



spring
by Pivotal™



“Spring MVC – Manejo de Formulario”

Módulo 4 / 1

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

Objetivo

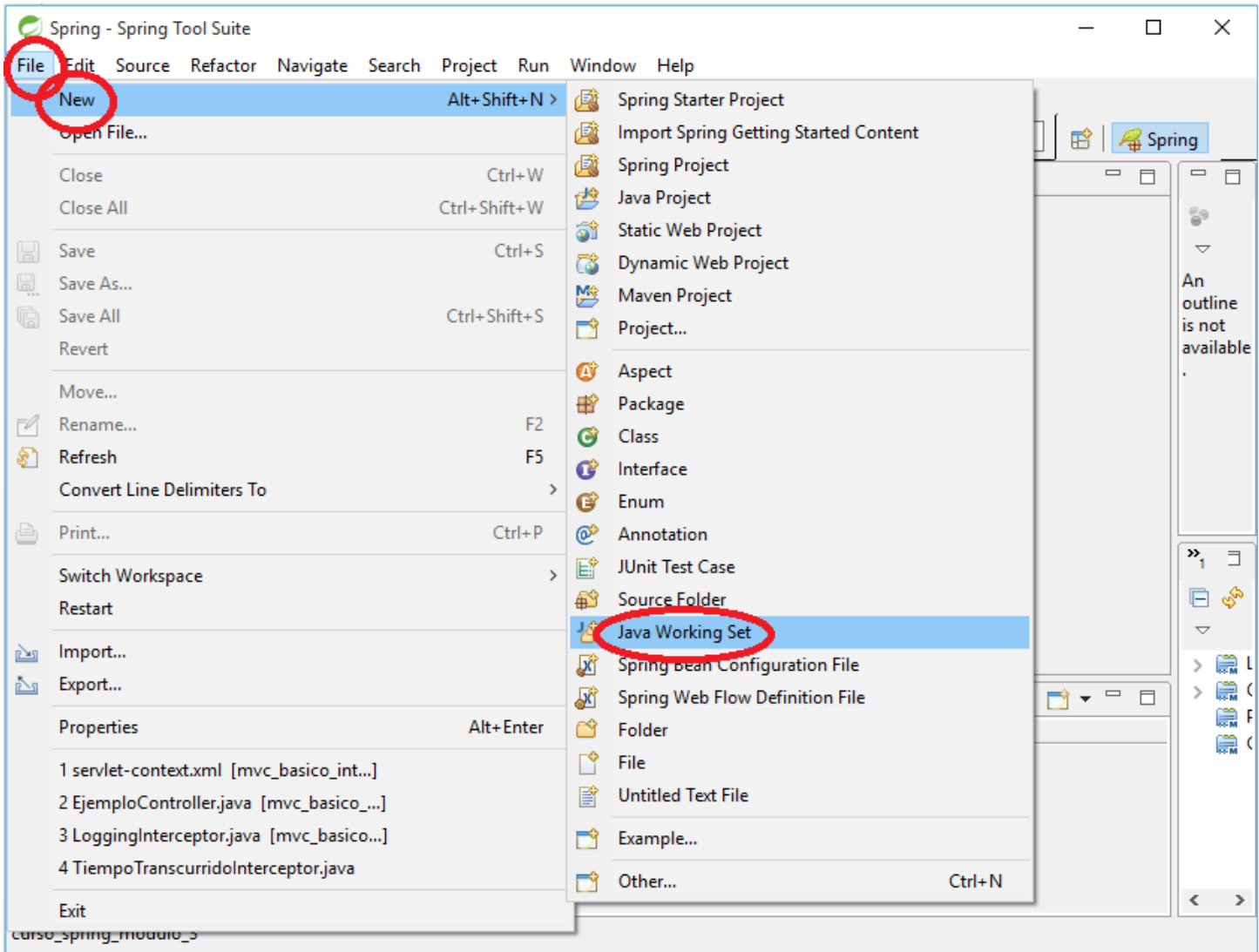
El objetivo de esta práctica es aprender a trabajar con formularios Spring MVC, que nos proporciona un completo conjunto de etiquetas para el manejo de los elementos de formulario en las vistas JSP, además nos permiten integrar y poblar directamente con los datos provenientes desde el controlador usando un conjunto de anotaciones específicas de Spring, además veremos cómo funciona el API de validación de Spring.

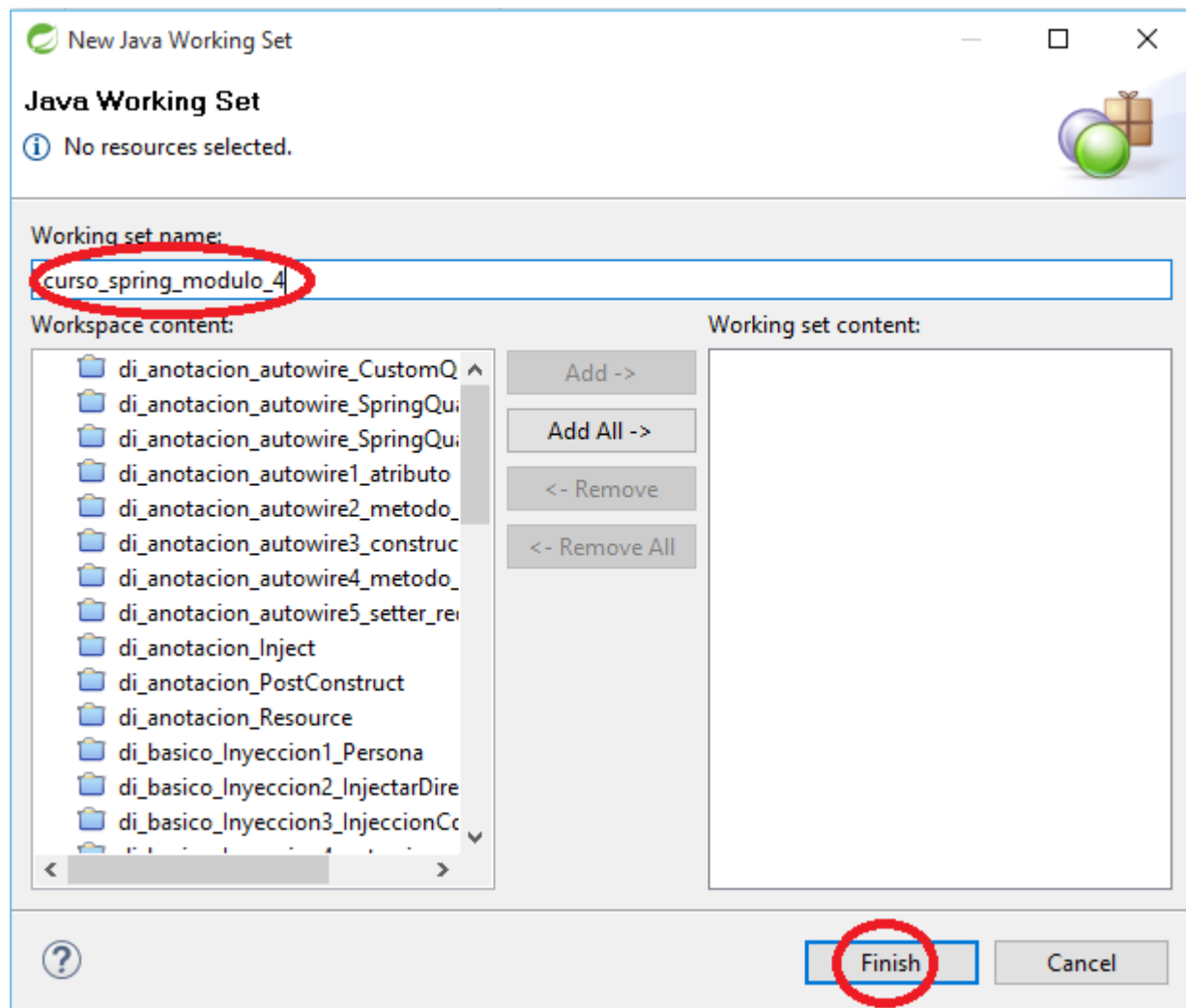
"Quemar etapas"

Es importante que saques provecho de cada módulo y consultes todos los temas que se van tratando, sin adelantar etapas.

Ejercicio 0: Importar todos los proyectos de ejemplo

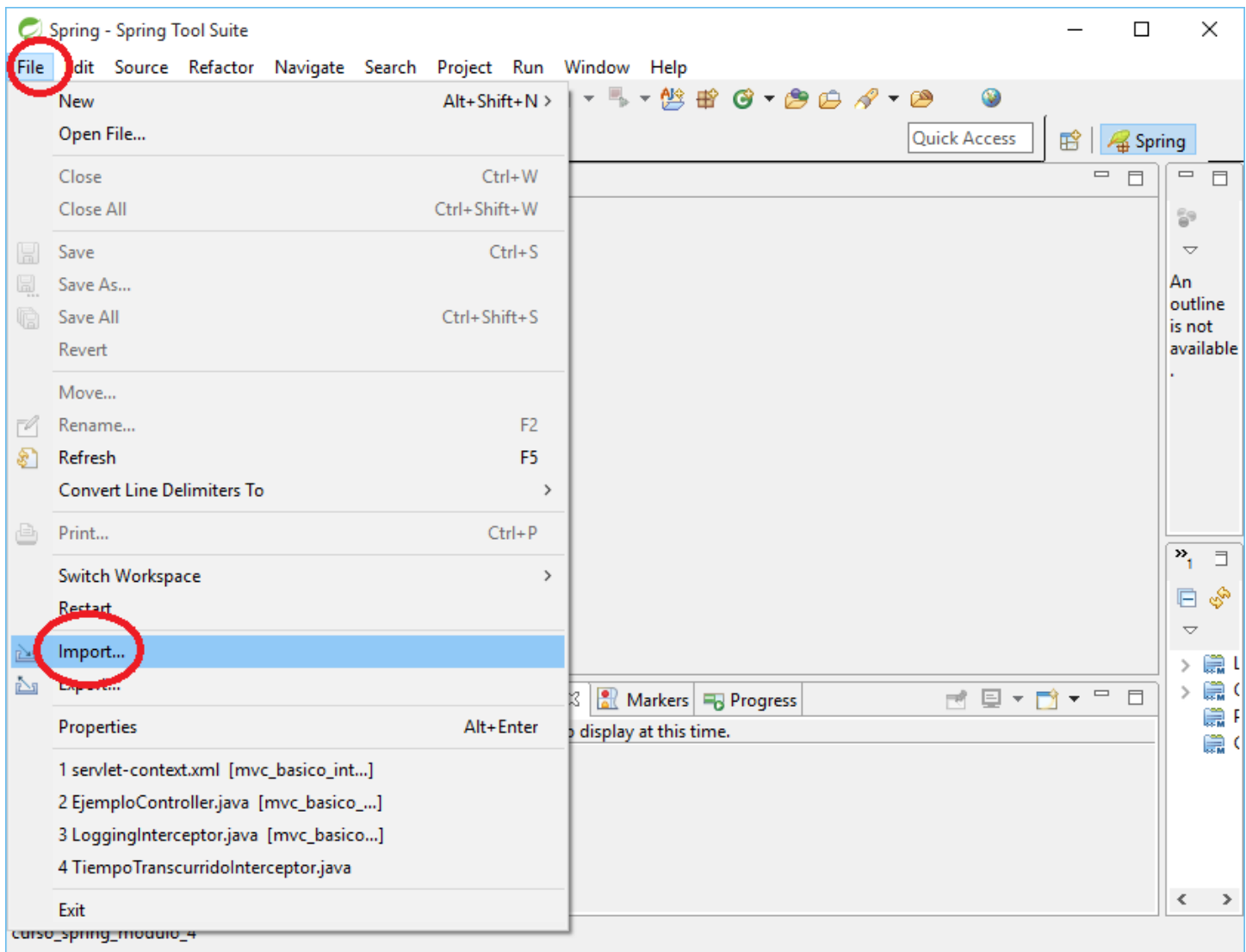
1. Crear un nuevo "Java Working Set" llamado "curso_spring_modulo_4". Esto es para organizar los proyectos bajo un esquema llamado **Working Set**, similar a como organizamos archivos en directorios.
 - Seleccionar **File->New->Java Working Set**.

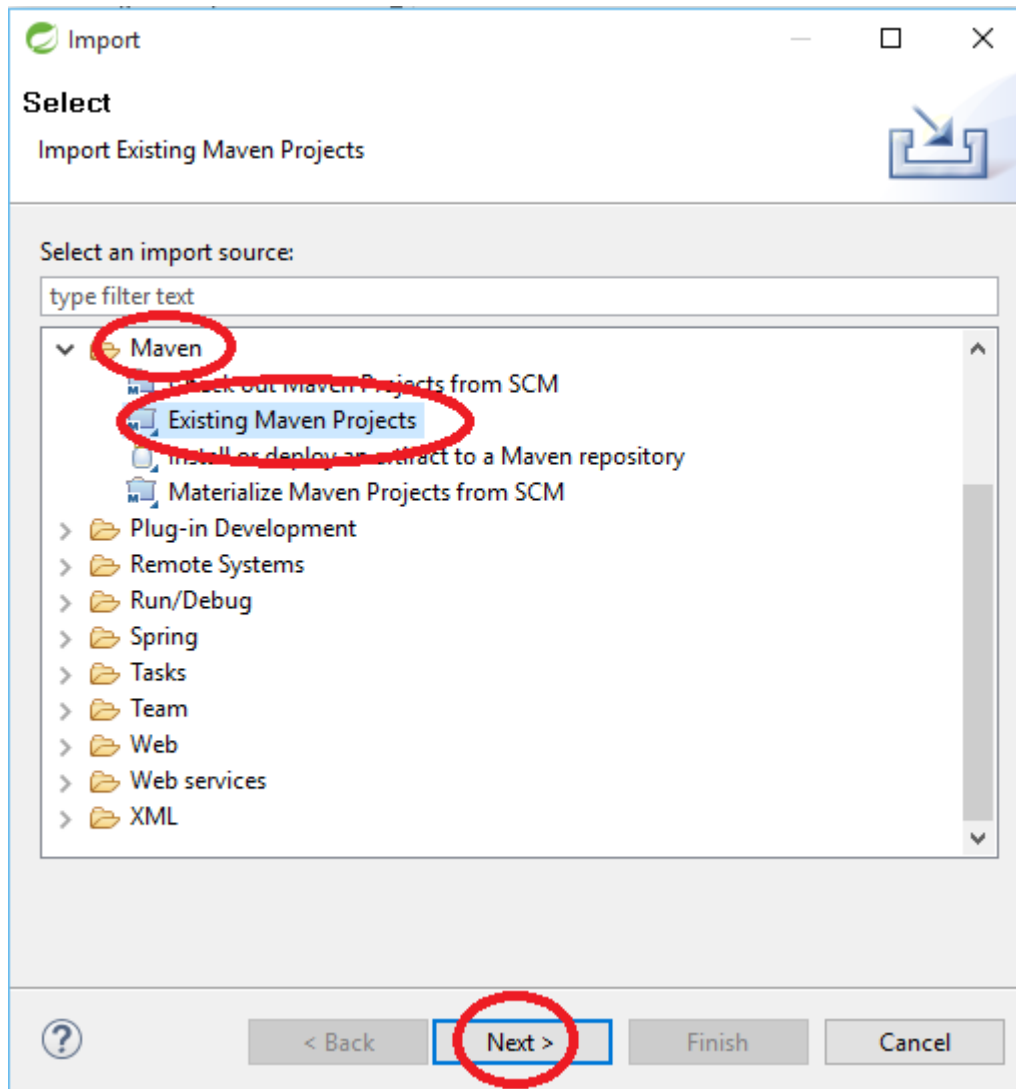




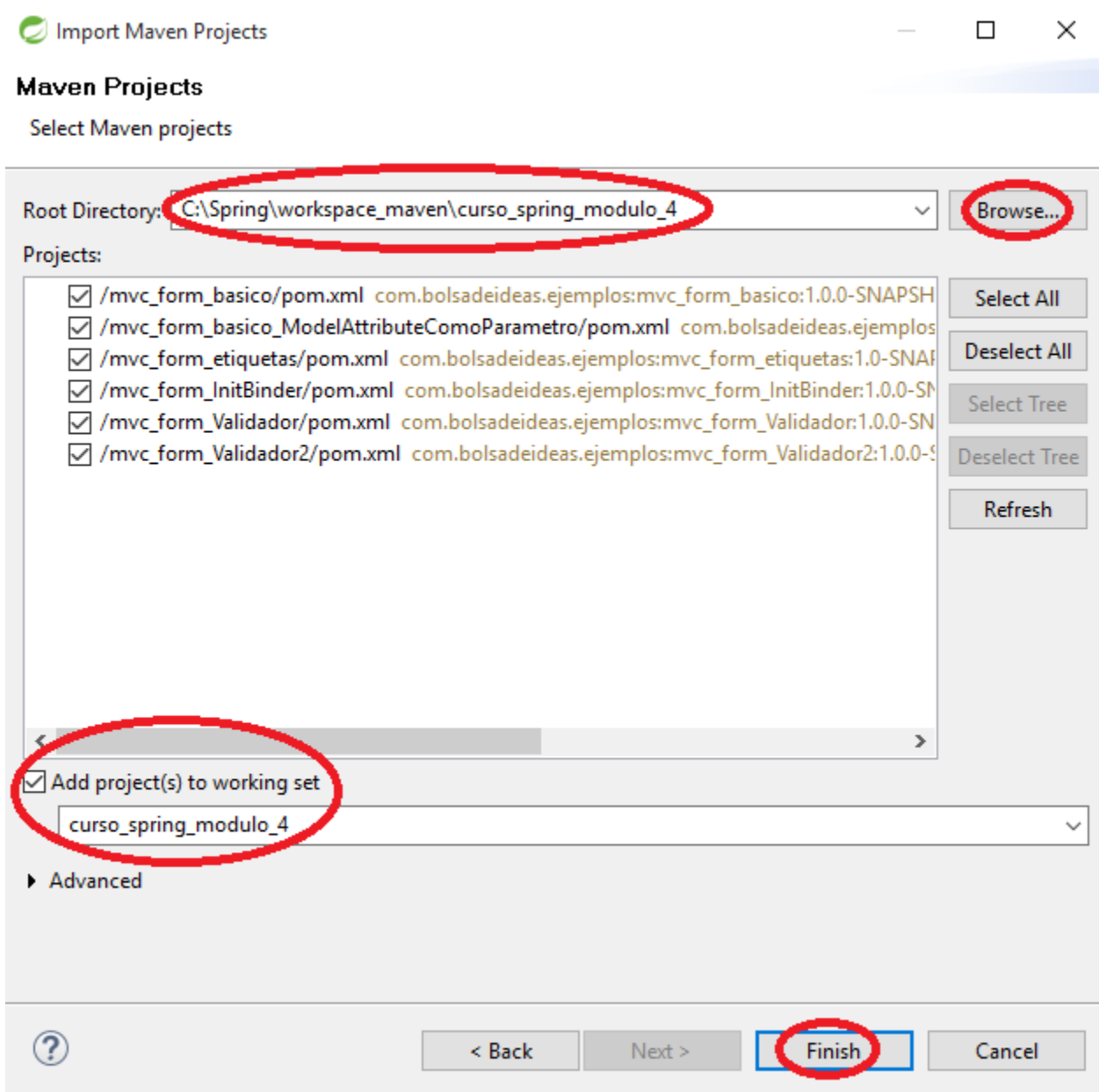
2. Importar los proyectos de ejemplos en maven.

- Seleccionar **File->Import**.





- Clic **Browse**
- Seleccionamos el directorio donde vienen los proyecto de ejemplo del laboratorio
- Agregamos los proyectos al **Working Set curso_spring_modulo_4**
- **Finish**



Ejercicio 1: Generar y ejecutar el ejemplo "mvc_form_basico"

En este ejercicio, aprenderemos lo siguiente:

- ✓ Cómo crear objeto comando (objeto que representa y contiene los datos del formulario) que puede ser accedido en las plantillas de vistas JSP.
- ✓ Cómo desplegar o crear un formulario inicial usando etiquetas form de spring.
- ✓ Cómo manejar o procesar la petición para enviar los datos del form al controlador.

1. Clic derecho sobre el proyecto y **Run As->Maven Clean**
2. Clic derecho **Run As->Maven Install**
3. Clic derecho **Maven->Update Project...**
4. Clic derecho sobre **mvc_form_basico-> Run As on Server**
5. Observe el resultado.

Archivo Editar Ver Historial Marcadores Herramientas Ayuda (H)

mvc_form_basico

localhost:8080/mvc_form_basico/ Buscar

Formulario Básico mvc_form_basico

Crear Cuenta

Nombre

Saldo

Nivel Endeudamiento

Fecha Renovación

Crear Cuenta

6. Ingresar Datos en el form

Archivo Editar Ver Historial Marcadores Herramientas Ayuda (H)

mvc_form_basico

localhost:8080/mvc_form_basico/ Buscar

Formulario Básico mvc_form_basico

Crear Cuenta

Nombre

Andrés Guzmán

Saldo

Ch\$5.000,00

Nivel Endeudamiento

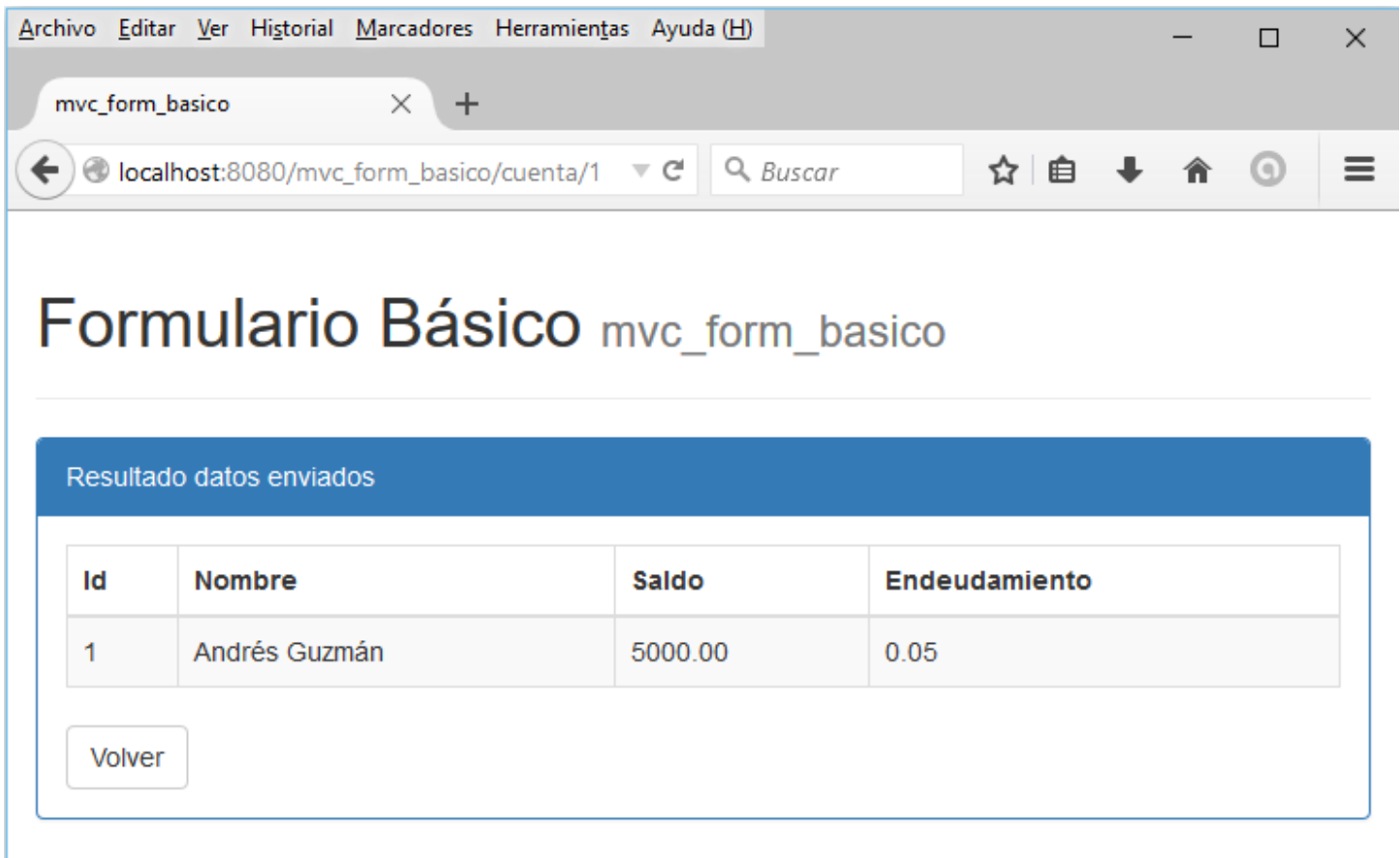
5%

Fecha Renovación

07-05-16

Crear Cuenta

7. Observe que se despliega la información de la cuenta recién creada, en la página de detalle.



Archivo Editar Ver Historial Marcadores Herramientas Ayuda (H)

mvc_form_basico

localhost:8080/mvc_form_basico/cuenta/1

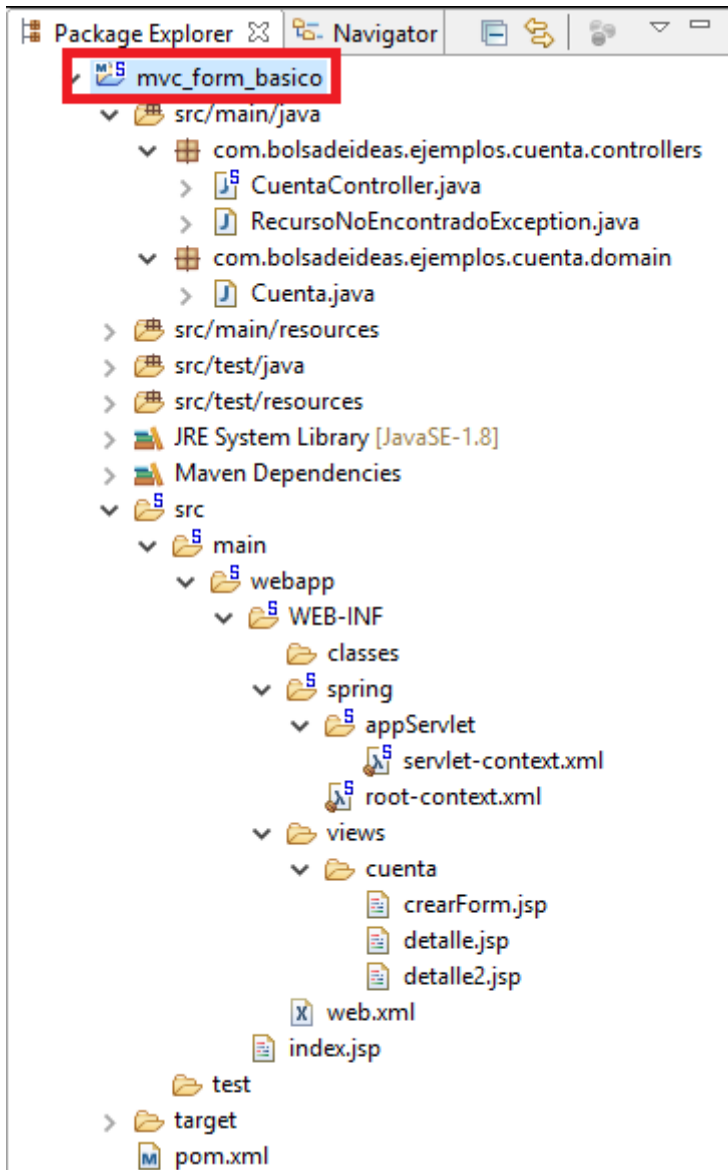
Formulario Básico mvc_form_basico

Resultado datos enviados

Id	Nombre	Saldo	Endeudamiento
1	Andrés Guzmán	5000.00	0.05

Volver

8. Estudiemos el proyecto:



9. Cuando arranca la aplicación ejecuta automáticamente el index.jsp de la raíz del webapp:

- Lo que hace es redirigir el control con forward hacia la URL /cuenta que está mapeada al controlador CuentaController de spring

```
<html>
<body>
<jsp:forward page="/cuenta"/>
</body>
</html>
```

10. Abrir y estudiar la clase controladora `/src/main/java/`

`com.bolsadeideas.ejemplos.cuenta.controllers/CuentaController.java`

- Un método handler del controlador es seleccionado para manejar la petición por defecto que crea el formulario y maneja la página inicial del request mostrando el form.
- El método handler `crearCuentaForm(Model model)` del `CuentaController` será seleccionado para controlar la petición inicial del tipo GET.
- Se crea un objeto `Cuenta` y se guarda en el atributo "`cuenta`" del objeto `model`
- Luego retornamos el nombre de la vista "`cuenta/crearForm`".

```
package com.bolsadeideas.ejemplos.cuenta.controllers;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

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

import com.bolsadeideas.ejemplos.cuenta.domain.Cuenta;

@Controller
@RequestMapping(value="/cuenta")
public class CuentaController {

    private Map<Long, Cuenta> cuentas = new ConcurrentHashMap<Long, Cuenta>();
```

```
// Metodo handler formulario, para crear la cuenta
@RequestMapping(method=RequestMethod.GET)
public String crearCuentaForm(Model model) {
    // La instancia de "Cuenta" es creada y guardada como atributo "cuenta"
    // en el objeto "model", el cual es accesible desde
    // la vista, "cuenta/crearForm.jsp".

    model.addAttribute("cuenta", new Cuenta());
    return "cuenta/crearForm";
}

// Metodo handler que procesa el envio de datos del form
@RequestMapping(method=RequestMethod.POST)
public String crearCuenta(Cuenta cuenta, BindingResult result) {
    // Si ocurre un error en la validación aparecera el formulario
    // "cuenta/crearForm.jsp" con los mensajes.
    if (result.hasErrors()) {
        return "cuenta/crearForm";
    }

    // Si todo está bien, crea la cuenta y la agrega a la lista
    // "cuentas", luego redirige hacia el detalle /cuenta/{id},
    // controlado por el método handler "verDetalle(..)" de mas abajo.
    this.cuentas.put(cuenta.asignarId(), cuenta);
    return "redirect:/cuenta/" + cuenta.getId();
}

@RequestMapping(value="{id}", method=RequestMethod.GET)
public String verDetalle(@PathVariable Long id, Model model) {
    Cuenta cuenta = this.cuentas.get(id);
    if (cuenta == null) {
        throw new RecursoNoEncontradoException(id);
    }
    model.addAttribute("cuenta", cuenta);
    // return "cuenta/detalle"; // Muestra el detalle de la cuenta en formato formulario
    return "cuenta/detalle2"; // Muestra el detalle de la cuenta en formato tabla html
}
}
```

11. Abrir y estudiar la clase Cuenta `/src/main/java/``com.bolsadeideas.ejemplos.cuenta.domain/Cuenta.java`

- Observamos que tiene algunos valores por defecto, por ejemplo `saldo` con valor **5000**

```
package com.bolsadeideas.ejemplos.cuenta.domain;

import java.math.BigDecimal;
import java.util.Date;
import java.util.concurrent.atomic.AtomicLong;

import javax.validation.constraints.Future;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

import org.springframework.format.annotation.DateTimeFormat;
import org.springframework.format.annotation.NumberFormat;
import org.springframework.format.annotation.NumberFormat.Style;

public class Cuenta {

    private Long id;

    @NotNull
    @Size(min = 1, max = 25)
    private String nombre;

    @NotNull
    @NumberFormat(style = Style.CURRENCY)
    private BigDecimal saldo = new BigDecimal("5000");

    @NotNull
    @NumberFormat(style = Style.PERCENT)
    private BigDecimal nivelEndeudamiento = new BigDecimal(".05");

    @DateTimeFormat(style = "S-")
    @Future
    private Date fechaRenovacion = new Date(new Date().getTime() + 21425000000L);

    public Long getId() {
        return id;
    }

    void setId(Long id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }
}
```

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

public BigDecimal getSaldo() {
    return saldo;
}

public void setSaldo(BigDecimal saldo) {
    this.saldo = saldo;
}

public BigDecimal getNivelEndeudamiento() {
    return nivelEndeudamiento;
}

public void setNivelEndeudamiento(BigDecimal nivelEndeudamiento) {
    this.nivelEndeudamiento = nivelEndeudamiento;
}

public Date getFechaRenovacion() {
    return fechaRenovacion;
}

public void setFechaRenovacion(Date fechaRenovacion) {
    this.fechaRenovacion = fechaRenovacion;
}

public Long asignarId() {
    this.id = idSequencia.incrementAndGet();
    return id;
}

private static final AtomicLong idSequencia = new AtomicLong();
}
```

12. Ahora estudiemos el View resolver seleccionado para manejar y resolver las vistas jsp en el archivo de configuración de spring **servlet-context.xml**.

- Ahora, basándonos sobre el retorno de la vista "cuenta/CrearForm", se selecciona la vista "cuenta/CrearForm.jsp" por su nombre, buscándola dentro de los recursos en el WEB-INF según el directorio configurado que es el **views** y la extensión **.jsp**, por lo tanto ya que sólo hay una sola configuración: InternalResourceViewResolver tomará esa regla.

```
<?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" />

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

    <!-- View Resolvers -->
    <!-- Resuelve la ubicacion 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>
```


13. Abrir y estudiar la vista jsp "cuenta/crearForm.jsp"

14. Observamos que el formulario de spring usa el atributo cuenta (del objeto model) a través del **modelAttribute** (con el valor **cuenta**). En otras palabras, accedemos al mismo objeto cuenta que se guardó en el controlador en el atributo cuenta dentro del objeto model.

```
<%@page contentType="text/html; charset=UTF-8"%>
<%@page pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<html>
<head>
  <meta content="text/html; charset=UTF-8" http-equiv="content-type" />
  <title>Crear Cuenta</title>
</head>
<body>
<div class="container">
  <h3>
    Crear Cuenta
  </h3>
  <div class="span-12 last">
    <!-- El objeto del model "cuenta" se crea dentro del metodo "crearCuentaForm(Model model)"
    de la clase controladora "CuentaController". -->
    <form:form modelAttribute="cuenta" action="cuenta" method="post">
      <fieldset>
        <legend>Cuenta</legend>
        <p>
          <form:label for="nombre" path="nombre" cssErrorClass="error">Nombre</form:label><br/>
          <form:input path="nombre" /> <form:errors path="nombre" />
        </p>
        <p>
          <form:label for="saldo" path="saldo" cssErrorClass="error">Saldo</form:label><br/>
          <form:input path="saldo" /> <form:errors path="saldo" />
        </p>
        <p>
          <form:label for="nivelEndeudamiento" path="nivelEndeudamiento" cssErrorClass="error">Nivel
          Endeudamiento</form:label><br/>
          <form:input path="nivelEndeudamiento" /> <form:errors path="nivelEndeudamiento" />
        </p>
        <p>
          <form:label for="fechaRenovacion" path="fechaRenovacion" cssErrorClass="error">Fecha
          Renovación</form:label><br/>
          <form:input path="fechaRenovacion" /> <form:errors path="fechaRenovacion" />
        </p>
        <p>
          <input type="submit" value="Crear Cuenta"/>
        </p>
      </fieldset>
    </form:form>
  </div>
</div>
</body>
</html>
```

15. Observe que los valores del formulario reflejan los valores por defecto de la clase Cuenta, por ejemplo el valor del atributo **saldo**, donde su valor se inicializa en **5000** y es justamente el mismo valor que ve muestra en la página del formulario, ya que el objeto cuenta puebla con los datos al formulario.

Archivo Editar Ver Historial Marcadores Herramientas Ayuda (H)

mvc_form_basico

localhost:8080/mvc_form_basico/ Buscar

Formulario Básico mvc_form_basico

Crear Cuenta

Nombre

Andrés Guzmán

Saldo

Ch\$5.000,00

Nivel Endeudamiento

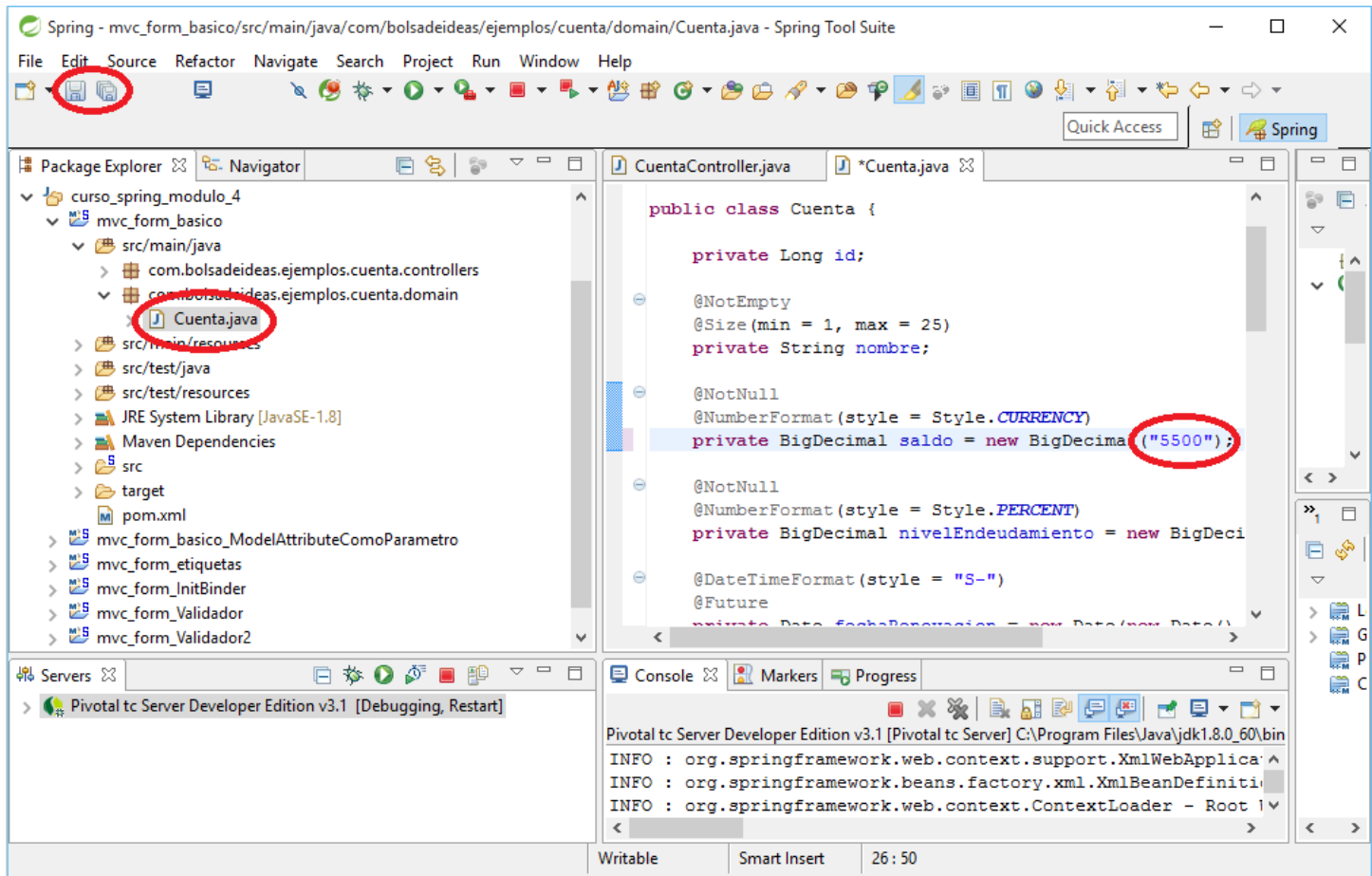
5%

Fecha Renovación

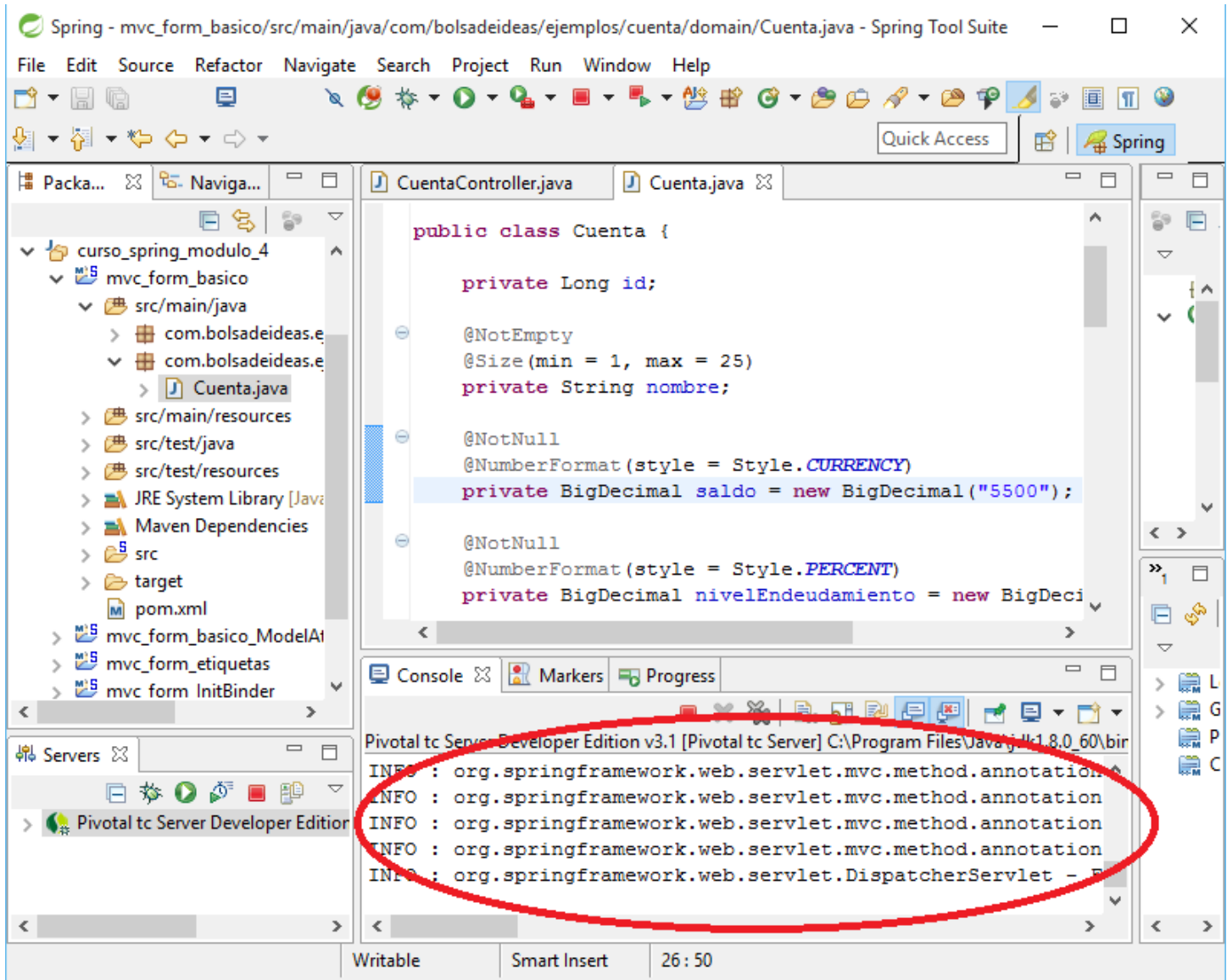
07-05-16

Crear Cuenta

16. Como ejercicio vamos a modificar el valor del atributo saldo de la clase Cuenta a **5500** y guardamos el cambio.



- Esperamos hasta que el STS re-construya y vuelva a publicar la aplicación:



- Refrescamos el navegador y observamos que se refleja el cambio del valor del saldo a 5500.

Archivo Editar Ver Historial Marcadores Herramientas Ayuda (H)

mvc_form_basico

localhost:8080/mvc_form_basico/ Buscar

Formulario Básico mvc_form_basico

Crear Cuenta

Nombre

Andrés Guzmán

Saldo

Ch\$5.500,00

Nivel Endeudamiento

5%

Fecha Renovación

07-05-16

Crear Cuenta

- También se puede utilizar `commandName` en lugar de `modelAttribute`, como se muestra a continuación.

```
<%@page contentType="text/html; charset=UTF-8"%>
<%@page pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<html>
<head>
  <meta content="text/html; charset=UTF-8" http-equiv="content-type" />
  <title>Crear Cuenta</title>
</head>
<body>
<div class="container">
  <h3>
    Crear Cuenta
  </h3>
  <div class="span-12 last">
    <!-- El objeto del model "cuenta" se crea dentro del metodo "crearCuentaForm(Model model)"
    de la clase controladora "CuentaController". -->
    <form:form commandName="cuenta" action="cuenta" method="post">
      <fieldset>
        <legend>Cuenta</legend>
        <p>
          <form:label for="nombre" path="nombre" cssErrorClass="error">Nombre</form:label><br/>
          <form:input path="nombre" /> <form:errors path="nombre" />
        </p>
        <p>
          <form:label for="saldo" path="saldo" cssErrorClass="error">Saldo</form:label><br/>
          <form:input path="saldo" /> <form:errors path="saldo" />
        </p>
        <p>
          <form:label for="nivelEndeudamiento" path="nivelEndeudamiento" cssErrorClass="error">Nivel
Endeudamiento</form:label><br/>
          <form:input path="nivelEndeudamiento" /> <form:errors path="nivelEndeudamiento" />
        </p>
        <p>
          <form:label for="fechaRenovacion" path="fechaRenovacion" cssErrorClass="error">Fecha
Renovación</form:label><br/>
          <form:input path="fechaRenovacion" /> <form:errors path="fechaRenovacion" />
        </p>
        <p>
          <input type="submit" value="Crear Cuenta"/>
        </p>
      </fieldset>
    </form:form>
  </div>
</div>
</body>
</html>
```

17. Estudiemos el flujo de control en un formulario de Spring:

- Ingrese un **nombre** y clic en **Crear Cuenta**.

Archivo Editar Ver Historial Marcadores Herramientas Ayuda (H)

mvc_form_basico

localhost:8080/mvc_form_basico/ Buscar

Formulario Básico mvc_form_basico

Crear Cuenta

Nombre

Andrés Guzmán

Saldo

Ch\$5.500,00

Nivel Endeudamiento

5%

Fecha Renovación

07-05-16

Crear Cuenta

- Luego, los datos del formulario son enviados al servidor, los datos del form serán capturados y poblados dentro del objeto cuenta.
- El envío será hacia la URL /cuenta a través de una petición HTTP POST.

```
<%@page contentType="text/html; charset=UTF-8"%>
<%@page pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<html>
<head>
  <meta content="text/html; charset=UTF-8" http-equiv="content-type" />
  <title>Crear Cuenta</title>
</head>
<body>
<div class="container">
  <h3>
    Crear Cuenta
  </h3>
  <div class="span-12 last">
    <!-- El objeto del model "cuenta" se crea dentro del metodo "crearCuentaForm(Model model)"
    de la clase controladora "CuentaController". -->
    <form:form modelAttribute="cuenta" action="cuenta" method="post">
      <fieldset>
        <legend>Cuenta</legend>
        <p>
          <form:label for="nombre" path="nombre" cssErrorClass="error">Nombre</form:label><br/>
          <form:input path="nombre" /> <form:errors path="nombre" />
        </p>
        <p>
          <form:label for="saldo" path="saldo" cssErrorClass="error">Saldo</form:label><br/>
          <form:input path="saldo" /> <form:errors path="saldo" />
        </p>
        <p>
          <form:label for="nivelEndeudamiento" path="nivelEndeudamiento" cssErrorClass="error">Nivel
          Endeudamiento</form:label><br/>
          <form:input path="nivelEndeudamiento" /> <form:errors path="nivelEndeudamiento" />
        </p>
        <p>
          <form:label for="fechaRenovacion" path="fechaRenovacion" cssErrorClass="error">Fecha
          Renovación</form:label><br/>
          <form:input path="fechaRenovacion" /> <form:errors path="fechaRenovacion" />
        </p>
        <p>
          <input type="submit" value="Crear Cuenta"/>
        </p>
      </fieldset>
    </form:form>
  </div>
</div>
</body>
</html>
```


- El envío será recibido y controlado por el método handler crearCuenta (Cuenta cuenta, BindingResult result).
- El objeto cuenta del formulario captura los valores ingresados por el usuario. Y automáticamente queda disponible como atributo del objeto model y es recibido por el método handler
- Luego lo valida y lo agrega en la lista cuentas, y redirige hacia la página de detalle pasando su id generado: [/cuenta/1](#).

```
package com.bolsadeideas.ejemplos.cuenta.controllers;

import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

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

import com.bolsadeideas.ejemplos.cuenta.domain.Cuenta;

@Controller
@RequestMapping(value="/cuenta")
public class CuentaController {

    private Map<Long, Cuenta> cuentas = new ConcurrentHashMap<Long, Cuenta>();

    // Metodo handler formulario, para crear la cuenta
    @RequestMapping(method=RequestMethod.GET)
    public String crearCuentaForm(Model model) {
        // La instancia de "Cuenta" es creada y guardada como atributo "cuenta"
        // en el objeto "model", el cual es accesible desde
        // la vista, "cuenta/crearForm.jsp".
        model.addAttribute("cuenta", new Cuenta());
        return "cuenta/crearForm";
    }

    // Metodo handler que procesa el envio de datos del form
    @RequestMapping(method=RequestMethod.POST)
    public String crearCuenta(Cuenta cuenta, BindingResult result) {
        // Si ocurre un error en la validación aparecera el formulario
        // "cuenta/crearForm.jsp" con los mensajes.
        if (result.hasErrors()) {
            return "cuenta/crearForm";
        }
    }
}
```

```
// Si todo está bien, crea la cuenta y la agrega a la lista
// "cuentas", luego redirige hacia el detalle /cuenta/{id},
// controlado por el método handler "verDetalle(..) de mas abajo.
this.cuentas.put(cuenta.asignarId(), cuenta);
return "redirect:/cuenta/" + cuenta.getId();
}

@RequestMapping(value="{id}", method=RequestMethod.GET)
public String verDetalle(@PathVariable Long id, Model model) {
    Cuenta cuenta = this.cuentas.get(id);
    if (cuenta == null) {
        throw new RecursoNoEncontradoException(id);
    }

    model.addAttribute("cuenta", cuenta);
    // return "cuenta/detalle"; // Muestra el detalle de la cuenta en formato formulario
    return "cuenta/detalle2"; // Muestra el detalle de la cuenta en formato tabla html
}
}
```

- La vista - /views/cuenta/detalle2.jsp es seleccionada por el view resolver y desplegada como respuesta en el navegador. El objeto del model "cuenta" es accesible desde la vista detalle.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<html>
<head>
<title>Mi Cuenta</title>
<style>
.error {
    color: #ff0000;
    font-weight: bold;
}
</style>
</head>

<body>

<table border="1">
    <tr>
        <th>Id</th>
        <th>Nombre</th>
        <th>Saldo</th>
        <th>Endeudamiento</th>
        <th></th>
    </tr>
    <tr>
        <td>${cuenta.id}</td>
        <td>${cuenta.nombre}</td>
        <td>${cuenta.saldo}</td>
        <td>${cuenta.nivelEndeudamiento}</td>
    </tr>
</table>
<br/>
<a href="..">Inicio</a>
</body>
</html>
```

Ejercicio 2: Generar y ejecutar el ejemplo "mvc_form_validador"

En este ejercicio, aprenderemos lo siguiente:

- ✓ Cómo crear y configurar un validador con WebDataBinder
 - ✓ Cómo usar la anotación @Value.
1. Clic derecho sobre el proyecto y **"Run As->Maven Clean"**.
 2. Clic derecho sobre el proyecto y **"Run As->Maven Install"**.
 3. Clic derecho **Maven->Update Project...**
 4. Clic derecho sobre el proyecto **mvc_form_validador-> Run As on Server**
 5. Observe el resultado.

The screenshot shows a web browser window with the address bar displaying 'localhost:8080/mvc_form_Validador/'. The page title is 'Ejemplo InitBinder y Validación usando clase CuentaValidator'. Below the title, there is a subtitle 'Crear Cuenta - Probar el validador: ingrese una fecha de renovación en tiempo pasado (ejemplo 03/04/15) para forzar el error.'.

The main content area contains a form titled 'Crear Cuenta' with the following fields:

- Nombre**: A text input field.
- Email**: A text input field.
- Saldo**: A text input field containing the value 'Ch\$5.000,00'.
- Nivel Endeudamiento**: A text input field containing the value '5%'.
- Fecha Renovación**: A text input field containing the value '08-05-16'.

At the bottom of the form is a blue button labeled 'Crear Cuenta'.

6. Ingresar una fecha en tiempo pasado - ejemplo **03-07-14** o bien **03-04-15**

validador: ingrese una fecha de renovación en tiempo pasado (ejemplo 03/04/15) para forzar el error.

Crear Cuenta

Nombre

Email

Saldo

Ch\$5.000,00

Nivel Endeudamiento

5%

Fecha Renovación

03-07-14

Crear Cuenta

7. Observe que se despliegan los mensajes de la validación en nombre y fecha de renovación.

Crear Cuenta

Nombre

El campo nombre es requerido!

Email

El formato del campo email es inválido!

Saldo

Ch\$5.000,00

Nivel Endeudamiento

5%

Fecha Renovación

03-07-14

la fecha de renovación tiene que ser del futuro!

Crear Cuenta

8. Abrir y estudiar la clase controladora `/src/main/java/``com.bolsadeideas.ejemplos.cuenta.controllers/CuentaController.java`

```
package com.bolsadeideas.ejemplos.cuenta.controllers;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

import javax.validation.Valid;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.InitBinder;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.bolsadeideas.ejemplos.cuenta.domain.Cuenta;
import com.bolsadeideas.ejemplos.cuenta.validators.CuentaValidator;

@Controller
@RequestMapping(value = "/cuenta")
public class CuentaController {

    private Map<Long, Cuenta> cuentas = new ConcurrentHashMap<Long, Cuenta>();

    // Configuramos un validador "CuentaValidator" para el "WebDataBinder".
    // En el ejemplo, ejecutamos dos tipos diferente de validaciones,
    // una usando anotaciones en la clase "Cuenta" y la segunda
    // usando una clase validadora de spring con cierta lógica
    // la clase "CuentaValidator".
    @InitBinder
    protected void initBinder(WebDataBinder binder) {
        binder.setValidator(new CuentaValidator());
    }

    // Metodo handler formulario, para crear la cuenta
    @RequestMapping(method = RequestMethod.GET)
    public String crearCuentaForm(Model model) {
        // La instancia de "Cuenta" es creada y guardada como atributo "cuenta"
        // en el objeto "model", el cual es accesible desde
        // la vista, "cuenta/crearForm.jsp".
        model.addAttribute("cuenta", new Cuenta());
        return "cuenta/crearForm";
    }
}
```

```
// Metodo handler que procesa el envio de datos del form
@RequestMapping(method = RequestMethod.POST)
public String crearCuenta(@Valid Cuenta cuenta, BindingResult result) {
    // Si ocurre un error en la validación aparecera el formulario
    // "cuenta/crearForm.jsp" con los mensajes.
    if (result.hasErrors()) {
        return "cuenta/crearForm";
    }

    // Si todo está bien, crea la cuenta y la agrega a la lista
    // "cuentas", luego redirige hacia el detalle /cuenta/{id},
    // controlado por el método handler "verDetalle(..) de mas abajo.
    this.cuentas.put(cuenta.asignarId(), cuenta);
    return "redirect:/cuenta/" + cuenta.getId();
}

@RequestMapping(value = "{id}", method = RequestMethod.GET)
public String verDetalle(@PathVariable Long id, Model model) {
    Cuenta cuenta = this.cuentas.get(id);
    if (cuenta == null) {
        throw new RecursoNoEncontradoException(id);
    }
    model.addAttribute("cuenta", cuenta);
    // return "cuenta/detalle"; // Muestra el detalle de la cuenta en
    // formato formulario
    return "cuenta/detalle2"; // Muestra el detalle de la cuenta en formato
        // tabla html
    }
}
```

- **Muy importante notar** que registramos la clase validadora **CuentaValidator** en el método anotado con **@InitBinder**, además el método handler **crearCuenta(...)** que recibe y procesa el objeto comando debe usar la anotación **@Valid** para llevar a cabo la validación.

9. Abrir y estudiar la clase CuentaValidator `/src/main/java/com.bolsadeideas.ejemplos.cuenta.validators/CuentaValidator.java`

```
package com.bolsadeideas.ejemplos.cuenta.validators;

import java.util.Date;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import org.springframework.validation.Errors;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Validator;

import com.bolsadeideas.ejemplos.cuenta.domain.Cuenta;

//La interfaz "validator" de spring está totalmente des-acoplada de cualquier capa o
//contexto, es decir ningún objeto esta acoplada a la validación, me refiero a ninguno
//ni los objetos de la capa web, ni de acceso a datos o models, o del nivel que sea.
//Como tal, es susceptible de ser utilizado en cualquier capa de la aplicación
public class CuentaValidator implements Validator{

    private static final String EMAIL_PATTERN = "^[_A-Za-z0-9-\\+]+(\\.[_A-Za-z0-9-\\+])*@"
        + "[A-Za-z0-9-\\+](\\.[A-Za-z0-9-\\+])*(\\.[A-Za-z]{2,})$";

    public boolean supports(Class<?> clazz) {
        //validamos que la instancia sea del tipo Cuenta
        return Cuenta.class.isAssignableFrom(clazz);
    }

    // Sobrescribimos el método para implementar nuestra lógica de validación.
    public void validate(Object target, Errors errors) {
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "nombre",
            "required.nombre", "nombre es requerido");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "saldo",
            "required.saldo", "saldo es requerido");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "nivelEndeudamiento",
            "required.nivelEndeudamiento", "nivelEndeudamiento es requerido");

        // Si la fecha Renovación es anterior a la actual,
        // Entonces resulta un error.
        Cuenta account = (Cuenta)target;
        if(account.getFechaRenovacion().before(new Date())){
            errors.rejectValue("fechaRenovacion", // nombre del campo
                "required.fechaRenovacion"); // codigo de error
        }

        if(!isEmail(account.getEmail())) {
            errors.rejectValue("email", "required.email", "email es invalido");
        }
    }
}
```

```
public static Boolean isEmail(String emailString) {  
    Pattern p = Pattern.compile(EMAIL_PATTERN);  
  
    // Match the given string with the pattern  
    Matcher m = p.matcher(emailString);  
  
    if (!m.matches()) {  
        return false;  
    }  
    return true;  
}
```

Existen dos formas de validar, incluso se pueden mezclar ambas y hacer un mix, una forma es implementando una clase validadora (tal cómo vimos) que implementa la interfaz Validator y la segunda es usando anotaciones en la clase del formulario o comando (la veremos luego en los siguientes ejemplos). Para la primera forma (implementada con una clase y la interfaz Validator) debemos registrarla en el controlador, tal como lo vimos en el método anotado con **@InitBinder**:

```
// Configuramos un validador "CuentaValidator" para el "WebDataBinder".  
// En el ejemplo, ejecutamos dos tipos diferente de validaciones,  
// una usando anotaciones en la clase "Cuenta" y la segunda  
// usando una clase validadora de spring con cierta lógica  
// la clase "CuentaValidator".  
@InitBinder  
protected void initBinder(WebDataBinder binder) {  
    binder.setValidator(new CuentaValidator());  
}  
  
/*ETC...*/
```

Luego con la anotación **@Valid** en el método que recibe y procesa el objeto formulario (objeto comando), le indicamos que debe validar la clase registrada en el controlador, finalmente usamos `if (result.hasErrors())`:

```
/*ETC...*/
// Metodo handler que procesa el envio de datos del form
@RequestMapping(method = RequestMethod.POST)
public String crearCuenta(@Valid Cuenta cuenta, BindingResult result) {
    // Si ocurre un error en la validación aparecera el formulario
    // "cuenta/crearForm.jsp" con los mensajes.
    if (result.hasErrors()) {
        return "cuenta/crearForm";
    }

    // Si todo está bien, crea la cuenta y la agrega a la lista
    // "cuentas", luego redirige hacia el detalle /cuenta/{id},
    // controlado por el método handler "verDetalle(..) de mas abajo.
    this.cuentas.put(cuenta.asignarId(), cuenta);
    return "redirect:/cuenta/" + cuenta.getId();
}
```

Como veremos más adelante, también usamos la anotación **@Valid** en el método handler para validar con anotaciones (especificación JSR 303: Bean Validation de la Plataforma Java).

Finalmente tenemos la clase de comando Cuenta, con sus atributos y métodos getter/setters:

```
package com.bolsadeideas.ejemplos.cuenta.domain;

import java.math.BigDecimal;
import java.util.Date;
import java.util.concurrent.atomic.AtomicLong;

import org.springframework.format.annotation.DateTimeFormat;
import org.springframework.format.annotation.NumberFormat;
import org.springframework.format.annotation.NumberFormat.Style;

public class Cuenta {

    private Long id;

    private String nombre;

    @NumberFormat(style = Style.CURRENCY)
    private BigDecimal saldo = new BigDecimal("5000");

    @NumberFormat(style = Style.PERCENT)
    private BigDecimal nivelEndeudamiento = new BigDecimal(".05");
}
```

```
@DateTimeFormat(style = "S-")
private Date fechaRenovacion = new Date(new Date().getTime() + 21425000000L);

private String email;

public Long getId() {
    return id;
}

void setId(Long id) {
    this.id = id;
}

public String getNombre() {
    return nombre;
}

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

public BigDecimal getSaldo() {
    return saldo;
}

public void setSaldo(BigDecimal saldo) {
    this.saldo = saldo;
}

public BigDecimal getNivelEndeudamiento() {
    return nivelEndeudamiento;
}

public void setNivelEndeudamiento(BigDecimal nivelEndeudamiento) {
    this.nivelEndeudamiento = nivelEndeudamiento;
}

public Date getFechaRenovacion() {
    return fechaRenovacion;
}

public void setFechaRenovacion(Date fechaRenovacion) {
    this.fechaRenovacion = fechaRenovacion;
}

ETC...
}
```

Ejercicio 3: Generar y ejecutar el ejemplo "mvc_form_validador2"

En este ejercicio, aprenderemos a crear y configurar dos clases validadoras para usarla en conjunto, un mix, una registrada con WebDataBinder (@InitBinder) y la segunda clase Inyectándola directamente en el controlador usando @Autowired.

1. Clic derecho sobre el proyecto y "Run As"->"Maven Clean".
2. Clic derecho sobre el proyecto y "Run As"->"Maven Install".
3. Clic derecho **Maven->Update Project...**
4. Clic derecho sobre el proyecto **mvc_form_validador2-> Run As on Server**
5. Observe el resultado.

Ejemplo Validación usando dos clases CuentaValidator y CuentaValidator2

Crear Cuenta - Probar el validador: ingrese una fecha de renovación en tiempo pasado (ejemplo 03/04/15) para forzar el error. Agregamos un segundo validador que revisa que el saldo sea mayor a 2000.

Crear Cuenta

Nombre

Email

Saldo

Nivel Endeudamiento

Fecha Renovación

Crear Cuenta

6. Ingresar una fecha incorrecta, ejemplo **03-07-12** y además un saldo menor de 2000, ej.: **1000**.

Crear Cuenta - Probar el validador: ingrese una fecha de renovación en tiempo pasado (ejemplo 03/04/15) para forzar el error. Agregamos un segundo validador que revisa que el saldo sea mayor a 2000.

Crear Cuenta

Nombre

Email

Saldo

Nivel Endeudamiento

Fecha Renovación

7. Observe que se despliegan los mensajes de la validación en nombre, saldo y fecha de renovación.

Crear Cuenta - Probar el validador: ingrese una fecha de renovación en tiempo pasado (ejemplo 03/04/15) para forzar el error. Agregamos un segundo validador que revisa que el saldo sea mayor a 2000.

Crear Cuenta

Nombre

El campo nombre es requerido!

Email

El formato del campo email es inválido!

Saldo

Ch\$1.000.00

El campo saldo está muy bajo!

Nivel Endeudamiento

5%

Fecha Renovación

03-07-12

la fecha de renovación tiene que ser del futuro!

Crear Cuenta

8. Abrir y estudiar la clase controladora `/src/main/java/com.bolsadeideas.ejemplos.cuenta.controllers/CuentaController.java`

```
package com.bolsadeideas.ejemplos.cuenta.controllers;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.InitBinder;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.bolsadeideas.ejemplos.cuenta.domain.Cuenta;
import com.bolsadeideas.ejemplos.cuenta.validators.CuentaValidator;
import com.bolsadeideas.ejemplos.cuenta.validators.CuentaValidator2;

@Controller
@RequestMapping(value="/cuenta")
public class CuentaController {

    private Map<Long, Cuenta> cuentas = new ConcurrentHashMap<Long, Cuenta>();

    // Configuramos un validador "CuentaValidator" para el "WebDataBinder".
    // En el ejemplo, ejecutamos dos tipos diferente de validación,
    // una usando anotaciones en la clase "Cuenta",
    // la segunda usando una clase validadora se spring con cierta lógica en
    // la clase "CuentaValidator".
    @InitBinder
    protected void initBinder(WebDataBinder binder) {
        binder.setValidator(new CuentaValidator());
    }

    // Usamos otro validador CuentaValidator2
    @Autowired
    CuentaValidator2 cuentaValidator2;

    // Metodo handler formulario, para crear la cuenta
    @RequestMapping(method = RequestMethod.GET)
    public String crearCuentaForm(Model model) {
        // La instancia de "Cuenta" es creada y guardada como atributo "cuenta"
        // en el objeto "model", el cual es accesible desde
        // la vista, "cuenta/crearForm.jsp".
        model.addAttribute("cuenta", new Cuenta());
        return "cuenta/crearForm";
    }
}
```



```

// Metodo handler que procesa el envio de datos del form
@RequestMapping(method = RequestMethod.POST)
public String crearCuenta(@Valid Cuenta cuenta, BindingResult result) {

    // Ejecutamos la validación directamente dentro del metodo
    cuentaValidator2.validate(cuenta, result);

    // Si ocurre un error en la validación aparecera el formulario
    // "cuenta/crearForm.jsp" con los mensajes.
    if (result.hasErrors()) {
        return "cuenta/crearForm";
    }

    // Si todo está bien, crea la cuenta y la agrega a la lista
    // "cuentas", luego redirige hacia el detalle /cuenta/{id},
    // controlado por el método handler "verDetalle(..)" de más abajo.
    this.cuentas.put(cuenta.asignarId(), cuenta);
    return "redirect:/cuenta/" + cuenta.getId();
}

@RequestMapping(value = "{id}", method = RequestMethod.GET)
public String verDetalle(@PathVariable Long id, Model model) {
    Cuenta cuenta = this.cuentas.get(id);
    if (cuenta == null) {
        throw new RecursoNoEncontradoException(id);
    }
    model.addAttribute("cuenta", cuenta);
    // return "cuenta/detalle"; // Muestra el detalle de la cuenta en
    // formato formulario
    return "cuenta/detalle2"; // Muestra el detalle de la cuenta en formato
        // tabla html
}
}

```

En este ejemplo funcionamos con dos validadores de forma mixta, implementando dos clases con la interfaz Validator, una se registra en el `initBinder(...)` y la segunda se inyecta con `@Autowired` (**CuentaValidator2**) y se valida de forma manual dentro del método `crearCuenta(...)`:

cuentaValidator2.validate(cuenta, result);

9. Abrir y estudiar la clase CuentaValidator2 `/src/main/java/com.bolsadeideas.ejemplos.cuenta.validators/CuentaValidator2.java`

```
package com.bolsadeideas.ejemplos.cuenta.validators;

import java.math.BigDecimal;
import java.util.Date;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;
import com.bolsadeideas.ejemplos.cuenta.domain.Cuenta;

//La interfaz "validator" de spring está totalmente des-acoplada de cualquier capa o
//contexto, es decir ningún objeto esta acoplada a la validación, me refiero a ninguno
//ni los objetos de la capa web, ni de acceso a datos o models, o del nivel que sea.
//Como tal, es susceptible de ser utilizado en cualquier capa de la aplicación
public class CuentaValidator2 implements Validator {

    private static final String EMAIL_PATTERN = "^[_A-Za-z0-9-\\+]+(\\.[_A-Za-z0-9-]+)*@"
        + "[A-Za-z0-9-]+(\\.[A-Za-z0-9-]+)*(\\.[A-Za-z]{2,})$";

    public boolean supports(Class<?> clazz) {
        // validamos que la instancia sea del tipo Cuenta
        return Cuenta.class.isAssignableFrom(clazz);
    }

    // Sobre-escribimos el método para implementar nuestra lógica de validación.
    public void validate(Object target, Errors errors) {
        Cuenta account = (Cuenta) target;

        // Si el saldo es menor a 2000,
        // entonces resulta un error por ser muy bajo.
        if (account.getSaldo() != null &&
            account.getSaldo().compareTo(new BigDecimal(2000)) < 0) {
            errors.rejectValue("saldo", // nombre del campo
                               "required.saldoBajo"); // codigo de error
        }

        // Si la fecha Renovacion es anterior a la actual, entonces resulta un error.
        if (account.getFechaRenovacion() != null &&
            account.getFechaRenovacion().before(new Date())) {
            errors.rejectValue("fechaRenovacion", // nombre del campo
                               "required.fechaRenovacion"); // codigo de error
        }

        if (!isEmail(account.getEmail())) {
            errors.rejectValue("email", "required.email", "email es invalido");
        }
    }
}
```

```
public static Boolean isEmail(String emailString) {
    Pattern p = Pattern.compile(EMAIL_PATTERN);
    // Match the given string with the pattern
    Matcher m = p.matcher(emailString);

    if (!m.matches()) {
        return false;
    }
    return true;
}
}
```

10. Abrir y estudiar el archivo de configuración de Spring - root-context.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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- Root Context: defines shared resources visible to all other web components -->
    <!-- Mensajes de idiomas -->
    <bean id="messageSource"
        class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
        <property name="basename" value="/WEB-INF/mensajes/mensajes" />
        <property name="cacheSeconds" value="0" />
    </bean>

    <!-- Validador bean -->
    <bean class="com.bolsadeideas.ejemplos.cuenta.validators.CuentaValidator2" />
</beans>
```

Ejercicio 4: Generar y ejecutar el ejemplo "mvc_form_InitBinder"

En este ejercicio, aprenderemos sobre los Binders para conversión de valores del formulario a objetos y validación con anotaciones (JSR 303: Bean Validation de la Plataforma Java).

Spring proporciona soporte para la validación de formularios mediante la especificación JSR 303: Bean Validation de la Plataforma Java. La especificación JSR-303 es una instancia de `javax.validation.Validator`, se encarga de validar todos los objetos del modelo que declaran restricciones mediante uso de anotaciones.

El componente `javax.validation.Validator` (JSR-303) se encarga de validar a todos los objetos comando que tienen definidas las reglas de validación vía anotaciones JSR-303

Cualquier violación de las reglas de validación automáticamente gatillará en errores en el `BindingResult` mostrando los mensajes de error mediante la etiqueta `form "errors"` (Spring MVC form taglib):

```
<form:errors path="nombre" cssClass="error"/>
```

Los Binders convierten los datos (string) ingresados en el formato que queramos. En el ejemplo, la fecha de renovación es ingresada en formato yyyy-MM-dd y automáticamente es convertida en un tipo `Date` de java. También permiten convertir a diferentes tipos de objeto e incluso para formatear datos, por ejemplo convertir el nombre ingresado en mayúscula.

1. Clic derecho sobre el proyecto **"Run As->Maven Clean"** y **"Run As->Maven Install"**.
2. Clic derecho **Maven->Update Project...**
3. Clic derecho sobre el proyecto **mvc_form_InitBinder-> Run As on Server**

Archivo Editar Ver Historial Marcadores Herramientas Ayuda (H)

mvc_form_InitBinder

localhost:8080/mvc_form_InitBinder/ Buscar

Ejemplo InitBinder y Validación

Anotaciones

ingrese una fecha renovación en el formato yyyy-MM-dd, también ingrese el nombre en minúscula el cual posteriormente será convertido en mayúscula

Crear Cuenta

Nombre

Email

Saldo

Nivel Endeudamiento

Fecha Renovación

Crear Cuenta

4. Clic en Crear Cuenta para la validación

Crear Cuenta

Nombre

no puede estar vacío
el tamaño tiene que estar entre 1 y 25

Email

El formato del email es inválido

Saldo

no puede ser null

Nivel Endeudamiento

Fecha Renovación

Crear Cuenta

5. Ingresamos datos

Crear Cuenta

Nombre

no puede estar vacío
el tamaño tiene que estar entre 1 y 25

Email

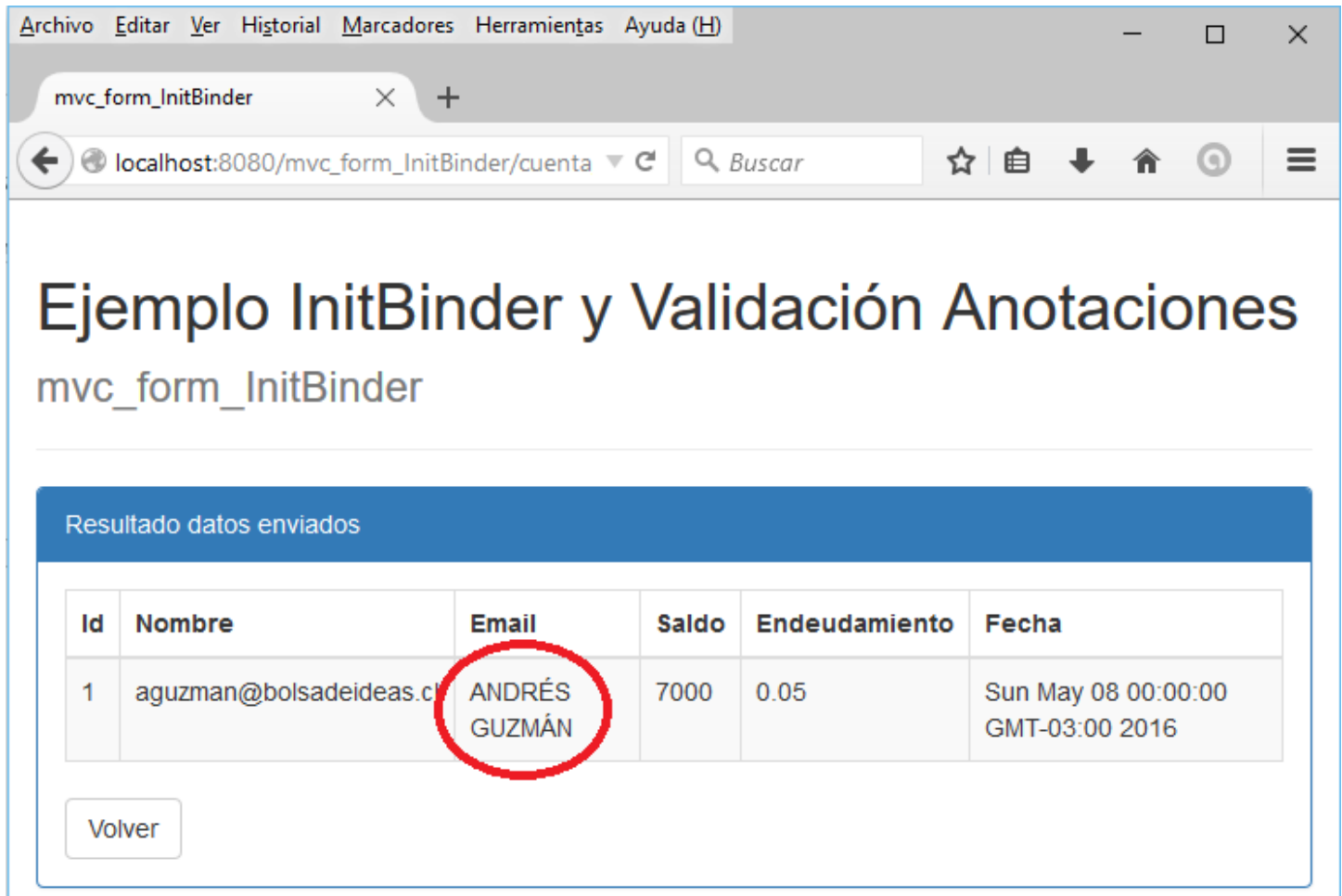
El formato del email es inválido

Saldo

no puede ser null

Nivel Endeudamiento

Fecha Renovación



Archivo Editar Ver Historial Marcadores Herramientas Ayuda (H)

localhost:8080/mvc_form_InitBinder/cuenta

Ejemplo InitBinder y Validación Anotaciones

mvc_form_InitBinder

Resultado datos enviados

Id	Nombre	Email	Saldo	Endeudamiento	Fecha
1	aguzman@bolsadeideas.c	ANDRÉS GUZMÁN	7000	0.05	Sun May 08 00:00:00 GMT-03:00 2016

Volver

6. Observamos que el nombre es convertido a mayúscula.

7. Abrir y estudiar la clase controladora `/src/main/java/``com.bolsadeideas.ejemplos.cuenta.controllers/CuentaController.java`

```
package com.bolsadeideas.ejemplos;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import javax.validation.Valid;
import org.springframework.beans.propertyeditors.CustomDateEditor;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.InitBinder;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.bolsadeideas.ejemplos.cuenta.binders.NombreMayusculaEditor;
import com.bolsadeideas.ejemplos.cuenta.domain.Cuenta;
import com.bolsadeideas.ejemplos.cuenta.validators.CuentaValidator;

@Controller
@RequestMapping(value = "/cuenta")
public class CuentaController {

    private Map<Long, Cuenta> cuentas = new ConcurrentHashMap<Long, Cuenta>();

    // La anotación "@InitBinder" identifica el método que
    // va a iniciar el "WebDataBinder", quien se encarga de registrar
    // validadores, editores y de poblar el objeto comando con los datos
    // del formulario. Los editores se encargan de convertir los datos del
    // formulario a cierto tipo o formato antes de que se pasen o pueblen
    // al objeto comando. Luego el comando será pasado por
    // argumento en los métodos anotados (handler) del controlador.
    @InitBinder
    protected void initBinder(WebDataBinder binder) {
        binder.setValidator(new CuentaValidator());

        // Registramos los editores. "CustomDateEditor" y "NombreMayusculaEditor".
        // "CustomDateEditor" es propio de Spring framework para validar la
        // fecha a un formato específico y convertir a Date de java, mientras
        // que el "NombreMayusculaEditor" es escrito por nosotros para convertir
        // el nombre ingresado a mayúscula.
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
        dateFormat.setLenient(false);
        binder.registerCustomEditor(Date.class, new CustomDateEditor(dateFormat, false));
        binder.registerCustomEditor(String.class, new NombreMayusculaEditor());
    }
}
```

```
// Metodo handler formulario, para crear la cuenta
@RequestMapping(method = RequestMethod.GET)
public String crearCuentaForm(Model model) {
    // La instancia de "Cuenta" es creada y guardada como atributo "cuenta"
    // en el objeto "model", el cual es accesible desde
    // la vista, "cuenta/crearForm.jsp".
    model.addAttribute("cuenta", new Cuenta());
    return "cuenta/crearForm";
}

// Metodo handler que procesa el envio de datos del form
@RequestMapping(method = RequestMethod.POST)
public String crearCuenta(@Valid Cuenta cuenta, BindingResult result) {
    // Si ocurre un error en la validación aparecera el formulario
    // "cuenta/crearForm.jsp" con los mensajes.
    if (result.hasErrors()) {
        return "cuenta/crearForm";
    }

    // Si todo está bien, crea la cuenta y la agrega a la lista
    // "cuentas", luego redirige hacia el detalle /cuenta/{id},
    // controlado por el método handler "verDetalle(..) de mas abajo.
    this.cuentas.put(cuenta.asignarId(), cuenta);
    return "redirect:/cuenta/" + cuenta.getId();
}

@RequestMapping(value = "{id}", method = RequestMethod.GET)
public String verDetalle(@PathVariable Long id, Model model) {
    Cuenta cuenta = this.cuentas.get(id);
    if (cuenta == null) {
        throw new RecursoNoEncontradoException(id);
    }
    model.addAttribute("cuenta", cuenta);
    // return "cuenta/detalle"; // Muestra el detalle de la cuenta en
    // formato formulario
    return "cuenta/detalle2"; // Muestra el detalle de la cuenta en formato
        // tabla html
}
}
```

8. Abrir y estudiar la clase Binder `/src/main/java/``com.bolsadeideas.ejemplos.cuenta.binders/NombreMayusculaEditor.java`

```
package com.bolsadeideas.ejemplos.cuenta.binders;
import java.beans.PropertyEditorSupport;

public class NombreMayusculaEditor extends PropertyEditorSupport {

    @Override
    public void setAsText(String text) {
        if (text == null) {
            setValue(null);
        }
        else {
            String value = text.toUpperCase();
            setValue(value);
        }
    }
}
```

9. Finalmente tenemos las anotaciones para validar en el objeto comando Cuenta, estas anotaciones tienen que ir sobre los atributos que queremos validar, por ejemplo nombre:

```
package com.bolsadeideas.ejemplos.cuenta.domain;

import java.math.BigDecimal;
import java.util.Date;
import java.util.concurrent.atomic.AtomicLong;

import javax.validation.constraints.Future;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Pattern;
import javax.validation.constraints.Size;

import org.hibernate.validator.constraints.NotEmpty;
import org.springframework.format.annotation.DateTimeFormat;
import org.springframework.format.annotation.NumberFormat;
import org.springframework.format.annotation.NumberFormat.Style;

public class Cuenta {

    private static final String EMAIL_PATTERN = "^[_A-Za-z0-9-\\+](\\.[_A-Za-z0-9-]+)*@"
        + "[A-Za-z0-9-](\\.[A-Za-z0-9]+)*(\\.[A-Za-z]{2,})$";

    private Long id;

    @NotEmpty
    @Size(min = 1, max = 25)
    private String nombre;

    @NotNull
    @NumberFormat(style = Style.CURRENCY)
    private BigDecimal saldo = new BigDecimal("5000");

    @NotNull
    @NumberFormat(style = Style.PERCENT)
    private BigDecimal nivelEndeudamiento = new BigDecimal(".05");

    @DateTimeFormat(style = "S-")
    @Future
    private Date fechaRenovacion = new Date(new Date().getTime() + 21425000000L);

    // @Email
    @Pattern(regexp = EMAIL_PATTERN, message = "El formato del email es inválido")
    private String email;

    ... ETC ...

}
```

En el nombre va a validar que el rango de caracteres sea justamente entre 1-25. También podríamos tener otras anotaciones como por ejemplo para validar **@Email** que es un validador propio de Hibernate, automáticamente valida que el string tenga un formato de correo es decir que contenga caracteres alfanuméricos y un arroba y luego caracteres alfanuméricos seguido de un punto y el dominio.

Para validar que una cadena no sea vacía usamos la anotación **@NotEmpty**, es decir para que un campo del formulario sea requerido (siempre del tipo String), en los String no nos sirve **@NotNull** ya que éste último es sólo para validar que un objeto no sea null.

Un String en el formulario nunca va a ser null, si no se llena el campo queda como un String vacío, pero jamás null. Para otros tipos de datos y objetos como Integer, Decimal, Double, Date etc... usamos **@NotNull**:

```
@Email
private String email;

@NotEmpty
@Size(min = 5, max = 8)
private String username;

@NotNull
@NumberFormat(style = Style.CURRENCY)
private BigDecimal saldo = new BigDecimal("5000");
```

Otro punto a tener en cuenta para que funcionen las validaciones usando anotaciones, primero es tener el pom.xml con las dependencias maven y bien implementado las anotaciones:

ETC...

```
<!-- Bean Validation (JSR-303) -->
<dependency>
  <groupId>javax.validation</groupId>
  <artifactId>validation-api</artifactId>
  <version>1.1.0.Final</version>
</dependency>
<!-- Hibernate Bean Validation (JSR-303) -->
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-validator</artifactId>
  <version>5.2.1.Final</version>
</dependency>
```

ETC...

10. Luego con la anotación **@Valid** en el método que recibe y procesa el objeto formulario (objeto comando), le indicamos que debe validar las reglas definidas con anotaciones, finalmente usamos `if (result.hasErrors())`:

```
/*ETC...*/
// Metodo handler que procesa el envio de datos del form
@RequestMapping(method = RequestMethod.POST)
public String crearCuenta(@Valid Cuenta cuenta, BindingResult result) {
    // Si ocurre un error en la validación aparecera el formulario
    // "cuenta/crearForm.jsp" con los mensajes.
    if (result.hasErrors()) {
        return "cuenta/crearForm";
    }

    // Si todo está bien, crea la cuenta y la agrega a la lista
    // "cuentas", luego redirige hacia el detalle /cuenta/{id},
    // controlado por el método handler "verDetalle(..) de mas abajo.
    this.cuentas.put(cuenta.asignarId(), cuenta);
    return "redirect:/cuenta/" + cuenta.getId();
}
```

Ejercicio 5: Generar y ejecutar el ejemplo "mvc_form_etiquetas"

En este ejercicio, aprenderemos lo siguiente:

- ✓ Aprenderemos a usar varios tipos de form tags de spring
 - ✓ Cómo usar la anotación `@ModelAttribute` para crear atributos de vistas y poder usarlas para llenar los controles del formulario.
 - ✓ Reforzaremos la validación mediante anotaciones Bean Validation (JSR-303)
1. Clic derecho **"Run As->Maven Clean"** y **"Run As->Maven Install"**.
 2. Clic derecho **Maven->Update Project...**
 3. Clic derecho sobre el proyecto **mvc_form_etiquetas-> Run As on Server**
 4. Clic en botón **Crear Estudiante** sin ingresar ningún dato, la idea es probar la validación.

Formulario Spring Etiquetas Form de Spring y Validaciones usando Anotaciones

Crear Estudiante

User Name :

Dirección :

Password :

Suscribirse al newsletter? : ☐

Temas favoritos : ☒Matematicas ☐Ciencia ☐Arte ☐Musica ☐Deporte

Género : ☒Hombre ☐Mujer

Seleccione un número : ☐Numero 1 ☐Numero 2 ☐Numero 3 ☐Numero 4 ☐Numero 5 ☐Numero 6 ☐Numero 7

País :

Habilidades de Spring :

Spring Core

Spring MVC

Spring Roo

Hibernate

Crear Estudiante

5. Observamos la validación con los mensajes de error

Crear Estudiante

El password es requerido!
Seleccione un pais!
Seleccione un número favorito!
El campo nombre es requerido!

User Name :

El campo nombre es requerido!

Dirección :

Av. Keneddy

Password :

El password es requerido!

Suscribirse al newsletter? :

☐

Temas favoritos :

☒Matematicas☐Ciencia☐Arte☐Musica☐Deporte

Género :

☒Hombre ☐Mujer

Seleccione un número :

☐Numero 1☐Numero 2☐Numero 3☐Numero 4☐Numero 5☐Numero 6☐Numero 7

Seleccione un número favorito!

País :

--- Seleccione ---

Seleccione un pais!

Habilidades de Spring :

Spring Core
Spring MVC
Spring Roo
Hibernate

6. Ingresamos los datos y enviamos el formulario.

El password es requerido!

Seleccione un pais!

Seleccione un número favorito!

El campo nombre es requerido!

User Name :

andresguzf

El campo nombre es requerido!

Dirección :

Av. Keneddy

Password :

••••••

El password es requerido!

Suscribirse al
newsletter? :



Temas favoritos :

☒Matematicas ☐Ciencia ☐Arte ☐Musica ☐Deporte

Género :

☒Hombre ☐Mujer

Seleccione un
número :

☐Numero 1 ☐Numero 2 ☐Numero 3 ☐Numero 4 ☐Numero 5 ☐Numero 6 ☒Numero 7 Seleccione un número favorito

País :

Chile

Seleccione un pais!

Habilidades de
Spring :

Spring Core
Spring MVC
Spring Roo
Hibernate

Crear Estudiante

7. Observe que se despliegan los datos.

Formulario Spring Etiquetas Form de Spring y Validaciones usando Anotaciones

Resultado Estudiante

User Name:	andresguzf
Dirección:	Av. Keneddy
Password:	123456
Recibe newsletter:	true
Temas favoritos:	[Matematicas]
Género:	H
Número Favorito :	Numero 7
País:	CL
Habilidades de Spring:	Spring Core, Spring MVC, Spring Roo, Hibernate
Valor Secreto :	Algun valor oculto

Volver

8. Abrir y estudiar la clase controladora `/src/main/java/com.bolsadeideas.ejemplos.controllers/EstudianteController.java`

```
package com.bolsadeideas.ejemplos.controllers;

import java.sql.Date;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import javax.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.propertyeditors.CustomDateEditor;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.InitBinder;
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.bind.support.SessionStatus;

import com.bolsadeideas.ejemplos.domain.Estudiante;
import com.bolsadeideas.ejemplos.validator.EstudianteValidator;

@Controller
@RequestMapping("/estudiante")
public class EstudianteController {

    EstudianteValidator estudianteValidator;

    @Autowired
    public EstudianteController(EstudianteValidator estudianteValidator) {
        this.estudianteValidator = estudianteValidator;
    }

    // Metodo handler formulario, para crear al estudiante
    @RequestMapping(method = RequestMethod.GET)
    public String getCreateForm(ModelMap model) {

        Estudiante estudiante = new Estudiante();

        estudiante.setDireccion("Av. Keneddy");
        estudiante.setTemas(new String[] { "Matematicas" });
        estudiante.setGenero("H");
        estudiante.setExperienciaSpring("Spring MVC");
        estudiante.setValorSecreto("Algún valor oculto");
    }
}
```

```
// Guardamos al objeto comando como "estudianteCommand" el cual
// será accesible en la vista estudianteForm.
model.addAttribute("estudianteCommand", estudiante);
// retornamos la vista form
return "estudianteForm";
}

// Metodo handler que procesa el envio de datos del form
@RequestMapping(method = RequestMethod.POST)
public String processSubmit(@Valid
    @ModelAttribute("estudianteCommand") Estudiante estudiante,
    BindingResult result, SessionStatus status) {

    if (result.hasErrors()) {
        // Si ocurre un error en la validacion retornamos al formulario
        // con los mensajes de error.
        return "estudianteForm";
    } else {
        status.setComplete();
        // Si pasa bien la validacion, retornamos "estudianteOk"
        return "estudianteOk";
    }
}

// La anotación "@ModelAttribute" une o relaciona
// un parámetro de un método handler (@RequestMapping) o un valor
// de retorno en un método hacia un "nombrado" atributo del model.
// El model atributo "listaTemas" podra ser usado en la vista jsp
// "estudianteForm.jsp".
@ModelAttribute("listaTemas")
public List<String> llenarListaTemas() {

    // Data reference de temas para llenar checkboxes
    List<String> listaTemas = new ArrayList<String>();
    listaTemas.add("Matemáticas");
    listaTemas.add("Ciencia");
    listaTemas.add("Arte");
    listaTemas.add("Musica");
    listaTemas.add("Deporte");
    return listaTemas;
}

// La anotación "@InitBinder" identifica el método que va a iniciar
// el "WebDataBinder", para poblar el objeto comando con los datos
// del formulario. Luego el comando será pasado por argumento
// en los métodos anotados (handler) del controlador.
@InitBinder
public void initBinder(WebDataBinder binder) {
    SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
    binder.registerCustomEditor(Date.class, new CustomDateEditor(dateFormat, true));
}
```

```
@ModelAttribute("listaNumeros")
public List<String> llenarListaNumeros() {

    // Data reference de números para radiobuttons
    List<String> numeros = new ArrayList<String>();
    numeros.add("Numero 1");
    numeros.add("Numero 2");
    numeros.add("Numero 3");
    numeros.add("Numero 4");
    numeros.add("Numero 5");
    numeros.add("Numero 6");
    numeros.add("Numero 7");

    return numeros;
}

@ModelAttribute("experienciaSpring")
public Map<String, String> llenarListaExperienciaSpring() {

    // Data reference de experiencias spring para list box
    Map<String, String> experienciaSpring = new LinkedHashMap<String, String>();
    experienciaSpring.put("Spring Core", "Spring Core");
    experienciaSpring.put("Spring MVC", "Spring MVC");
    experienciaSpring.put("Spring Roo", "Spring Roo");
    experienciaSpring.put("Hibernate", "Hibernate");

    return experienciaSpring;
}

@ModelAttribute("listaPaises")
public Map<String, String> llenarPaises() {

    // Data reference de paises para list box
    Map<String, String> paises = new LinkedHashMap<String, String>();
    paises.put("CL", "Chile");
    paises.put("ES", "España");
    paises.put("MX", "México");
    paises.put("US", "United Stated");
    paises.put("AR", "Argentina");

    return paises;
}
}
```

9. Abrir y estudiar la Estudiante con las validaciones anotadas

/src/main/java/com.bolsadeideas.ejemplos.domain/Estudiante.java

```
package com.bolsadeideas.ejemplos.domain;

import javax.validation.constraints.Size;
import org.hibernate.validator.constraints.NotEmpty;

public class Estudiante{

    @Size(message="El campo nombre es requerido!", min = 4, max = 12)
    private String userName;

    @NotEmpty(message="El campo dirección es requerido!")
    private String direccion;

    @Size(message="El password es requerido!", min = 4, max = 6)
    private String password;

    private boolean recibeNewsletter;

    @NotEmpty(message="Seleccione temas!")
    private String[] temas;

    @NotEmpty(message="Seleccione un número favorito!")
    private String numeroFavorito;

    @NotEmpty(message="Seleccione un género!")
    private String genero;

    @NotEmpty(message="Seleccione un país!")
    private String pais;

    @NotEmpty(message="Seleccione habilidades de Spring!")
    private String experienciaSpring;

    //valor oculto
    private String valorSecreto;

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getDireccion() {
        return direccion;
    }
}
```

```
public void setDireccion(String direccion) {
    this.direccion = direccion;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

public boolean isRecibeNewsletter() {
    return recibeNewsletter;
}

public void setRecibeNewsletter(boolean recibeNewsletter) {
    this.recibeNewsletter = recibeNewsletter;
}

public String[] getTemas() {
    return temas;
}

public void setTemas(String[] temas) {
    this.temas = temas;
}

public String getNumeroFavorito() {
    return numeroFavorito;
}

public void setNumeroFavorito(String numeroFavorito) {
    this.numeroFavorito = numeroFavorito;
}

public String getGenero() {
    return genero;
}

public void setGenero(String genero) {
    this.genero = genero;
}

public String getPais() {
    return pais;
}

public void setPais(String pais) {
    this.pais = pais;
}
```

```

public String getExperienciaSpring() {
    return experienciaSpring;
}

public void setExperienciaSpring(String experienciaSpring) {
    this.experienciaSpring = experienciaSpring;
}

public String getValorSecreto() {
    return valorSecreto;
}

public void setValorSecreto(String valorSecreto) {
    this.valorSecreto = valorSecreto;
}
}

```

10. Abrir y estudiar la vista jsp estudianteForm.jsp

```

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>
<style>
.error {
    color: #ff0000;}
.errorblock {
    color: #000;
    background-color: #ffEEEE;
    border: 3px solid #ff0000;
    padding: 8px;
    margin: 16px;}
</style>
</head>
<body>
<h2>Ejemplo de etiquetas form de Spring - Formulario</h2>

<form:form modelAttribute="estudianteCommand" method="POST" >

    <form:errors path="*" cssClass="errorblock" element="div" />

    <table>
        <tr>
            <td>User name:</td>
            <td><form:input path="userName" /></td>
            <td><form:errors path="userName" cssClass="error" /></td>
        </tr>
        <tr>
            <td>Dirección :</td>
            <td><form:textarea path="direccion" /></td>
            <td><form:errors path="direccion" cssClass="error" /></td>

```



```

</tr>
<tr>
  <td>Password :</td>
  <td><form:password path="password" /></td>
  <td><form:errors path="password" cssClass="error" /></td>
</tr>
<tr>
  <td>Suscribirse al newsletter? :</td>
  <td><form:checkbox path="recibeNewsletter" /></td>
  <td><form:errors path="recibeNewsletter" cssClass="error" /></td>
</tr>
<tr>
  <td>Temas favoritos:</td>
  <td><form:checkboxes items="${listaTemas}" path="temas" /></td>
  <td><form:errors path="temas" cssClass="error" /></td>
</tr>
<tr>
  <td>Género :</td>
  <td><form:radio button path="genero" value="H" />Hombre <form:radio button
    path="genero" value="F" />Mujer</td>
  <td><form:errors path="genero" cssClass="error" /></td>
</tr>
<tr>
  <td>Seleccione un número:</td>
  <td><form:radio buttons path="numeroFavorito" items="${listaNumeros}" />
  </td>
  <td><form:errors path="numeroFavorito" cssClass="error" /></td>
</tr>
<tr>
  <td>País:</td>
  <td><form:select path="pais">
    <form:option value="NONE" label="--- Seleccione ---" />
    <form:options items="${listaPaises}" />
  </form:select></td>
  <td><form:errors path="pais" cssClass="error" /></td>
</tr>
<tr>
  <td>Habilidades de Spring:</td>
  <td><form:select path="experienciaSpring" items="${experienciaSpring}"
    multiple="true" /></td>
  <td><form:errors path="experienciaSpring" cssClass="error" /></td>
</tr>
<form:hidden path="valorSecreto" />
<tr>
  <td colspan="3"><input type="submit" value="Enviar" /></td>
</tr>
</table>
</form:form>
</body>
</html>

```

11. Abrir y estudiar la vista jsp estudianteOk.jsp.jsp

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<html>
<body>
  <h2>Ejemplo de etiquetas form de Spring - resultado</h2>
  <table>
    <tr>
      <td>User Name:</td>
      <td>${estudianteCommand.userName}</td>
    </tr>
    <tr>
      <td>Dirección:</td>
      <td>${estudianteCommand.direccion}</td>
    </tr>
    <tr>
      <td>Password:</td>
      <td>${estudianteCommand.password}</td>
    </tr>
    <tr>
      <td>Recibe newsletter:</td>
      <td>${estudianteCommand.recibeNewsletter}</td>
    </tr>
    <tr>
      <td>Temas favoritos:</td>
      <td><c:forEach items="${estudianteCommand.temas}" var="tema">
        [<c:out value="${tema}" />]
      </c:forEach></td>
    </tr>
    <tr>
      <td>Género:</td>
      <td>${estudianteCommand.genero}</td>
    </tr>
    <tr>
      <td>Número Favorito :</td>
      <td>${estudianteCommand.numeroFavorito}</td>
    </tr>
    <tr>
      <td>País:</td>
      <td>${estudianteCommand.pais}</td>
    </tr>
    <tr>
      <td>Habilidades de Spring:</td>
      <td>${estudianteCommand.experienciaSpring}</td>
    </tr>
    <tr>
      <td>Valor Secreto:</td>
      <td>${estudianteCommand.valorSecreto}</td>
    </tr>
  </table></body></html>
```

Ejercicio 6: Generar y ejecutar el ejemplo "ModelAttributeComoParametro"

En este ejercicio, aprenderemos lo siguiente:

- ✓ Cómo usar la anotación `@ModelAttribute` a nivel de método y argumento pasado al método handler.
- 1. Clic derecho sobre el proyecto "Run As->Maven Clean" y "Run As->Maven Install".
- 2. Clic derecho **Maven->Update Project...**
- 3. Clic derecho en `mvc_form_basico_ModelAttributeComoParametro-> Run As on Server`
- 4. Observe el resultado.

Archivo Editar Ver Historial Marcadores Herramientas Ayuda (H)

mvc_form_basico_ModelAttri... X +

localhost:8080/mvc_form_basico_ModelAttributeComoPa Buscar

Formulario Básico @ModelAttribute cómo parametro/atributode de la vista!

Crear Cuenta

Nombre

Andrés Guzmán

Saldo

Ch\$5.000,00

Nivel Endeudamiento

5%

Fecha Renovación

08-05-16

Crear Cuenta

Archivo Editar Ver Historial Marcadores Herramientas Ayuda (H)

mvc_form_basico_ModelAttri... × +

localhost:8080/mvc_form_basico_ModelAttributeComoPa Buscar

Formulario Básico

mvc_form_basico_ModelAttributeComoParametro

Resultado datos enviados

Id	Nombre	Saldo	Endeudamiento
1	Andrés Guzmán	5000.00	0.05

Productos

Cuenta Corriente

Tarjeta de credito VISA

Tarjeta de credito MASTERCARD

Fondos Mutuos

Credito Hipotecario

Volver

5. Abrir y estudiar la clase controladora `/src/main/java/com.bolsadeideas.ejemplos.cuenta.controllers/CuentaController.java`

```
package com.bolsadeideas.ejemplos.cuenta.controllers;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
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.SessionAttributes;

import com.bolsadeideas.ejemplos.cuenta.domain.Cuenta;

@Controller
@RequestMapping(value = "/cuenta")
@SessionAttributes("cuenta1")
public class CuentaController {

    private Map<Long, Cuenta> cuentas = new ConcurrentHashMap<Long, Cuenta>();

    // Metodo handler formulario, para crear la cuenta
    @RequestMapping(method=RequestMethod.GET)
    public String crearCuentaForm(Model model) {
        // La instancia de "Cuenta" es creada y guardada como atributo "cuenta"
        // en el objeto "model", el cual es accesible desde
        // la vista, "cuenta/crearForm.jsp".
        model.addAttribute("cuenta", new Cuenta());
        return "cuenta/crearForm";
    }

    // Metodo handler que procesa el envio de datos del form
    @RequestMapping(method=RequestMethod.POST)
    public String crearCuenta(Cuenta cuenta, BindingResult result) {
        // Si ocurre un error en la validación aparecera el formulario
        // "cuenta/crearForm.jsp" con los mensajes.
        if (result.hasErrors()) {
            return "cuenta/crearForm";
        }
    }
}
```

```
// Si todo está bien, crea la cuenta y la agrega a la lista
// "cuentas", luego redirige hacia el detalle /cuenta/{id},
// controlado por el método handler "verDetalle(..) de mas abajo.
this.cuentas.put(cuenta.asignarId(), cuenta);
return "redirect:/cuenta/" + cuenta.getId();
}

@RequestMapping(value="{id}", method=RequestMethod.GET)
public String verDetalle(@PathVariable Long id,
    // Ejemplo de @ModelAttribute pasado como parametro
    @ModelAttribute("listaProductos") List<String> listaProductos2,
    Model model) {
    Cuenta cuenta = this.cuentas.get(id);
    if (cuenta == null) {
        throw new RecursoNoEncontradoException(id);
    }
    model.addAttribute("cuenta", cuenta);
    model.addAttribute("listaProductos2", listaProductos2);
    // return "cuenta/detalle"; // Muestra el detalle de la cuenta en formato formulario
    return "cuenta/detalle2"; // Muestra el detalle de la cuenta en formato tabla html
}

// Creamos un atributo del modelo "listaProductos" con la anotación
// @ModelAttribute retornando los valores del método "poblarListaProductos()".
// El atributo "listaProductos" podrá ser usado en la vista jsp como atributo.
@ModelAttribute("listaProductos")
public List<String> poblarListaProductos() {
    List<String> lista = new ArrayList<String>();
    lista.add("Cuenta Corriente");
    lista.add("Tarjeta de crédito VISA");
    lista.add("Tarjeta de crédito MASTERCARD");
    lista.add("Fondos Mutuos");
    lista.add("Crédito Hipotecario");

    return lista;
}
}
```

6. Abrir y estudiar la vista jsp detalle2.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<html>
<head>
<title>Mi Cuenta</title>
<style>
.error {
    color: #ff0000;
    font-weight: bold;
}
</style>
</head>
<body>
<table border="1">
    <tr>
        <th>Id</th>
        <th>Nombre</th>
        <th>Saldo</th>
        <th>Endeudamiento</th>
        <th></th>
    </tr>
    <tr>
        <td>${cuenta.id}</td>
        <td>${cuenta.nombre}</td>
        <td>${cuenta.saldo}</td>
        <td>${cuenta.nivelEndeudamiento}</td>
    </tr>
</table>
<br />
<table>
    <thead>
        <tr>
            <th>Producto</th>
        </tr>
    </thead>
    <c:forEach var="item" items="${listaProductos2}">
        <tr>
            <td>${item}</td>
        </tr>
    </c:forEach>
</table>
<a href="..">Inicio</a>
</body>
</html>
```

Resumen

En este capítulo vimos cómo funciona el componente Form de Spring, cómo se puede generar un formulario XHTML, la manera que se agregan los elementos de todo formulario y de qué forma se pueden agregar validadores.

La próxima tarea se concentrará en aplicar todos estos conceptos.

FIN.

Envía tus consultas a los foros!

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

Lectura Recomendada y Bibliografía

- [Form Tags](#): Recomendable lectura de esta sección del manual oficial para complementar con este workshop.
- [Spring Validation](#): recomendable lectura del manual oficial para complementar con este workshop.
- [Handling Forms in Spring MVC](#): recomendable lectura de un artículo en inglés.
- [Spring type conversion and validation](#): lectura recomendada.