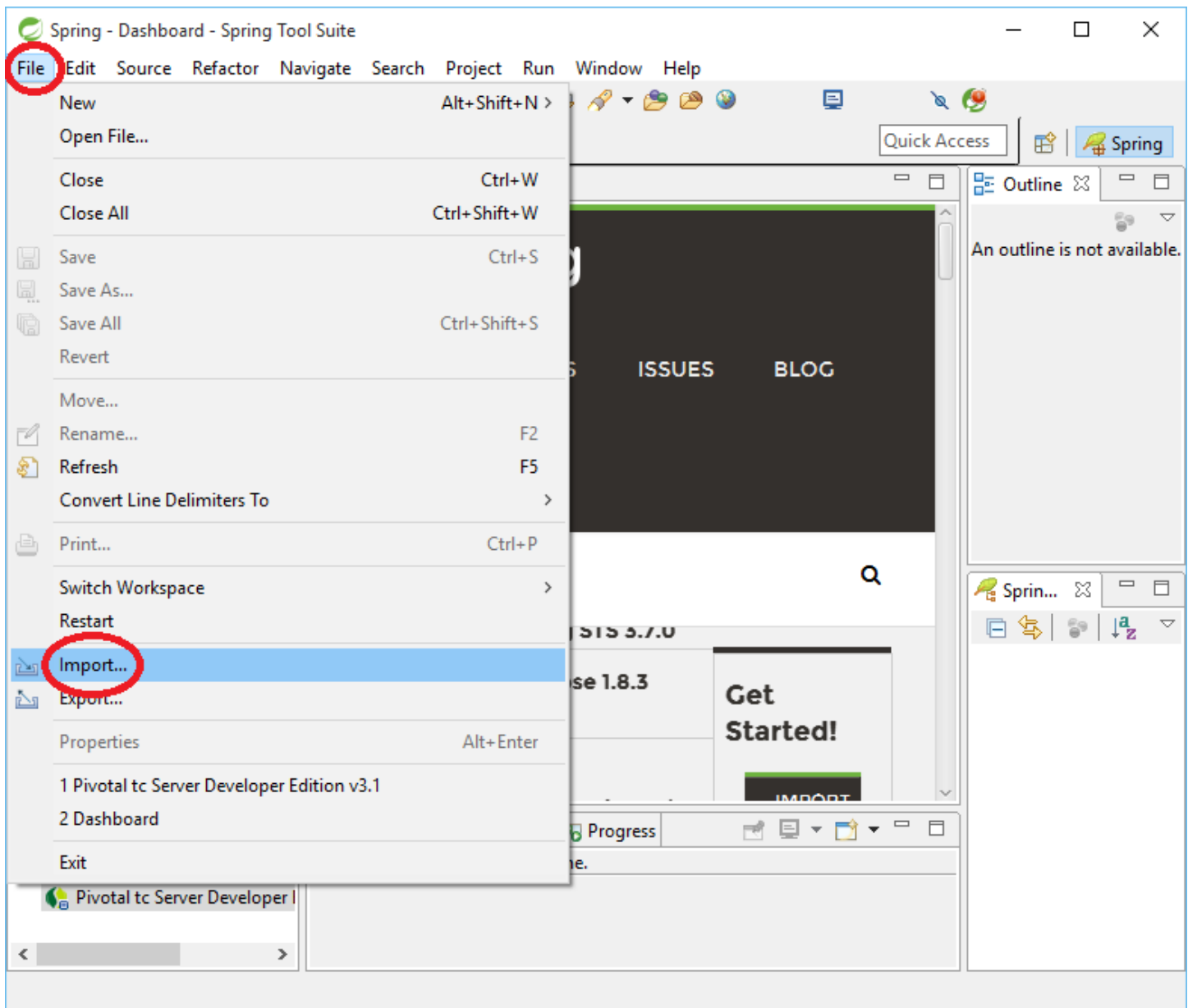
1. Crear un nuevo "Java Working Set" llamado "curso\_spring\_modulo\_3". Esto es para organizar los proyectos bajo un esquema llamado **Working Set**, similar a como organizamos archivos en directorios.
  - Seleccionar **File->New->Java Working Set**.



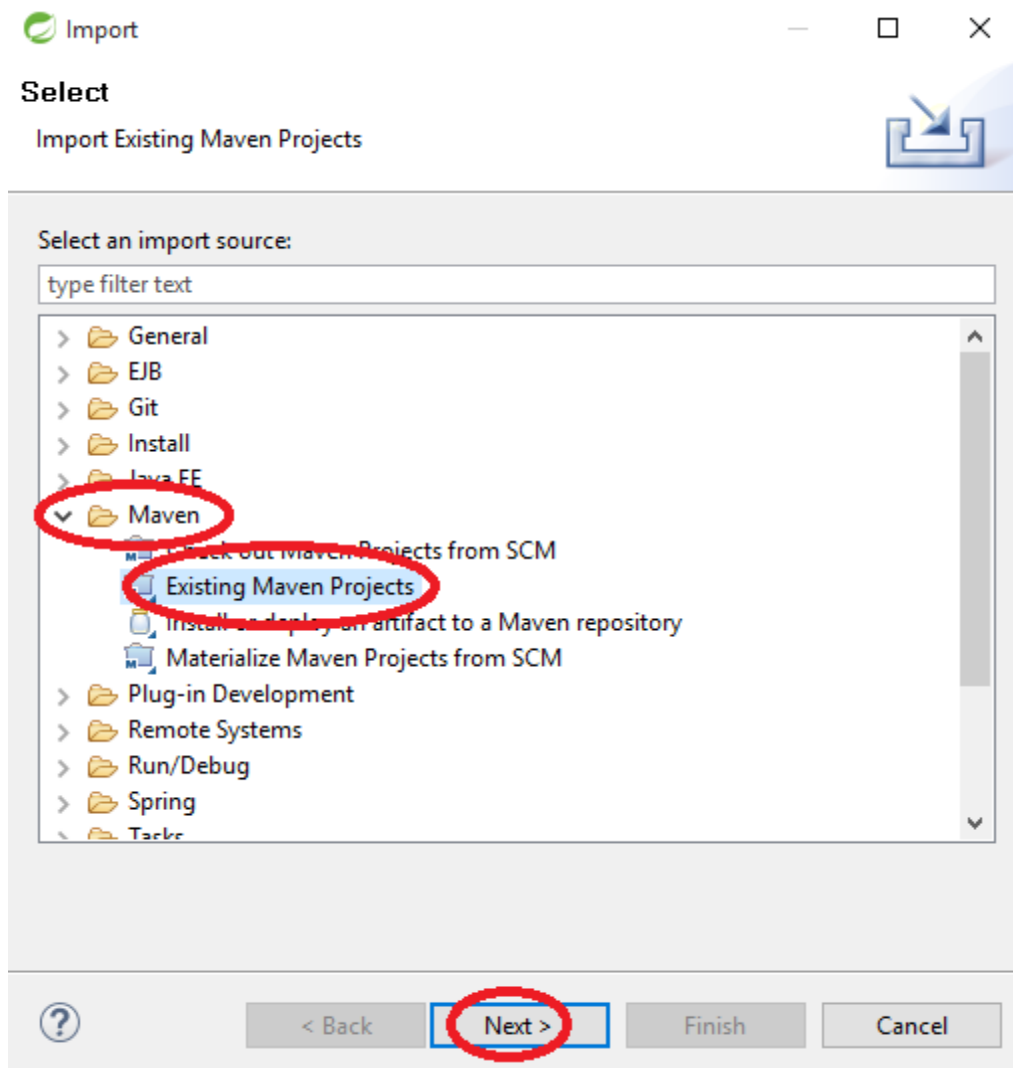


## 2. Importar los proyectos de ejemplos en maven.

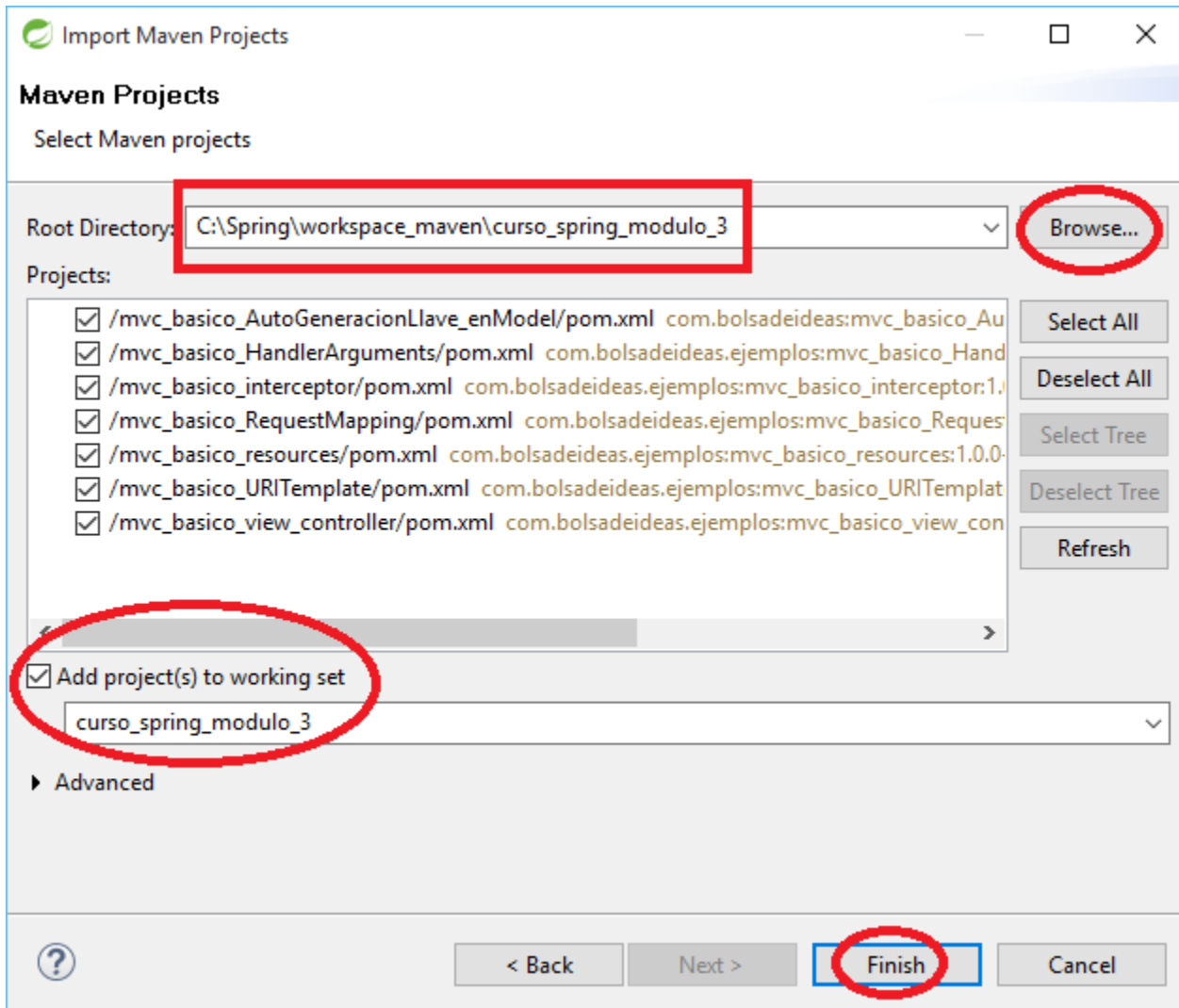
- Seleccionar **File->Import**.



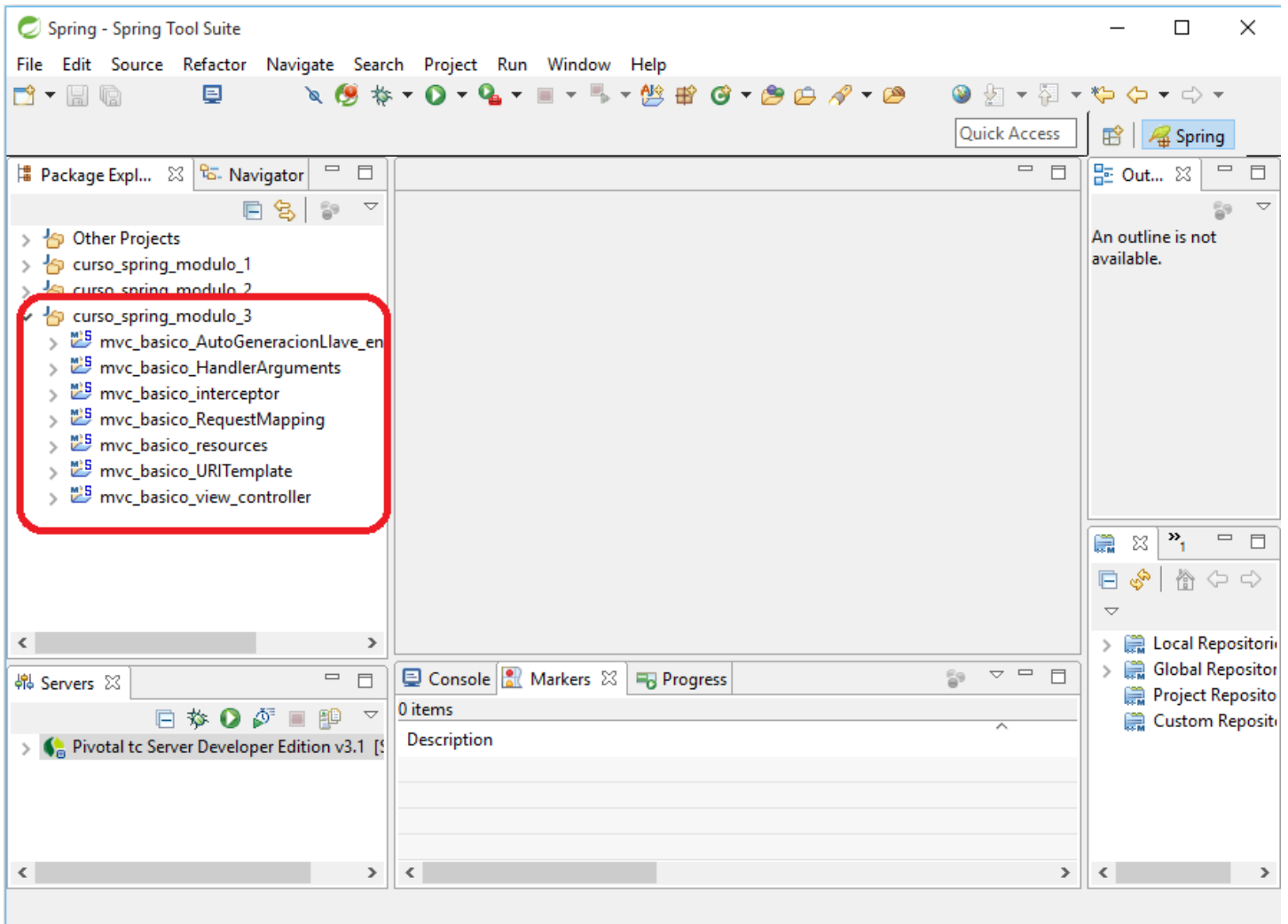
- Buscamos **Maven -> Existing Maven Projects**
- Clic **Next >**



- Clic **Browse**
- Seleccionamos el directorio donde vienen los proyecto de ejemplo del **laboratorio módulo 3**
- Agregamos los proyectos al **Working Set curso\_spring\_modulo\_3**
- **Finish**



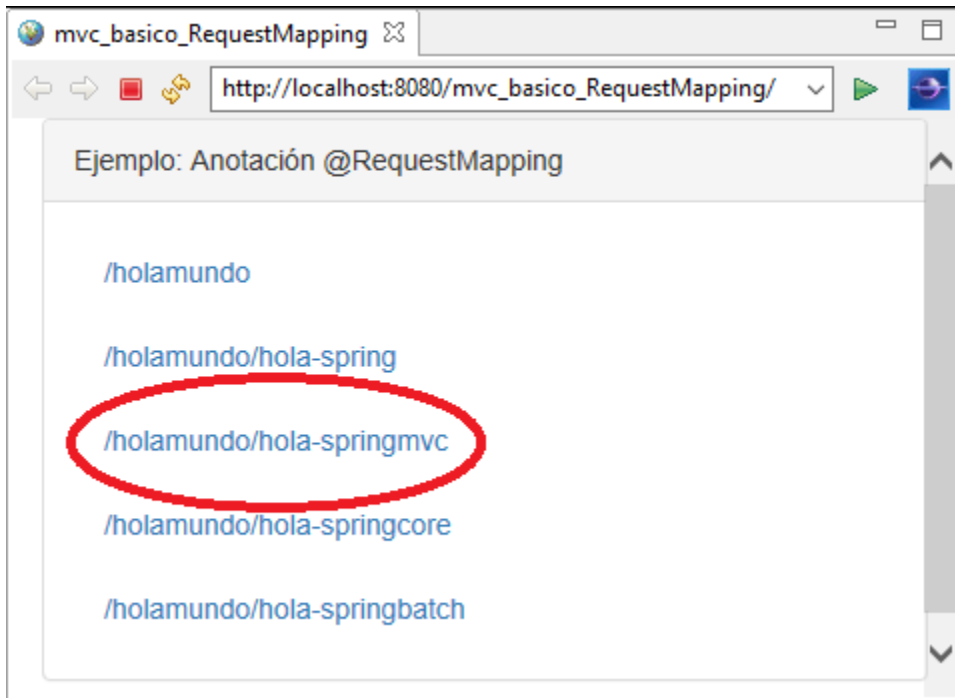
- Finalmente tenemos organizados nuestros proyectos en **Working Set**.



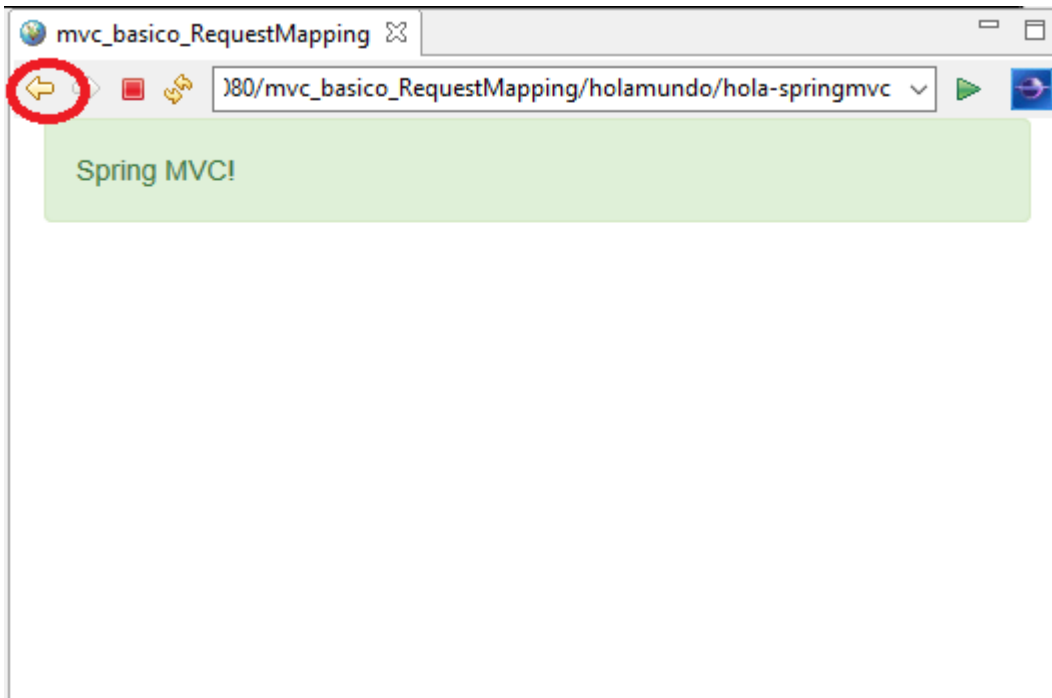
## Ejercicio 1: Abrir, generar, y ejecutar el proyecto de ejemplo "mvc\_basico\_RequestMapping", Request mapping, Models, lógica de views (como retornar valores)

En este ejercicio, vamos a ver diferentes formas de request mappings, cómo pasar atributos a la vista usando mediante el objeto model y por supuesto crear las vistas.

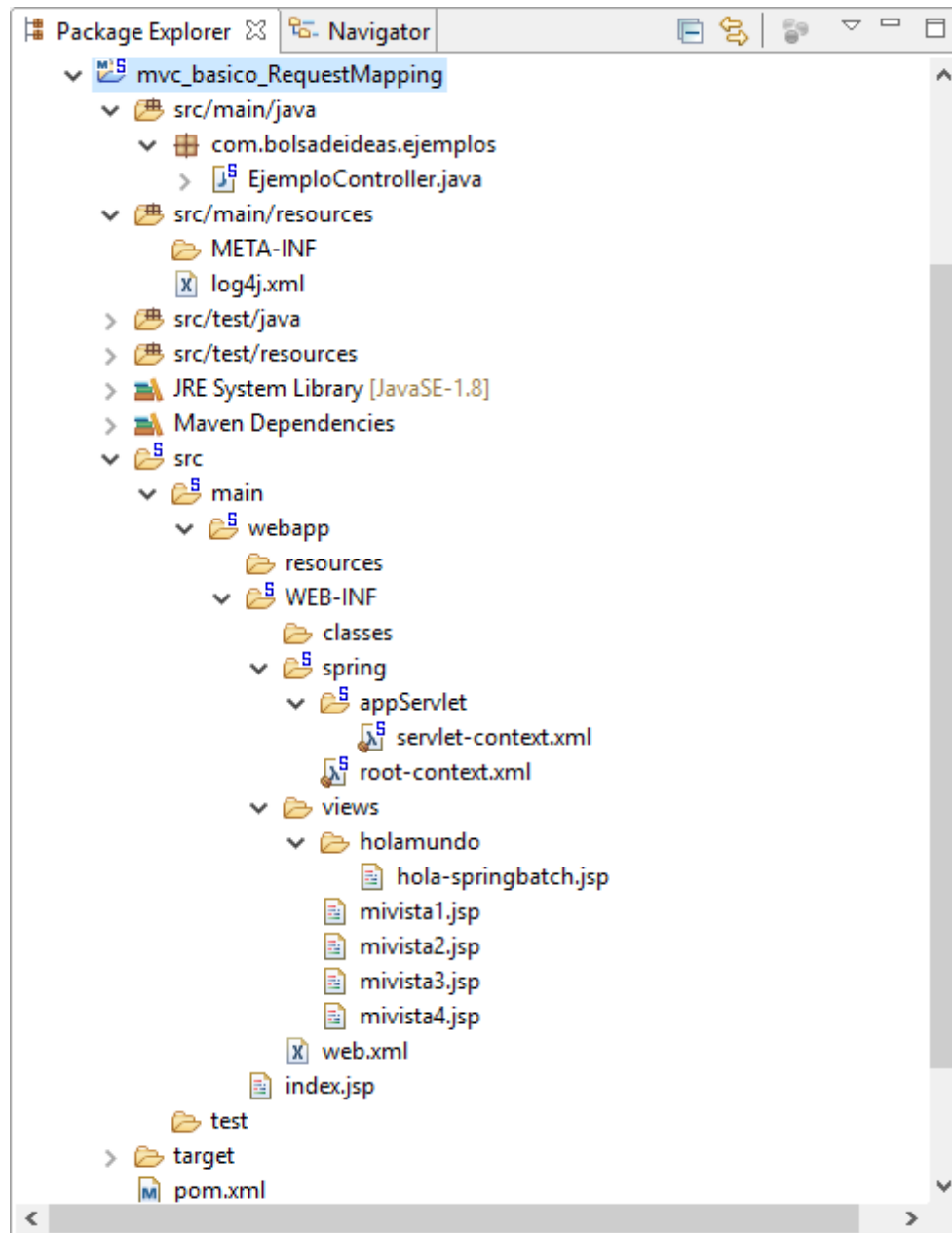
1. Clic derecho sobre el proyecto y "Run As"->"Maven Clean".
2. Clic derecho sobre el proyecto y "Run As"->"Maven Install"
3. Hacemos un **Maven->Update Project...**
4. Clic derecho sobre el proyecto y "Run As"->"Run on Server".
5. Ejecutamos la aplicación: clic derecho sobre **mvc\_basico\_RequestMapping -> Run As on Server**
6. Observe el resultado.







## 7. Estudiamos El proyecto



- Abrir y estudiar la clase controladora

**/src/main/java/com.bolsadeideas.ejemplos/EjemploController.java**

```
package com.bolsadeideas.ejemplos;

import java.util.Map;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

@Controller
@RequestMapping("/holamundo")
public class EjemploController {

    // Mapeada a la URL /holamundo
    @RequestMapping(method = RequestMethod.GET)
    // Retorna un tipo String
    // Como argumento recibe el objeto model del tipo ModelMap
    public String holaMundo(ModelMap modelMap) {

        modelMap.addAttribute("mensaje1", "Hola Mundo!");
        return "mivista1";
    }

    // Mapeada a la URL /holamundo/hola-spring
    @RequestMapping(value = "/hola-spring", method = RequestMethod.GET)
    // Retorna un tipo String
    // Como argumento recibe el objeto model del tipo Map
    public String holaMundo(Map map) {

        map.put("mensaje2", "Spring!");
        return "mivista2";
    }

    // Mapeada a la URL /holamundo/hola-springmvc
    @RequestMapping(value = "/hola-springmvc", method = RequestMethod.GET)
    // Retorna un tipo ModelAndView
    // Se crea el objeto ModelAndView dentro del método
    public ModelAndView holaMundo() {

        ModelAndView modelAndView = new ModelAndView("mivista3");
        modelAndView.addObject("mensaje3", "Spring MVC!");

        return modelAndView;
    }
}
```

```
// Mapeada a la URL /holamundo/hola-springcore
@RequestMapping(value = "/hola-springcore", method = RequestMethod.GET)
// Retorna un tipo String
// Como argumento recibe el objeto model del tipo Model
public String holaMundo(Model model) {

    model.addAttribute("mensaje4", "Spring Core!");
    return "mivista4";
}

// Mapeada a la URL holamundo/hola-springbatch
@RequestMapping(value = "/hola-springbatch", method = RequestMethod.GET)
// No Retorna nada, es void,
// "/holamundo/hola-springbatch" define el nombre y ubicación de la vista,
// por lo tanto la vista que ejecutará será
// "/holamundo/hola-springbatch.jsp".
// Como argumento recibe el objeto model del tipo Model
public void holaMundo2(Model model) {
    model.addAttribute("mensaje5", "Spring Batch!");
}
}
```

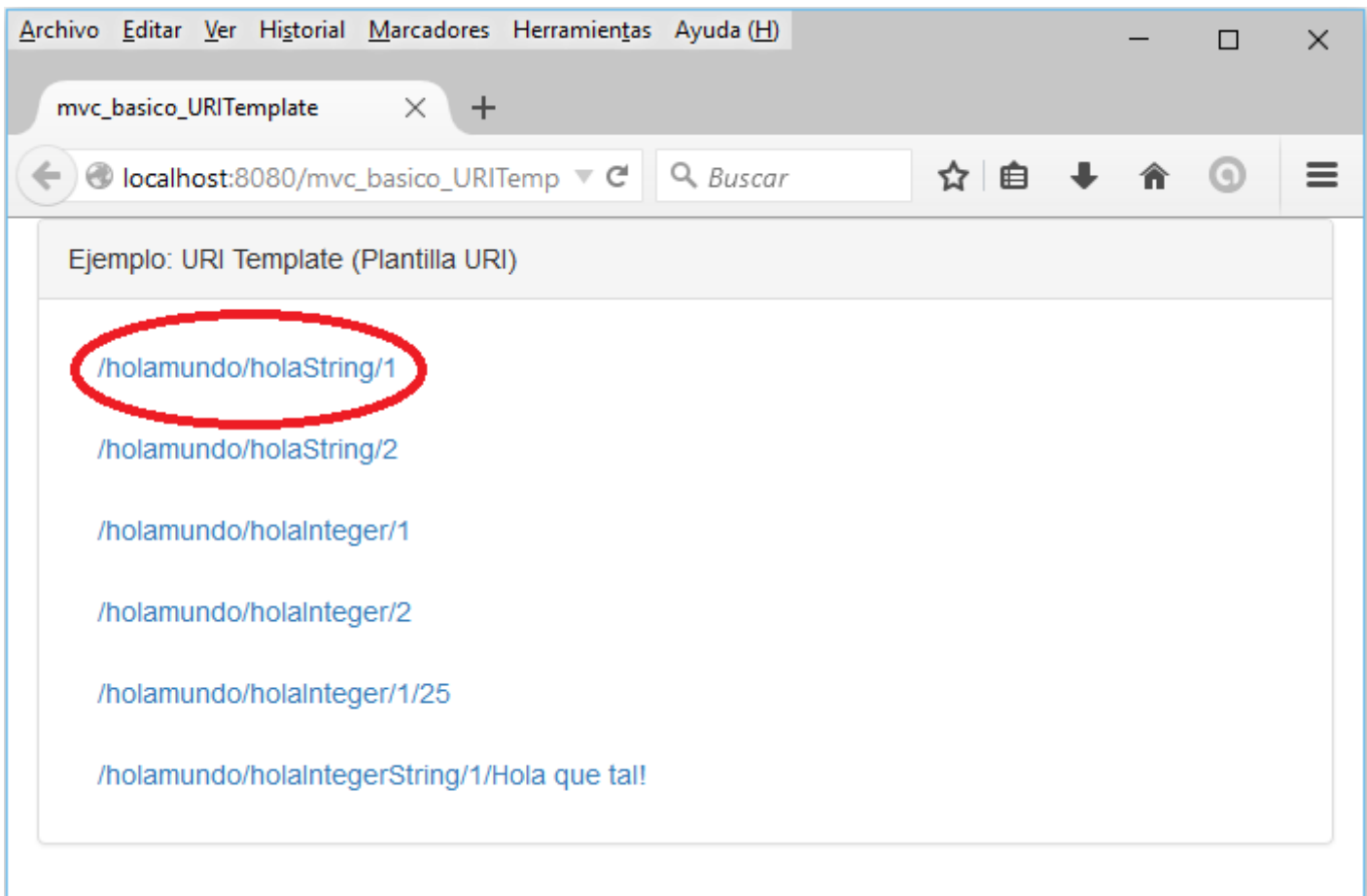
---

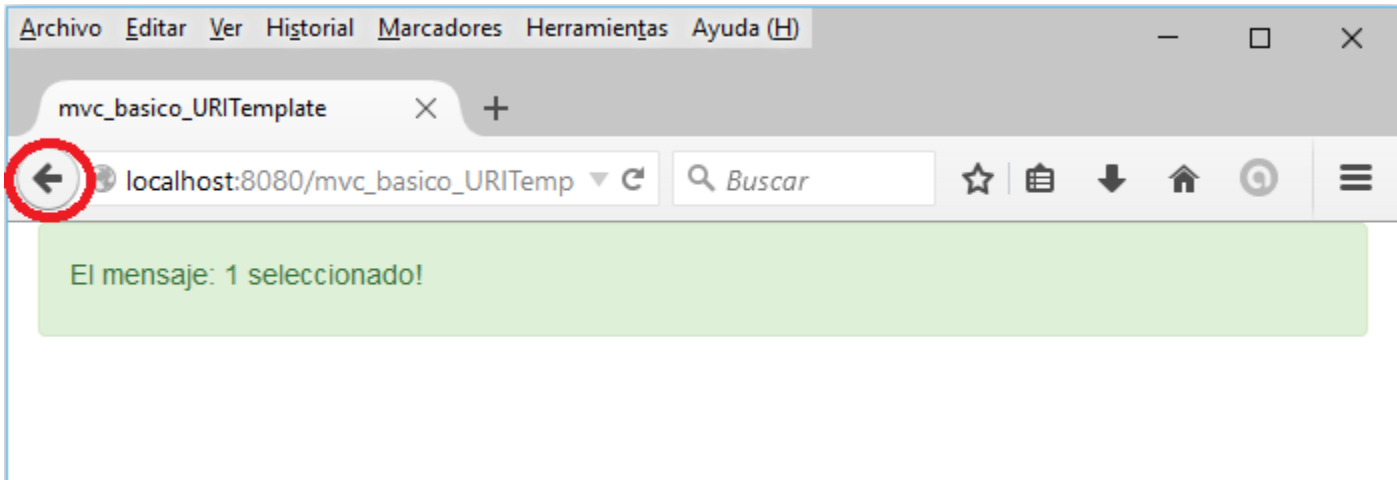
- Abrir y estudiar las vistas `/src/main/webapp/WEB-INF/views/...`
- Abrir y estudiar los otros componentes y archivos.

## Ejercicio 2: Abrir, generar y ejecutar el proyecto de ejemplo URI Templates "mvc\_basico\_URI\_Template"

Un template URI es un String URL, que contiene uno o más nombres de variables.

1. Clic derecho sobre el proyecto y "Run As"->"Maven Clean".
2. Clic derecho sobre el proyecto y "Run As"->"Maven Install".
3. Hacemos un **Maven->Update Project...**
4. Clic derecho sobre el proyecto y "Run As"->"Run on Server".
5. Ejecutamos la aplicación: clic derecho sobre **mvc\_basico\_URI\_Template -> Run As on Server**
6. Observe el resultado.





## 2. Abrir y estudiar la clase controladora

**/src/main/java/com.bolsadeideas.ejemplos/EjemploController.java**

```
package com.bolsadeideas.ejemplos;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping("/holamundo")
public class EjemploController {

    // Handle URI template, parametro holaId es recibido como tipo String
    @RequestMapping(value = "/holaString/{holaId}", method = RequestMethod.GET)
    public String holaURITemplate(@PathVariable String holaId, Model model) {

        model.addAttribute("mensaje" + holaId, "El mensaje: " + holaId
            + " seleccionado!");
        return "miPagina" + holaId;
    }

    // Handle URI template, parametro holaId es recibido como tipo int
    @RequestMapping(value = "/holaInteger/{holaId}", method = RequestMethod.GET)
    public String holaURITemplate2(@PathVariable int holaId, Model model) {

        model.addAttribute("mensaje" + holaId, "El mensaje: " + holaId
            + " seleccionado!");
        return "miPagina" + holaId;
    }
}
```

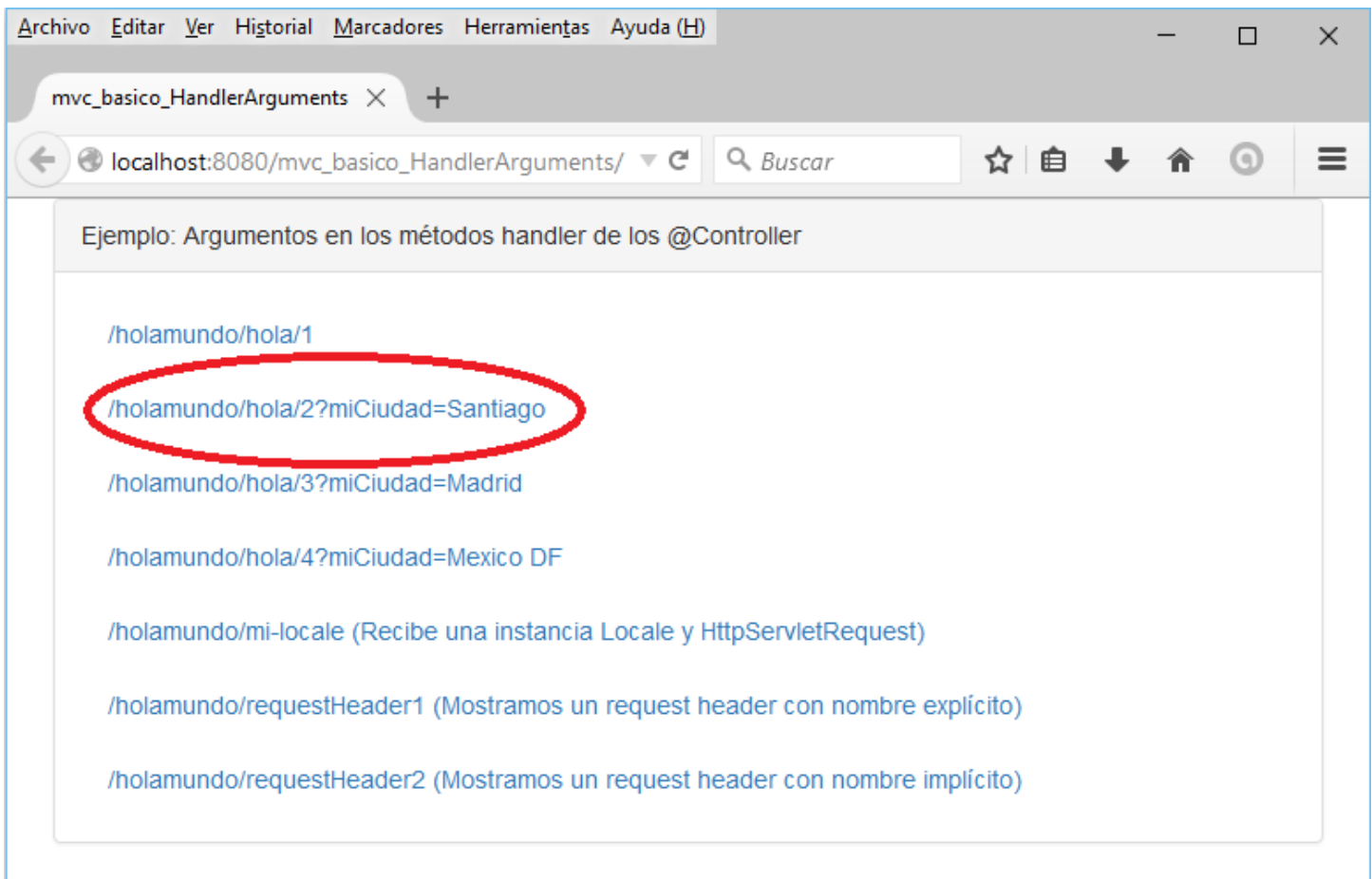
```
// Handle URI template, parametro holaId es recibido como tipo int
@RequestMapping(value = "/holaInteger/{holaId}/{numero}", method = RequestMethod.GET)
public String holaURITemplate3(@PathVariable int holaId,
    @PathVariable int numero, Model model) {
    model.addAttribute("mensaje" + holaId, "El mensaje: " + holaId
        + " seleccionado!");
    model.addAttribute("miNumero", "mi número es " + numero);
    return "miPagina" + holaId;
}

// Handle URI template, parametro holaId es recibido como tipo int
// y un parametro saludo String
@RequestMapping(value = "/holaIntegerString/{holaId}/{saludo}", method =
RequestMethod.GET)
public String holaURITemplate4(@PathVariable int holaId,
    @PathVariable String saludo, Model model) {
    model.addAttribute("mensaje" + holaId, "mensaje" + holaId
        + " seleccionado!" + " El mensaje de saludo es: " + saludo);
    return "miPagina" + holaId;
}
}
```

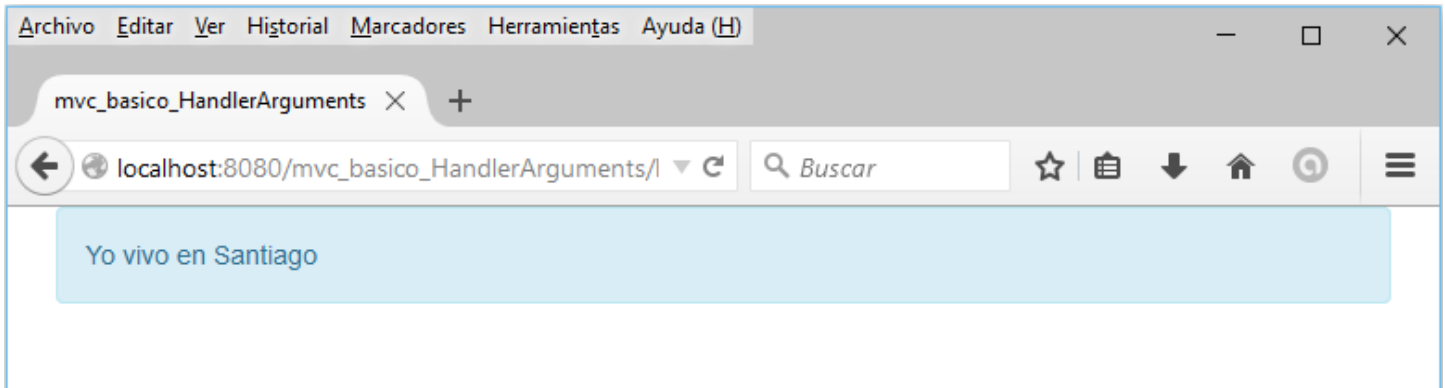
### Ejercicio 3: Abrir, generar y ejecutar el proyecto de ejemplo "mvc\_basico\_HandlerArguments"

Los métodos Handler pueden recibir varios tipos de argumentos, tales como objetos pre-construidos.

1. Clic derecho sobre el proyecto y "Run As"->"Maven Clean".
2. Clic derecho sobre el proyecto y "Run As"->"Maven Install".
3. Hacemos un **Maven->Update Project...**
4. Clic derecho sobre el proyecto y "Run As"->"Run on Server".
5. Ejecutamos la aplicación: clic derecho sobre **mvc\_basico\_HandlerArguments-> Run As on Server**
6. Observe el resultado.

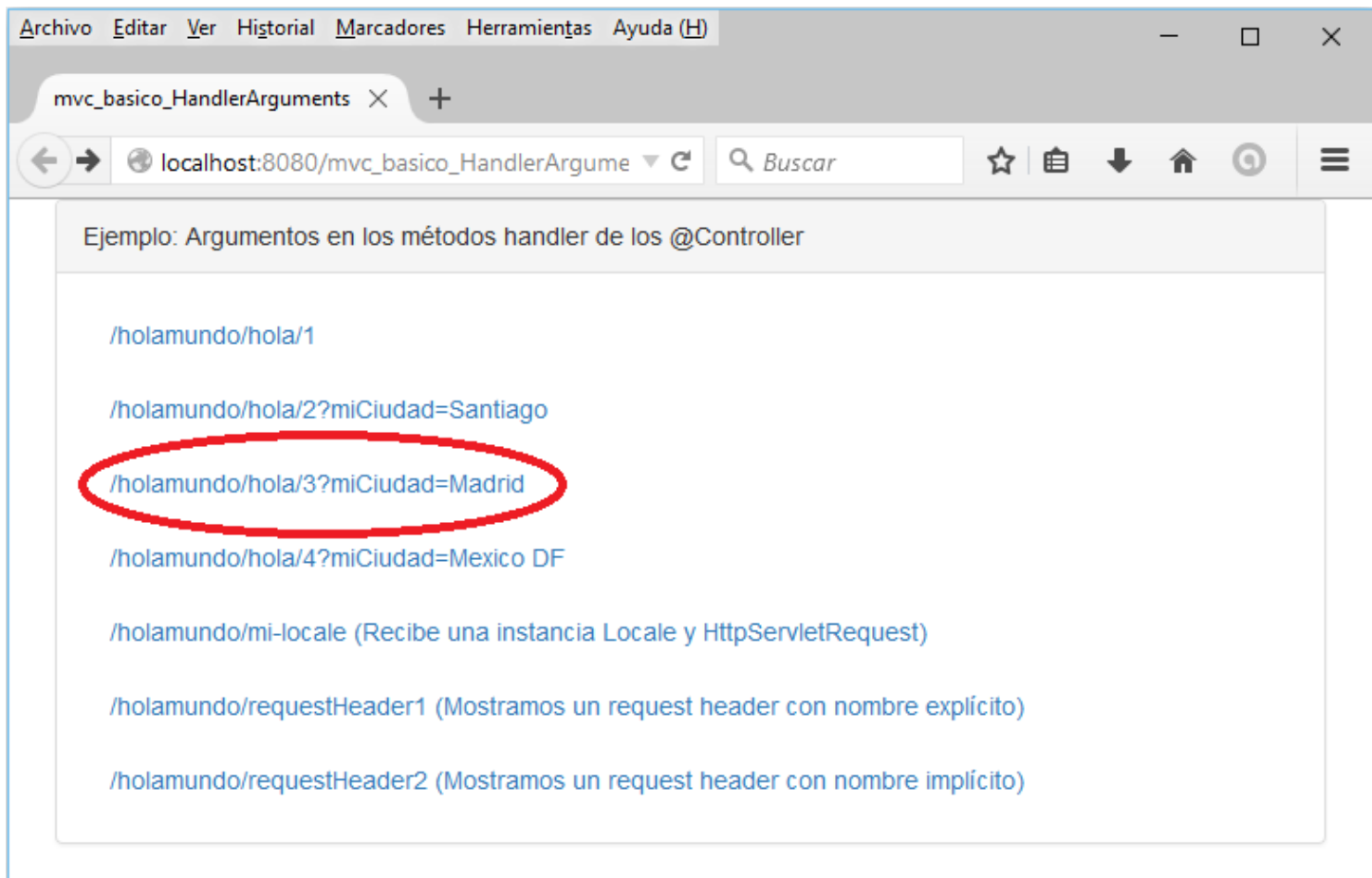


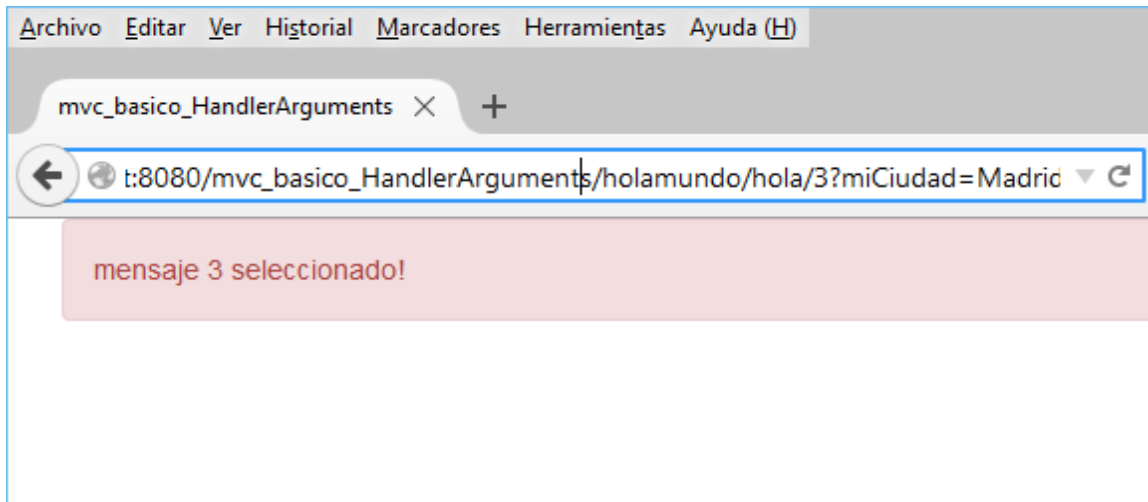




3. Intentemos con

[http://localhost:8080/mvc\\_basico\\_HandlerArguments/holamundo/hola/1?miCiudad=Madrid](http://localhost:8080/mvc_basico_HandlerArguments/holamundo/hola/1?miCiudad=Madrid)





No lo muestra ya que se espera que el valor del parámetro sea "**Santiago**":

**params = "miCiudad=Santiago"**

4. Abrir y estudiar la clase controladora

**/src/main/java/com.bolsadeideas.ejemplos/EjemploController.java**

```
package com.bolsadeideas.ejemplos;
import java.util.Locale;

import javax.servlet.http.HttpServletRequest;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
@RequestMapping("/holamundo")
public class EjemploController {

    @RequestMapping(value = "/hola/{holaId}", method = RequestMethod.GET)
    public String holaURI(@PathVariable int holaId, Model model) {

        model.addAttribute("mensaje" + holaId, "mensaje" + holaId
            + " seleccionado!");
        return "miPagina" + holaId;
    }
}
```

```
// Recivimos parámetros del request
@RequestMapping(value = "/hola/{holaId}", method = RequestMethod.GET,
params = "miCiudad=Santiago")
public String holaURIparams(@PathVariable int holaId,
    @RequestParam("miCiudad") String ciudad, Model model) {

    model.addAttribute("mensaje" + holaId, "Yo vivo en " + ciudad);
    return "miPagina" + holaId;
}

// Recive una instancia Locale y HttpServletRequest
@RequestMapping(value = "/mi-locale", method = RequestMethod.GET)
public String holaObjects(Model model, Locale locale,
    HttpServletRequest httpRequest) {

    model.addAttribute("pais", locale.getCountry());
    model.addAttribute("ContextPath", httpRequest.getContextPath());
    return "miLocale";
}

// Mostramos un request header con nombre explícito
@RequestMapping(value = "requestHeader1")
public @ResponseBody
String conHeader1(@RequestHeader("Accept") String Accept) {
    return "Obtenemos 'Accept' header '" + Accept + "'";
}

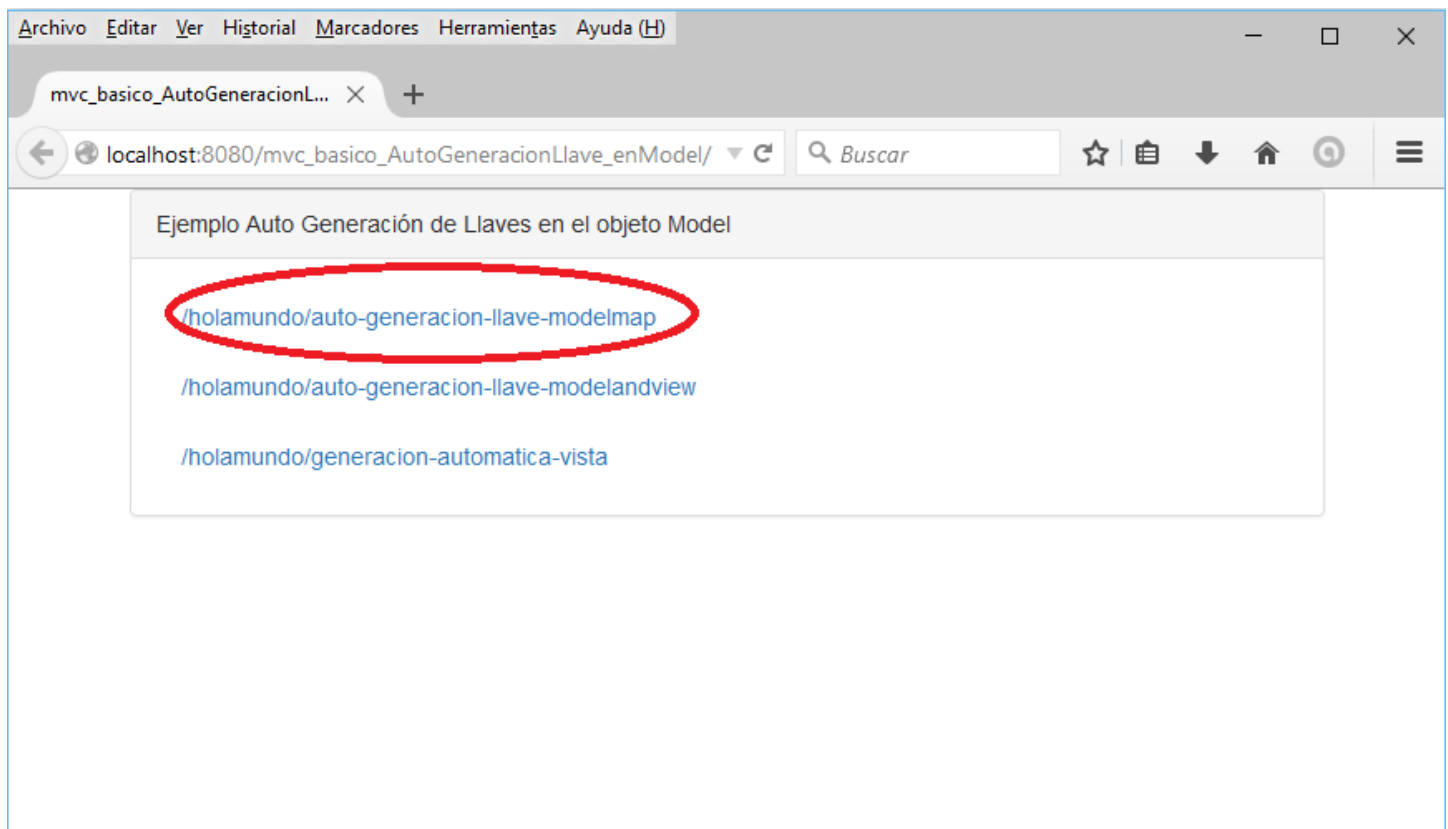
// Mostramos un request header con nombre implícito
@RequestMapping(value = "requestHeader2")
public @ResponseBody
String conHeader2(@RequestHeader String Accept) {
    return "Obtenemos 'Accept' header '" + Accept + "'";
}
}
```

---

## Ejercicio 4: Generación automática de llaves y vistas ejemplo "mvc\_basico\_AutoGeneracionLlave\_enModel"

En el siguiente ejemplo aprenderemos sobre la generación automática de llaves en atributos soportados tanto para ModelMap como en ModelAndView. Además veremos la generación automática del nombre de las vistas en ModelAndView cuando no se especifica una de forma explícita.

1. Clic derecho sobre el proyecto y "Run As"-"Maven Clean".
2. Clic derecho sobre el proyecto y "Run As"-"Maven Install".
3. Hacemos un **Maven->Update Project...**
4. Clic derecho sobre el proyecto y "Run As"-"Run on Server".
5. Observe el resultado.
6. Clic en auto generación de llaves en atributos para ModelMap.



Archivo Editar Ver Historial Marcadores Herramientas Ayuda (H)

Carro de Compras

odel/holamundo/auto-generacion-llave-modelmap

Buscar

## Carro de Compras

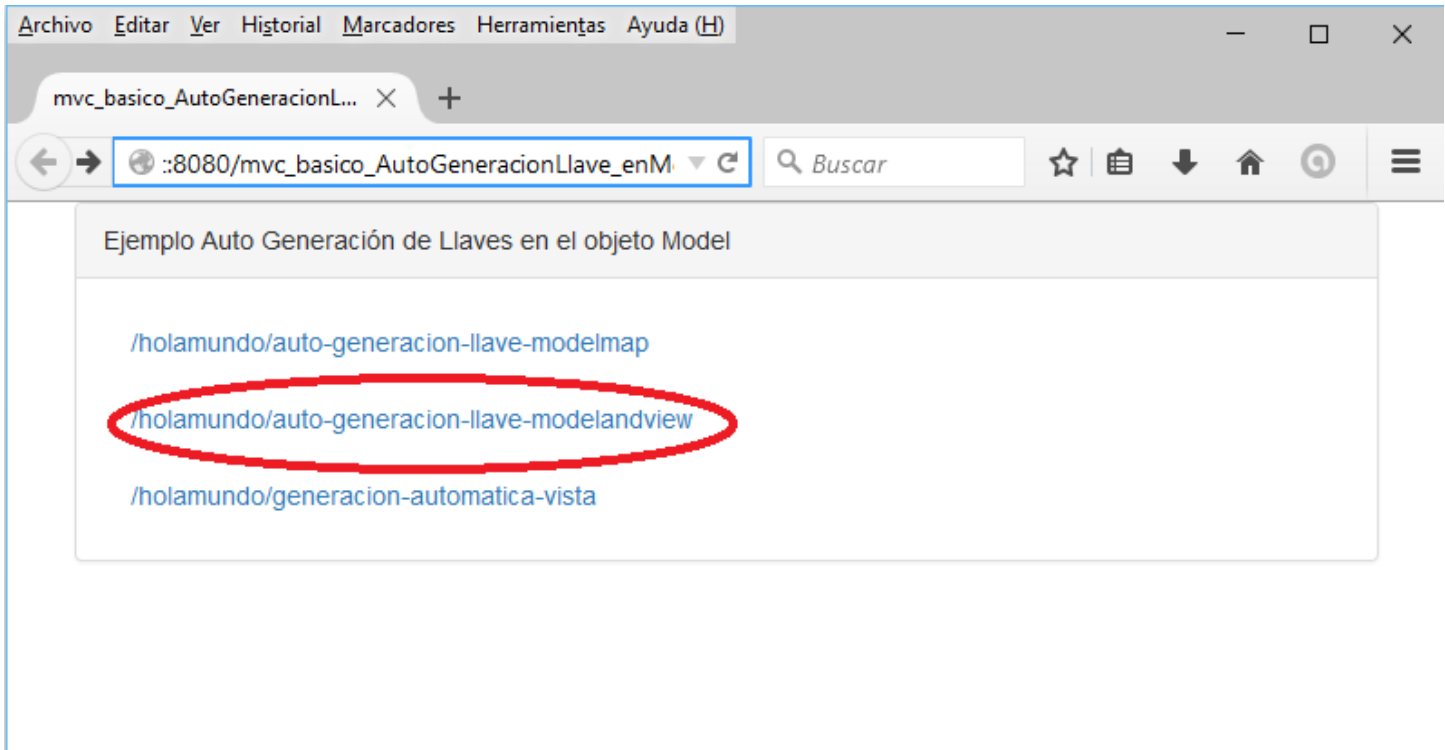
### Detalle de compras

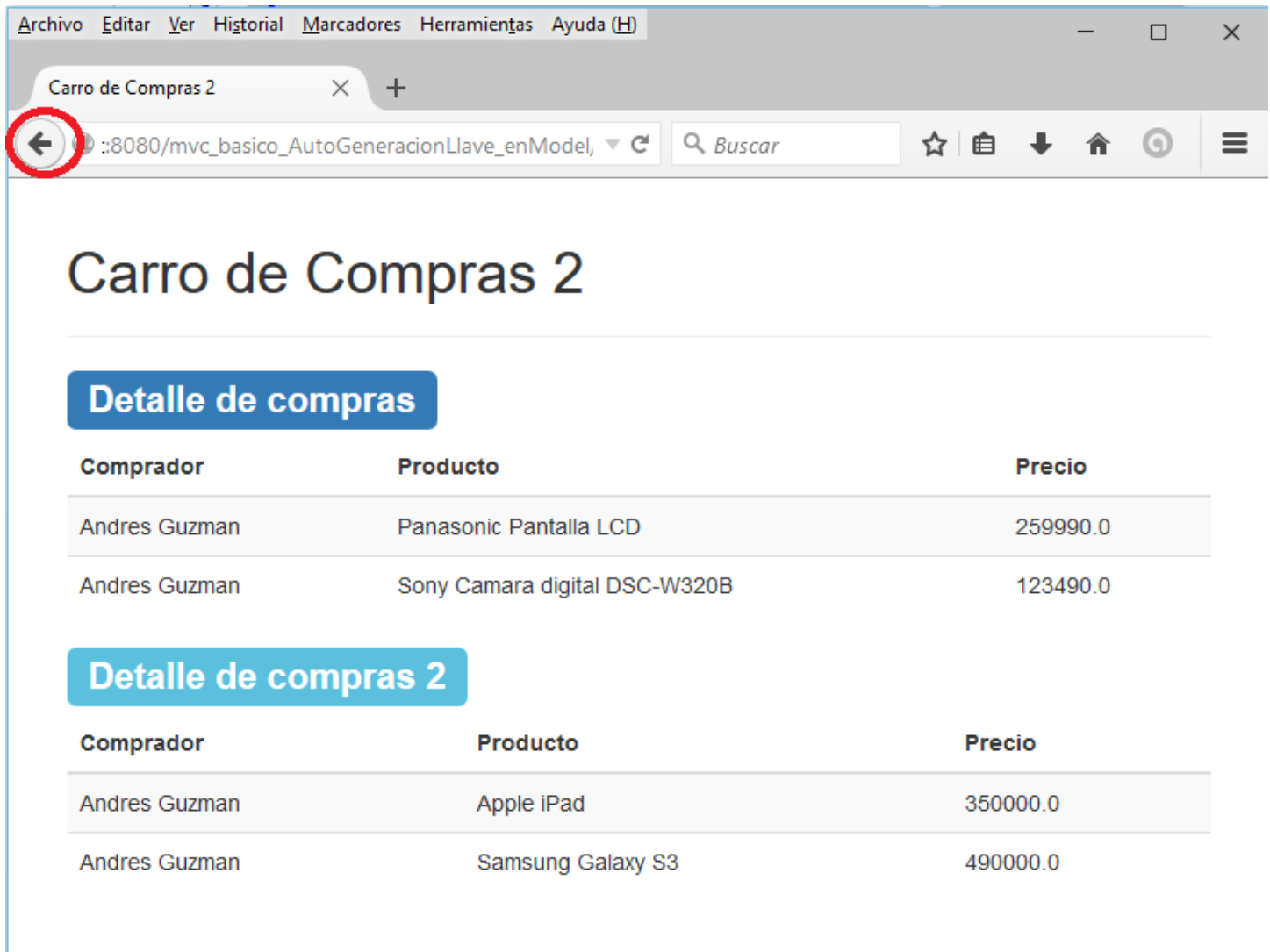
Comprador	Producto	Precio
Andres Guzman	Panasonic Pantalla LCD	259990.0
Andres Guzman	Sony Camara digital DSC-W320B	123490.0

### Detalle de compras 2

Comprador	Producto	Precio
Andres Guzman	Apple iPad	350000.0
Andres Guzman	Samsung Galaxy S3	490000.0

7. Clic en auto generación de llaves en atributos para ModelAndView.





Archivo Editar Ver Historial Marcadores Herramientas Ayuda (H)

Carro de Compras 2

http://localhost:8080/mvc\_basico\_AutoGeneracionLlave\_enModel/ Buscar

## Carro de Compras 2

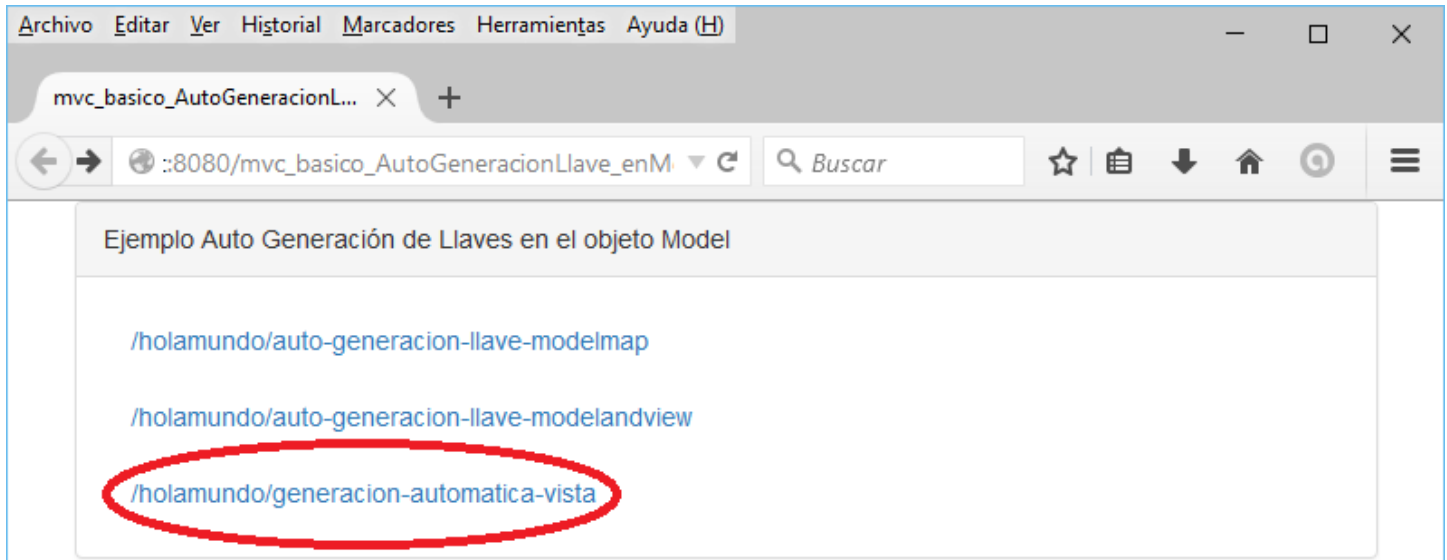
### Detalle de compras

Comprador	Producto	Precio
Andres Guzman	Panasonic Pantalla LCD	259990.0
Andres Guzman	Sony Camara digital DSC-W320B	123490.0

### Detalle de compras 2

Comprador	Producto	Precio
Andres Guzman	Apple iPad	350000.0
Andres Guzman	Samsung Galaxy S3	490000.0

8. Clic en auto generación de nombre de vistas en ModelAndView.





Archivo Editar Ver Historial Marcadores Herramientas Ayuda (H)

Generación automática de vist... X +

← ::8080/mvc\_basico\_AutoGeneracionLlave\_enModel, ↻ 🔍 Buscar ☆ 📄 ⬇️ 🏠 ↻ ☰

## Generación automática de vista en ModelAndView

### Detalle de compras

Comprador	Producto	Precio
Andres Guzman	Panasonic Pantalla LCD	259990.0
Andres Guzman	Sony Camara digital DSC-W320B	123490.0

### Detalle de compras 2

Comprador	Producto	Precio
Andres Guzman	Apple iPad	350000.0
Andres Guzman	Samsung Galaxy S3	490000.0

## 9. Abrir y estudiar la clase controladora

**/src/main/java/com.bolsadeideas.ejemplos/EjemploController.java**

```
package com.bolsadeideas.ejemplos;
import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

@Controller
@RequestMapping("/holamundo")
public class EjemploController {

    // Generación automática de llaves en atributos para ModelMap
    @RequestMapping(value = "/auto-generacion-llave-modelmap", method = RequestMethod.GET)
    public String handleRequest1(ModelMap modelMap) {

        // Tome nota de que carroCompras es un List del tipo Item
        List<Item> carroCompras = new ArrayList<Item>();
        carroCompras.add(new Item("Panasonic Pantalla LCD", 259990.0));
        carroCompras.add(new Item("Sony Cámara digital DSC-W320B", 123490.0));

        Usuario usuario = new Usuario("Andrés Guzmán");

        // Esto es lo mismo que modelMap.addAttribute("itemList", carroCompras);
        modelMap.addAttribute(carroCompras); // "itemList" es generado
            // automáticamente
            // como una llave en el Map

        // Esto es lo mismo que modelMap.addAttribute("usuario", usuario);
        modelMap.addAttribute(usuario); // "usuario" es generado automáticamente
            // como una
            // llave en el Map

        return "verCarro";
    }

    // Generación automática de llaves en atributos para ModelAndView
    @RequestMapping(value = "/auto-generacion-llave-modelandview", method = RequestMethod.GET)
    // El tipo de retorno es ModelAndView
    // El objeto ModelAndView es creado dentro del método handler
    public ModelAndView handleRequest2() {

        List<Item> carroCompras = new ArrayList<Item>();
        carroCompras.add(new Item("Panasonic Pantalla LCD", 259990.0));
```

```
carroCompras.add(new Item("Sony Cámara digital DSC-W320B", 123490.0));

Usuario usuario = new Usuario("Andrés Guzmán");

// "verCarro2" la vista jsp
ModelAndView mav = new ModelAndView("verCarro2");

// Esto es lo mismo que mav.addObject("itemList", carroCompras);
mav.addObject(carroCompras); // "itemList" es generado
                             // automáticamente como una llave en
                             // ModelAndView

// Esto es lo mismo que mav.addObject("usuario", usuario);
mav.addObject(usuario); // "usuario" es generado automáticamente como
                        // una llave

return mav;
}

// Generación automática de vista en ModelAndView
@RequestMapping(value = "/generacion-automatica-vista", method = RequestMethod.GET)
// El tipo de retorno es ModelAndView
// El objeto ModelAndView es creado dentro del método handler
public ModelAndView handleRequest3() {

    List<Item> carroCompras = new ArrayList<Item>();
    carroCompras.add(new Item("Panasonic Pantalla LCD", 259990.0));
    carroCompras.add(new Item("Sony Cámara digital DSC-W320B", 123490.0));

    Usuario usuario = new Usuario("Andrés Guzmán");

    // "generacion-automatica-vista" es nombre de la vista auto generada
    // ya que no se asigna de forma explícita
    ModelAndView mav = new ModelAndView();

    // Esto es lo mismo que mav.addObject("itemList", carroCompras);
    mav.addObject(carroCompras); // "itemList" es generado automáticamente
                                // como una llave

    // Esto es lo mismo que as mav.addObject("usuario", usuario);
    mav.addObject(usuario); // "usuario" es generado automáticamente como
                            // una llave

    return mav;
}
```

```
// Otra forma de crear atributos del model (hacia la vista)
@ModelAttribute("itemList2")
public List<Item> populateSubjectList() {

    List<Item> carroCompras = new ArrayList<Item>();
    carroCompras.add(new Item("Apple iPad", 350000.0));

    carroCompras.add(new Item("Samsung Galaxy S3", 490000.0));

    return carroCompras;
}

}
```

---

## 10. Abrir y estudiar la clase Item.java

---

```
package com.bolsadeideas.ejemplos;

public class Item {
    private String descripcion;
    private double precio;

    public Item(String descripcion, double precio) {
        this.descripcion = descripcion;
        this.precio = precio;
    }

    public String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }

    public double getPrecio() {
        return precio;
    }

    public void setPrecio(double precio) {
        this.precio = precio;
    }
}
```

---

## 11. Abrir y estudiar la clase Usuario.java

---

```
package com.bolsadeideas.ejemplos;

public class Usuario {
    private String nombre;

    public Usuario(String nombre) {
        super();
        this.nombre = nombre;
    }

    public String getNombre() {
        return nombre;
    }

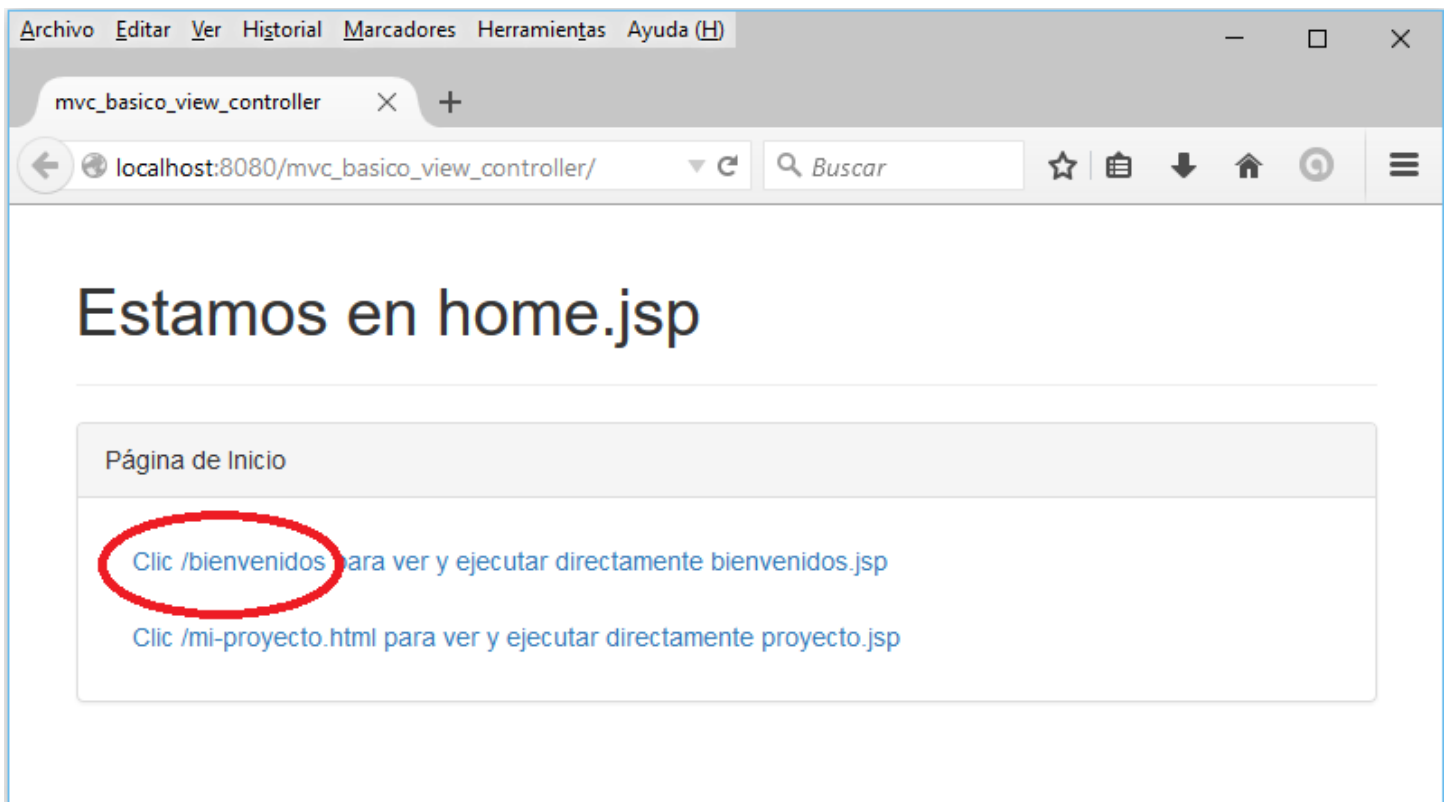
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}
```

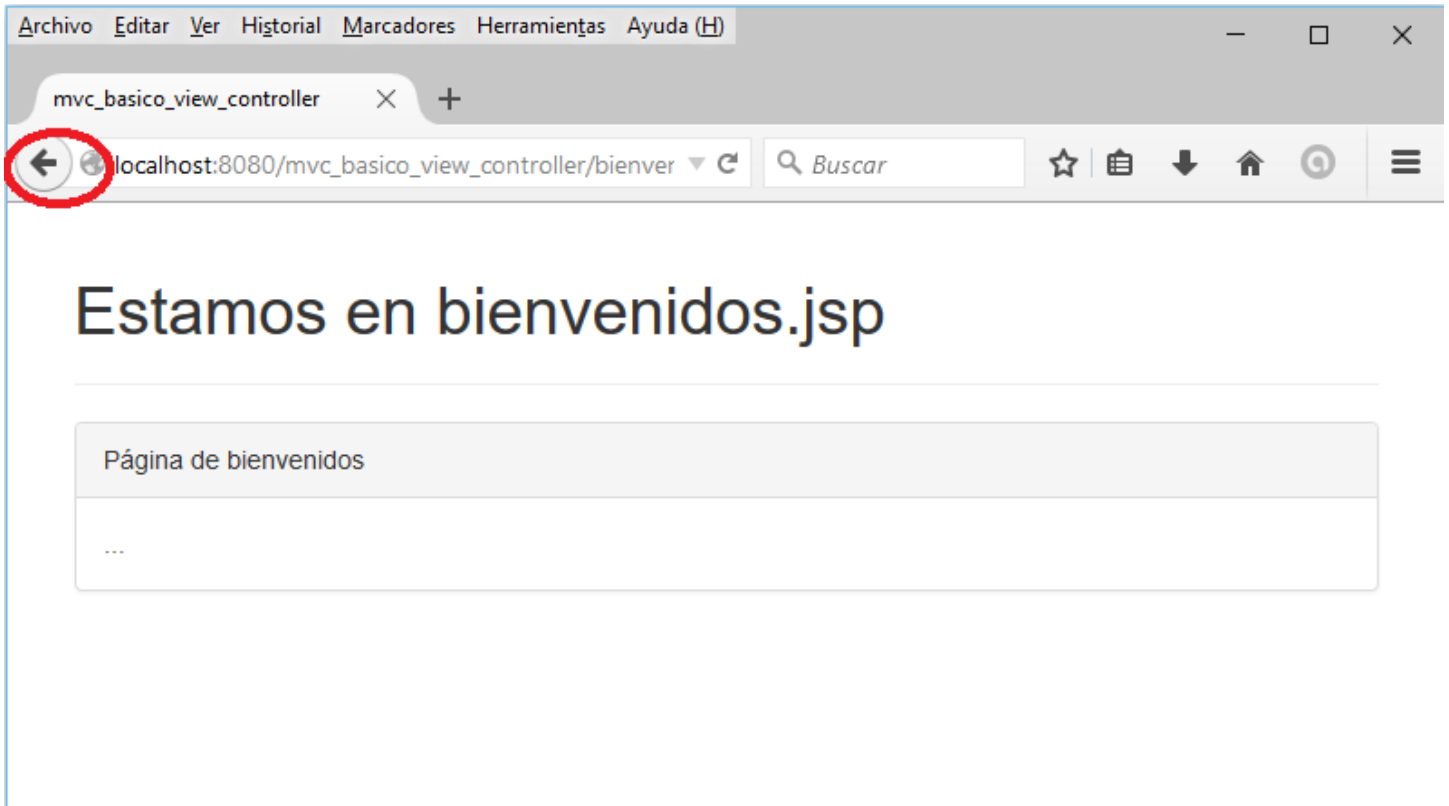
---

## Ejercicio 5: <mvc:view-controller ..> - generar y ejecutar el ejemplo "mvc\_basico\_view\_controller"

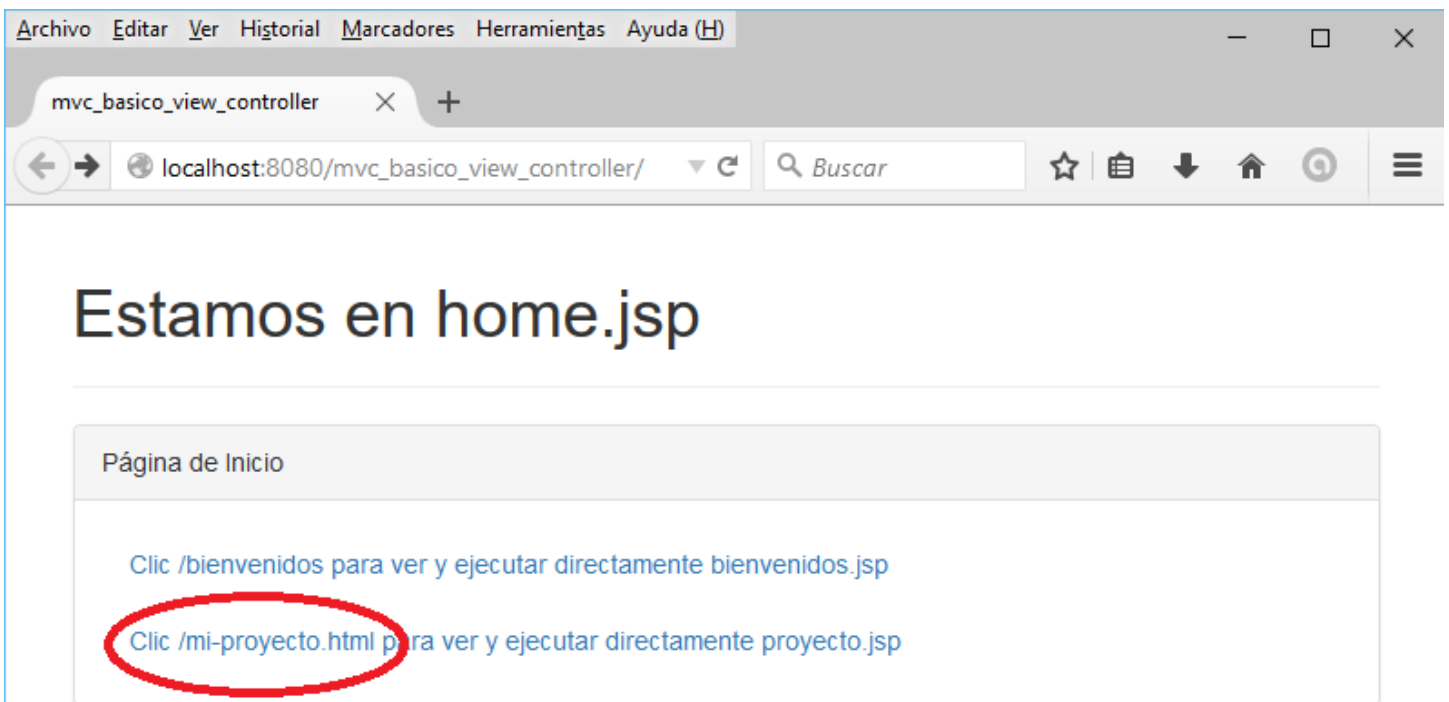
<mvc:view-controller ..> Esta etiqueta es usada en el archivo de configuración de Spring para mapear un determinado request o url directamente hacia una vista, la usamos cuando no necesitamos implementar ningún tipo de lógica del controlador, por lo tanto el controlador es obviado y se usa uno por defecto de spring que sólo carga la vista especificada.

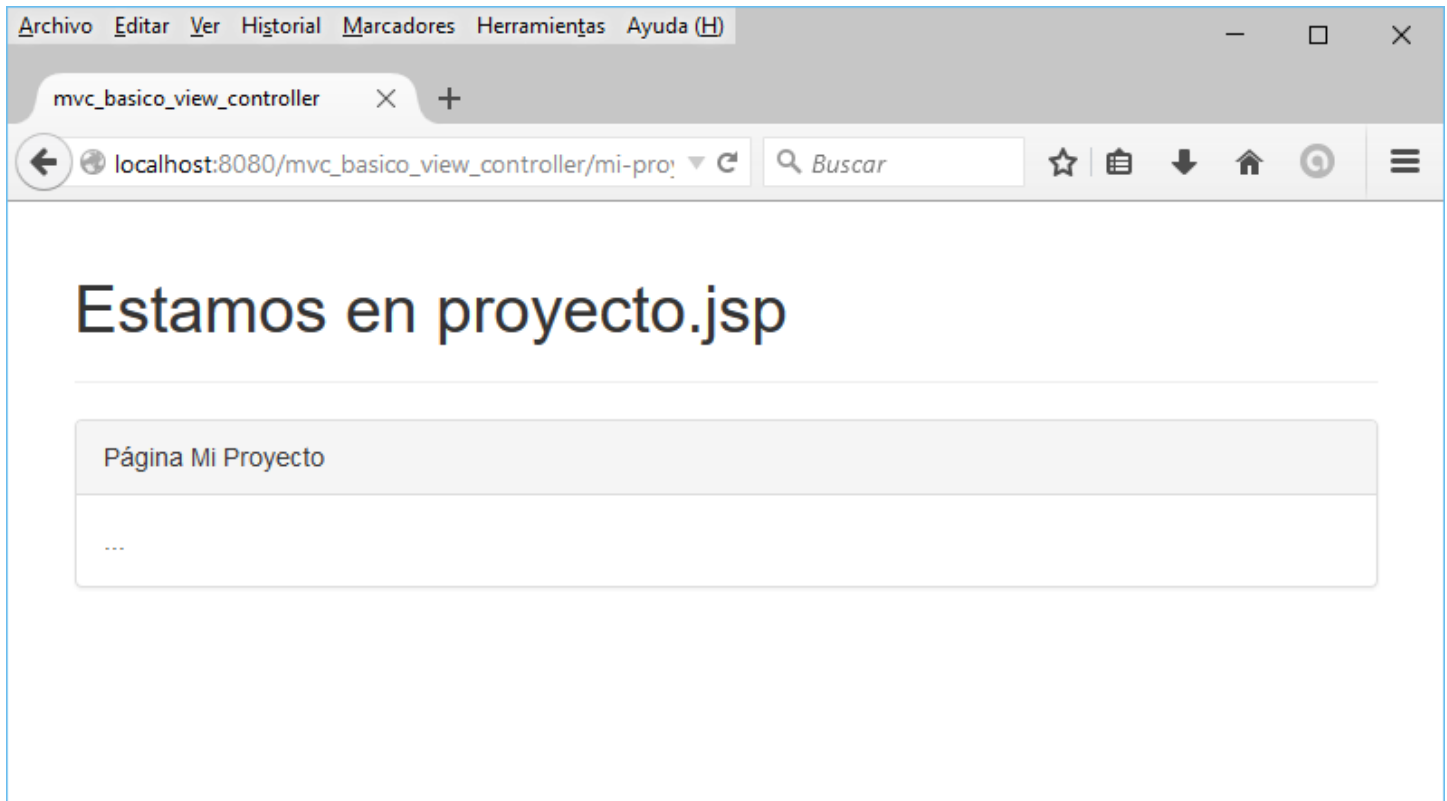
1. Clic derecho sobre el proyecto y "Run As"-"Maven Clean".
2. Clic derecho sobre el proyecto y "Run As"-"Maven Install".
3. Hacemos un **Maven->Update Project...**
4. Clic derecho sobre el proyecto y "Run As"-"Run on Server".
5. Observe el resultado.
6. Clic en bienvenidos.





7. Clic en proyecto.







## 8. Abrir y estudiar archivo contexto de spring servlet-context.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xsi:schemaLocation="
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- Habilita la anotacion de Spring MVC @Controller -->
    <annotation-driven />

    <!-- La url /bienvenidos inmediatamente esta mapeada hacia la vista bienvenidos -->
    <view-controller path="/bienvenidos" view-name="bienvenidos" />
    <!-- La url /mi-proyecto.html inmediatamente esta mapeada hacia la vista proyecto -->
    <view-controller path="/mi-proyecto.html" view-name="proyecto" />

    <!-- Resuelve la ubicacion de las vistas .jsp de @Controllers
    en la ruta /WEB-INF/views -->
    <beans:bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <beans:property name="prefix" value="/WEB-INF/views/" />
        <beans:property name="suffix" value=".jsp" />
    </beans:bean>

    <!-- Imports user-defined @Controller beans that process client requests -->
    <beans:import resource="controllers.xml" />
</beans:beans>

```

## 9. Abrir y estudiar vista home.jsp

```

<html>
<head>
    <title>Página de Inicio</title>
</head>
<body>
<h1>
Estamos en inicio.jsp.
</h1>

    <a href="/mvc_basico_view_controller/bienvenidos">Clic /bienvenidos para ver y ejecutar
        directamente bienvenidos.jsp</a>
    <br />
    <a href="/mvc_basico_view_controller/mi-proyecto.html">Clic /mi-proyecto.html para ver y
ejecutar directamente proyecto.jsp</a>
    <br />
</body>
</html>

```

## 10. Abrir y estudiar vista bienvenidos.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>
<title>Bienvenidos</title>
</head>
<body>
  <h1>Estamos en bienvenidos.jsp.</h1>
</body>
</html>
```

## 11. Abrir y estudiar vista proyecto.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>
<title>Mi Proyecto</title>
</head>
<body>
  <h1>Estamos en proyecto.jsp.</h1>
</body>
</html>
```

## 12. Modificar el proyecto

- Agregar otro mapping a elección como el ejemplo de abajo y ejecutar la url.

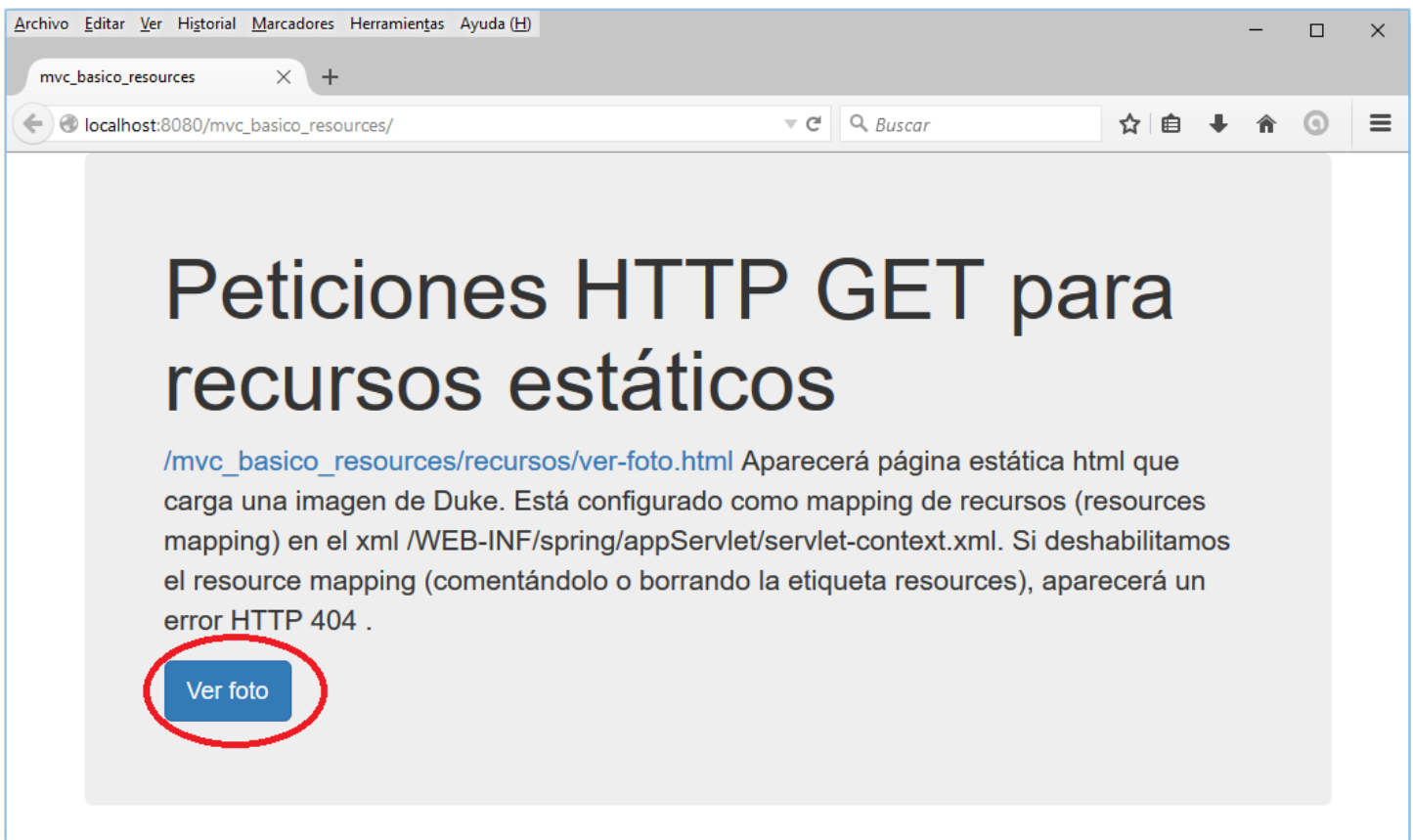
```
<view-controller path="/mi-gran-proyecto" view-name="otro-proyecto" />
```

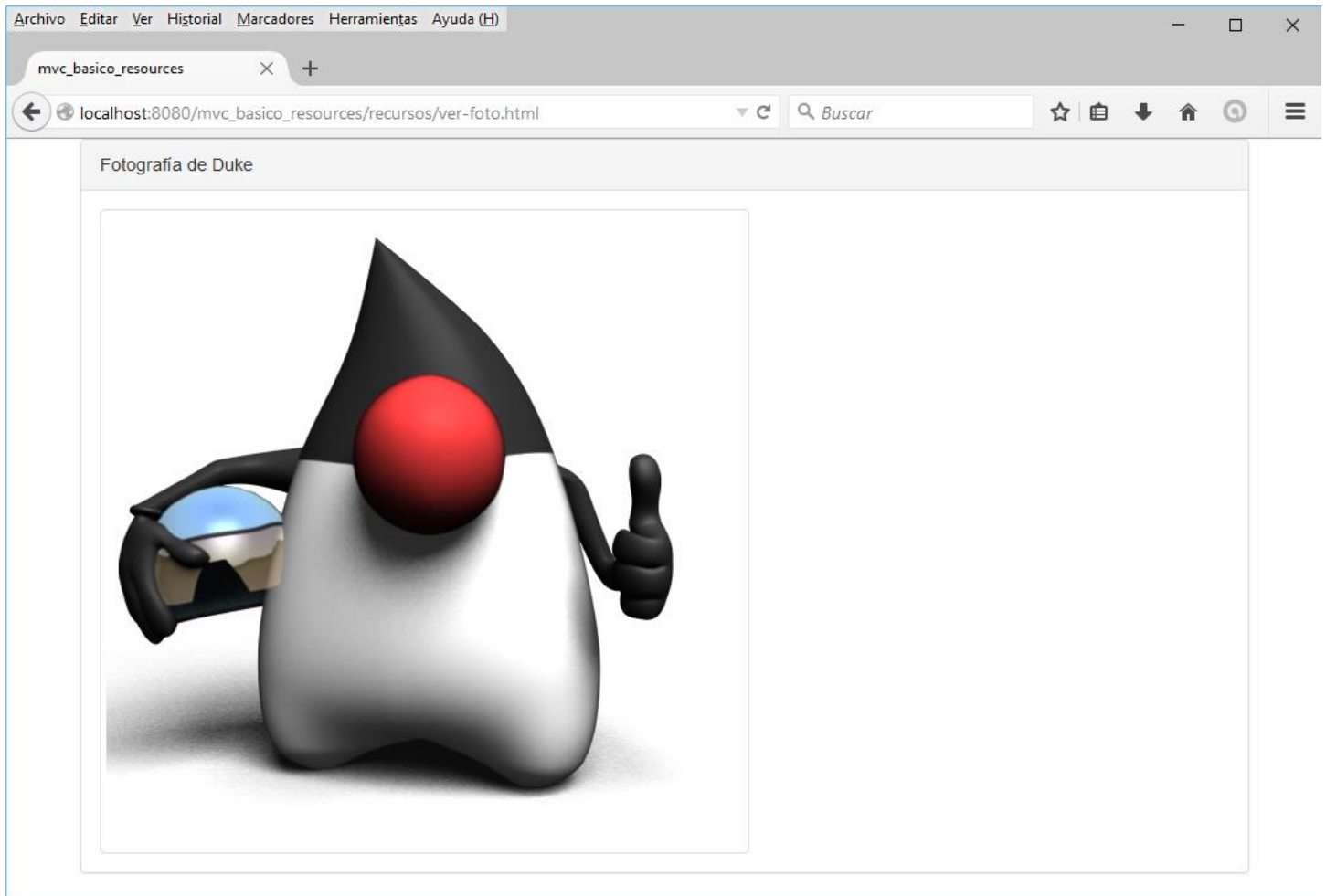
- Crear la vista otro-proyecto.jsp relacionada.

## Ejercicio 6: <mvc:resources ..> - generar y ejecutar el ejemplo "mvc\_basico\_resources" project

<mvc:resources> provee una vía para controlar las peticiones HTTP GET de recursos estáticos como por ejemplo imágenes, que se almacenan en un directorio de recurso predeterminado dentro del web application root.

1. Clic derecho sobre el proyecto y "Run As"->"Maven Clean".
2. Clic derecho sobre el proyecto y "Run As"->"Maven Install".
3. Hacemos un **Maven->Update Project...**
4. Clic derecho sobre el proyecto y "Run As"->"Run on Server".
5. Observe el resultado.
6. Clic en bienvenidos.





## 7. Abrir y estudiar archivo contexto de spring servlet-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xsi:schemaLocation="
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- Habilita la anotacion de Spring MVC @Controller -->
    <annotation-driven />

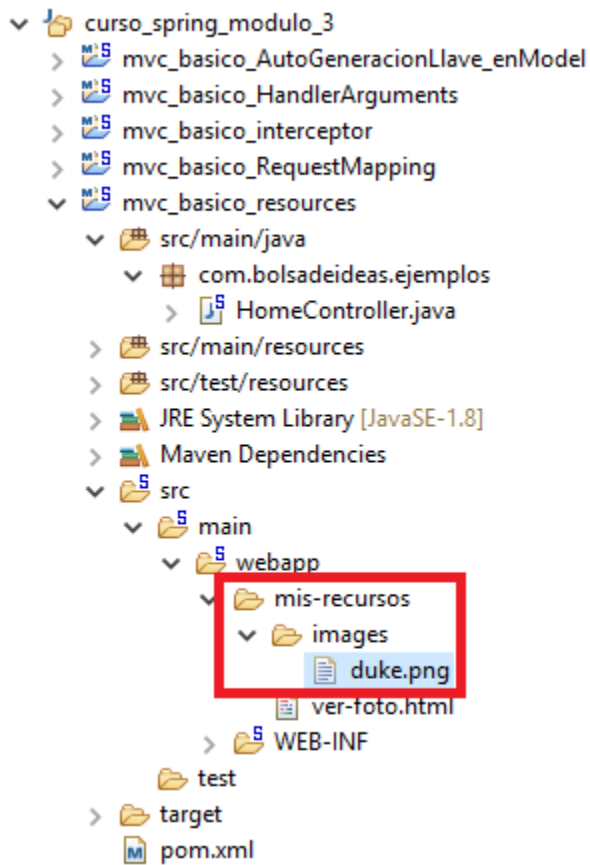
    <!-- Controla las peticiones HTTP GET /recursos/** para los recursos estáticos
        (ej: imagenes), estos se almacenan en el directorio ${webappRoot}/mis-recursos -->
    <resources mapping="/recursos/**" location="/mis-recursos/"
        cache-period="10000" />

    <!-- Resuelve la ubicion de las vistas .jsp de @Controllers en la ruta /WEB-INF/views -->
    <beans:bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <beans:property name="prefix" value="/WEB-INF/views/" />
        <beans:property name="suffix" value=".jsp" />
    </beans:bean>

    <!-- Imports user-defined @Controller beans that process client requests -->
    <beans:import resource="controllers.xml" />

</beans:beans>
```

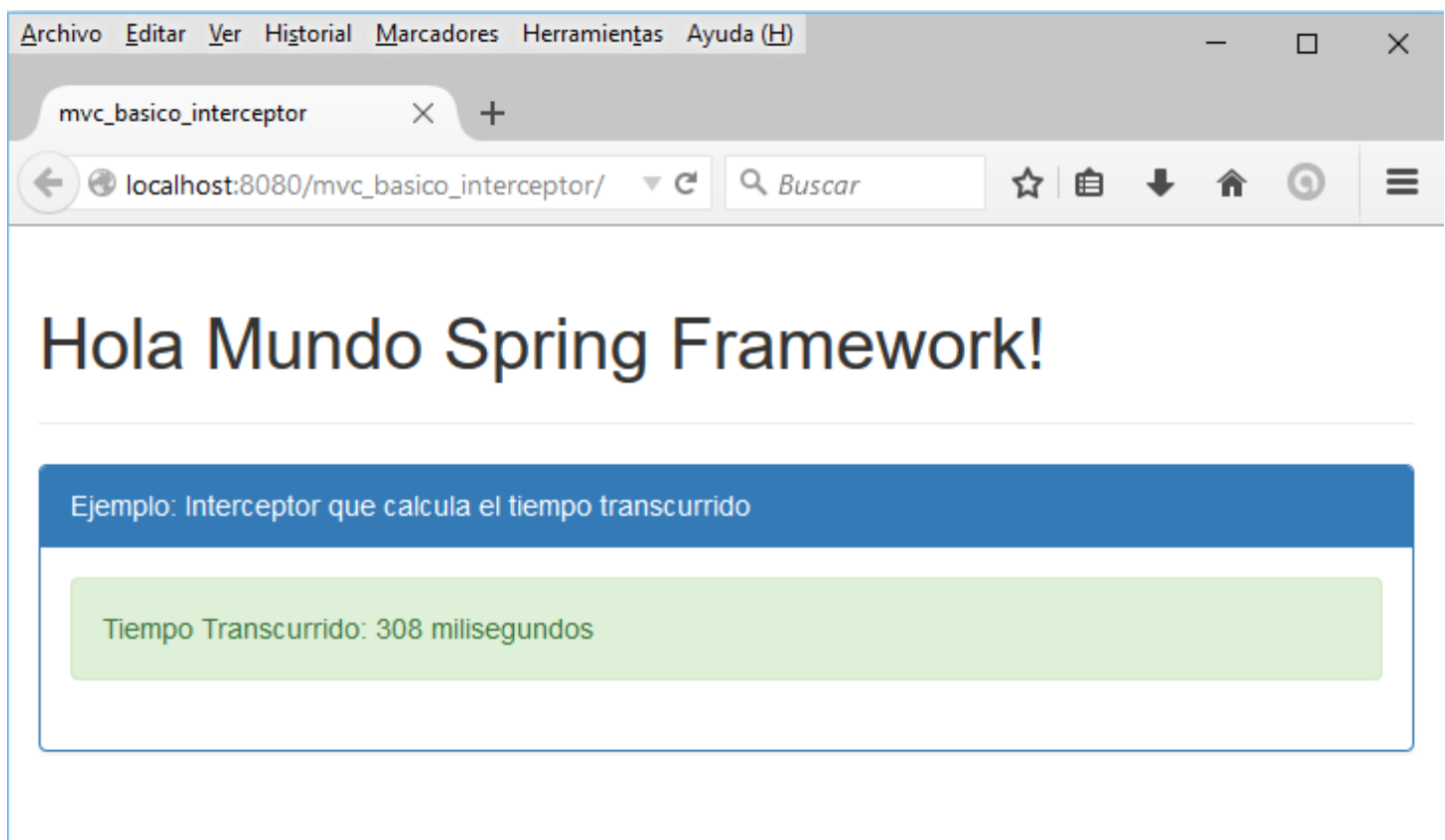
## 8. Ver la imagen Duke.png



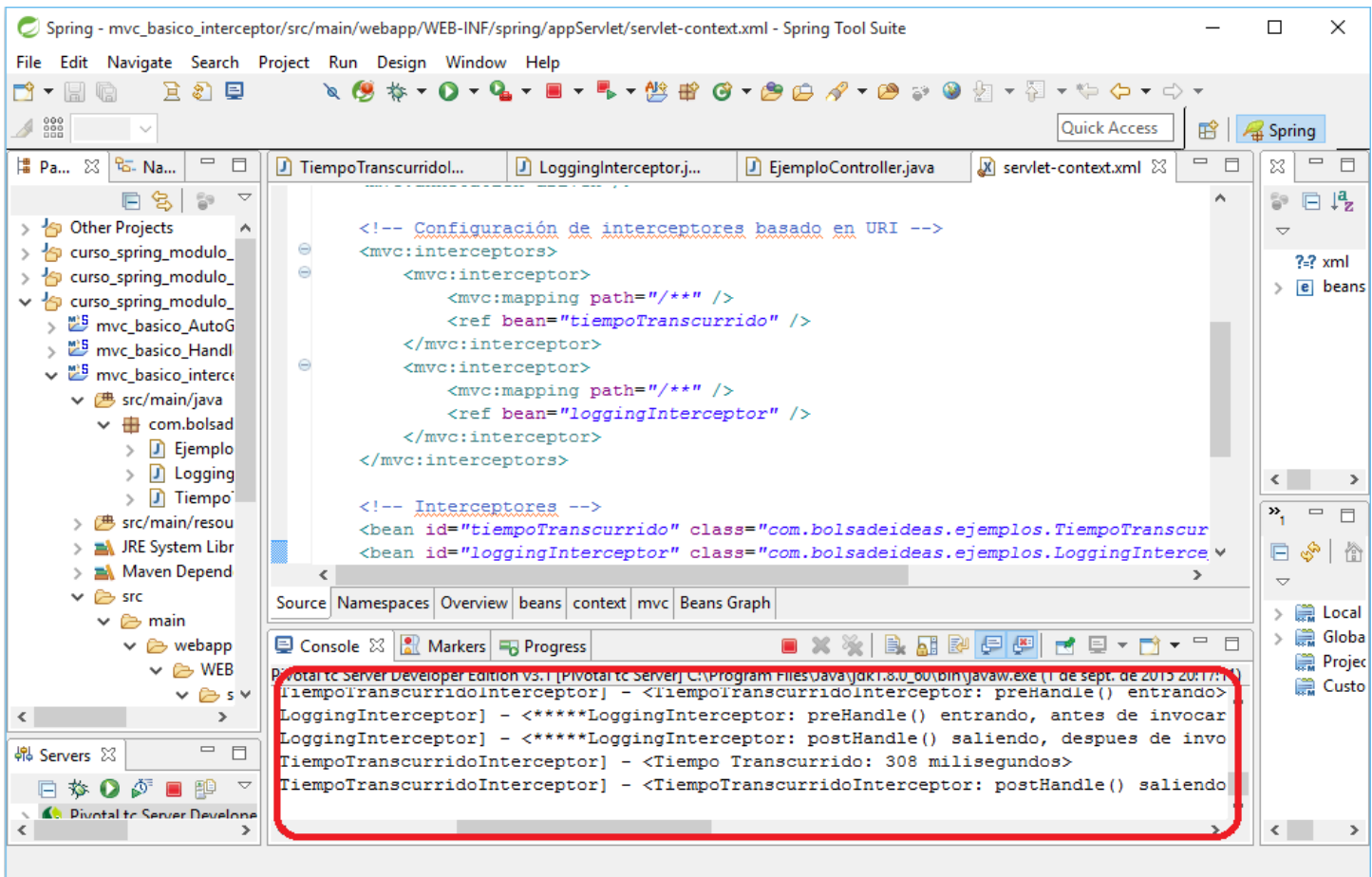
## Ejercicio 7: Generar y ejecutar el ejemplo "mvc\_basico\_interceptor"

En el siguiente ejemplo aprenderemos a crear y configurar interceptores y además cómo obtener y asignar atributos del request dentro de un interceptor.

1. Clic derecho sobre el proyecto y "Run As"->"Maven Clean".
2. Clic derecho sobre el proyecto y "Run As"->"Maven Install".
3. Hacemos un **Maven->Update Project...**
4. Clic derecho sobre el proyecto y "Run As"->"Run on Server".
5. Observe el resultado.
6. Observe el tiempo transcurrido que ha sido calculado por el interceptor.



7. Observe que se muestran los mensajes logging en la consola generado por el segundo interceptor: LoggingInterceptor.





8. Abrir y estudiar la clase `Interceptor TiempoTranscurridoInterceptor`.

```
package com.bolsadeideas.ejemplos;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;

public class TiempoTranscurridoInterceptor extends HandlerInterceptorAdapter {

    private static final Logger logger =
        LoggerFactory.getLogger(TiempoTranscurridoInterceptor.class);

    public boolean preHandle(HttpServletRequest request,
        HttpServletResponse response, Object handler) throws Exception {

        logger.info("TiempoTranscurridoInterceptor: preHandle() entrando");
        long tiempoInicio = System.currentTimeMillis();
        request.setAttribute("tiempoInicio", tiempoInicio);

        try {
            Random rand = new Random();
            Thread.sleep(rand.nextInt(500)); //hacemos una pausa
        } catch (InterruptedException ie) {}

        return true;
    }

    public void postHandle(HttpServletRequest request,
        HttpServletResponse response, Object handler,
        ModelAndView modelAndView) throws Exception {
        long tiempoInicio = (Long) request.getAttribute("tiempoInicio");
        request.removeAttribute("tiempoInicio");
        long tiempoFin = System.currentTimeMillis();
        long tiempoTranscurrido = tiempoFin - tiempoInicio;
        modelAndView.addObject("tiempoTranscurrido", tiempoTranscurrido);
        logger.info("Tiempo Transcurrido: " + tiempoTranscurrido + " milisegundos");
        logger.info("TiempoTranscurridoInterceptor: postHandle() saliendo");
    }
}
```

## 9. Abrir y estudiar la clase Interceptor LoggingInterceptor.

---

```
package com.bolsadeideas.ejemplos;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;

public class LoggingInterceptor extends HandlerInterceptorAdapter {

    private static final Logger logger = LoggerFactory.getLogger(LoggingInterceptor.class);

    public boolean preHandle(HttpServletRequest request,
        HttpServletResponse response, Object handler) throws Exception {
        logger.info("*****LoggingInterceptor: preHandle() entrando, antes de invocar el
método Handler");
        return true;
    }

    public void postHandle(HttpServletRequest request,
        HttpServletResponse response, Object handler,
        ModelAndView modelAndView) throws Exception {
        logger.info("*****LoggingInterceptor: postHandle() saliendo, después de invocar el
método Handler");
    }
}
```

---

## 10. Abrir y estudiar archivo configuración de spring holamundo-servlet.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:mvc="http://www.springframework.org/schema/mvc"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="
    http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd
    http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-
context.xsd">

  <!-- Escanea o busca en el package base de la aplicación clases beans anotados
  con @Components, @Controller, @Service, @Repository -->
  <context:component-scan base-package="com.bolsadeideas.ejemplos" />

  <mvc:annotation-driven />

  <!-- Configuración de interceptores basado en URI -->
  <!-- Interceptores que son aplicados a todos los controladores anotados @Controller -->
  <mvc:interceptors>
    <mvc:interceptor>
      <mvc:mapping path="/**" />
      <ref bean="tiempoTranscurrido" />
    </mvc:interceptor>
    <mvc:interceptor>
      <mvc:mapping path="/**" />
      <ref bean="loggingInterceptor" />
    </mvc:interceptor>
  </mvc:interceptors>

  <!-- Interceptores -->
  <bean id="tiempoTranscurrido" class="com.bolsadeideas.ejemplos.TiempoTranscurridoInterceptor" />
  <bean id="loggingInterceptor" class="com.bolsadeideas.ejemplos.LoggingInterceptor" />

  <!-- Resuelve la ubicion de las vistas .jsp de @Controllers en la ruta /WEB-INF/views -->
  <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/views/" />
    <property name="suffix" value=".jsp" />
  </bean>

</beans>

```

## Resumen

En este workshop, hemos construido, ejecutado y estudiado varios ejemplos de funciones avanzadas en Spring MVC, entre ellos Request mapping, Handler Arguments, URI Template, Interceptores, diferentes formas de objetos model, recursos web y view controller.

**Envía tus consultas a los foros!**

Aquí es cuando debes sacarte todas las dudas haciendo consultas en los foros correspondientes

### Lectura Recomendada y Bibliografía

- [Spring tutorials](#) del sitio krams915.blogspot.com: recomendable lectura de un artículo en inglés.
- [Spring MVC tutorial](#) de Viral Patel: recomendable lectura de un artículo en inglés.
- [Green Beans](#): Getting Started with Spring MVC: recomendable lectura de un artículo en inglés.
- [MVC Simplifications in Spring 3.0](#) por Keith Donald: lectura recomendada.