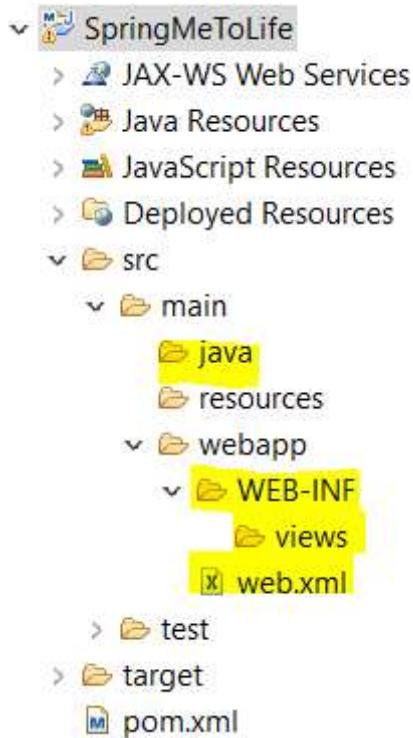


Setup nuova web application con Spring usando Maven senza Archetype

Con Eclipse Neon per EE developer proviamo a generare un nuovo progetto che abbia Spring MVC web:

File > Nuovo Progetto Maven > Skip Archetype (Create simple project)

Dopo generiamo le seguenti cartelle (evidenziate in giallo):



Fatto questo aggiustiamo il pom e il web xml:

pom.xml:

```

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>
  
```

```
<groupId>it.euris.example</groupId>
<artifactId>SpringMeToLife</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>war</packaging>

<properties>
    <java-version>1.7</java-version>
    <springframework.version>4.3.1.RELEASE</springframework.version>
    <jackson.version>2.7.5</jackson.version>
</properties>

<dependencies>
    <dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-web-api</artifactId>
        <version>7.0</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>
</dependencies>

<build>
    <finalName>SpringMeToLife</finalName>
    <pluginManagement>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.3.2</version>
                <configuration>
                    <source>${java-version}</source>
                    <target>${java-version}</target>
                
```

```

</configuration>

</plugin>

<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-war-plugin</artifactId>
    <version>2.4</version>
    <configuration>
        <warSourceDirectory>src/main/webapp</warSourceDirectory>
        <warName>SpringMeToLife</warName>
        <failOnMissingWebXml>false</failOnMissingWebXml>
    </configuration>
</plugin>

</plugins>

</pluginManagement>

</build>

</project>

```

web.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

    <display-name>Spring me to life!</display-name>

</web-app>

```

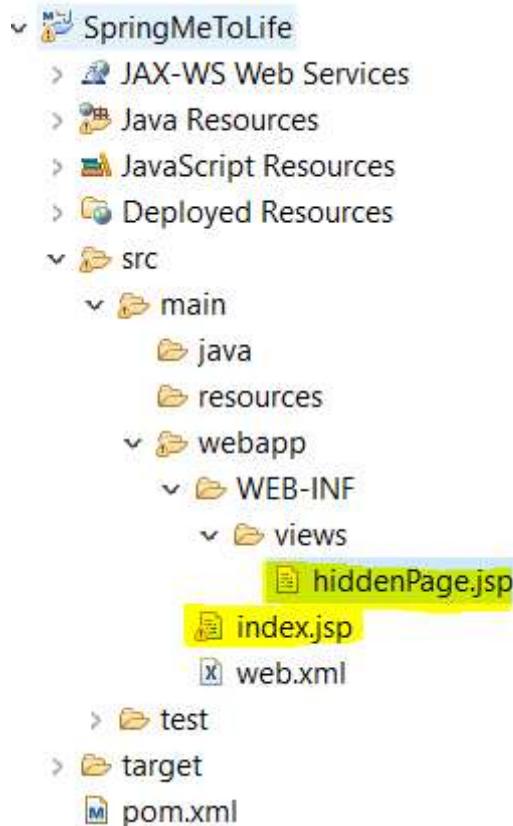
Clicco col destro sull'albero del progetto e vado in Proprietà > Java Compiler e setto 1.7.

Poi vado in Server Facets e setto Tomcat 7 e 9.

Poi faccio salva, refresh e Maven > Update Project.

E siamo pronti per iniziare!

Creiamo delle pagine JSP per la visualizzazione di dati: una pubblica in webapp e una nascosta in webapp>WEB-INF>views



Con la seguente struttura (facendo attenzione che il linguaggio EL sia abilitato grazie all'apposito TAG (in rosso):

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<%@ page isELIgnored="false" %>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Hidden Page</title>
</head>
<body>

```

```
<p> Hidden Page </p>
</body>
</html>
```

Creiamo una servlet classica prima di inserire Spring, così da confrontare i due approcci.

andiamo su main/java e clicchiamo col destro. Facciamo New >Other>Java Class.

Si apre la finestra di creazione della classe Java e inseriamo il package: it.euris.example.servlets e la Classe la chiamiamo MyClassicServlet:

```
package it.euris.example.servlets;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(
    name = "myClassicServlet",
    urlPatterns = {" /ciao", " /salve"},
    loadOnStartup = 1
)
public class MyClassicServlet extends HttpServlet{

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
    IOException{
    }

    @Override
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException{
        this.doGet(request, response);
    }
}
```

```
}
```

```
}
```

Adesso creiamo un'entità cliccando sulla cartella main>java e facendo new >other>Java Class:

```
package: it.euris.example.models
```

```
name: User.
```

```
package it.euris.example.models;
```

```
public class User {
```

```
    private String name;
```

```
    private String surname;
```

```
    public User(String _name, String _surname){
```

```
        this.name = _name;
```

```
        this.surname = _surname;
```

```
}
```

```
    public User(){}
```

```
    public String getName() {
```

```
        return name;
```

```
}
```

```
    public void setName(String name) {
```

```
        this.name = name;
```

```
}
```

```
    public String getSurname() {
```

```
        return surname;
```

```
}
```

```
    public void setSurname(String surname) {
```

```
this.surname = surname;
}

}
```

Adesso usiamo questa classe nelle servlet e rimandiamo i dati alle pagine JSP:

```
public class MyClassicServlet extends HttpServlet{

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException{
        User myUser = new User("Armando", "Donzelli");
        request.setAttribute("mioUtente", myUser);
        request.getRequestDispatcher("/WEB-INF/views/hiddenPage.jsp").forward(request, response);
    }

    @Override
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException{
        this.doGet(request, response);
    }

}
```

E inseriamo il parametro appena passato alla pagina hiddenPage.jsp:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<%@ page isELIgnored="false" %>
<!DOCTYPE html>
<html>
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Hidden Page</title>
</head>
<body>
<p> Hidden Page </p>
<p> ${mioUtente.name} </p>
</body>
</html>
```

L'Expression Language nella pagina JSP va a recuperare direttamente i dati dalla request attraverso le variabili implicite.

Non c'è bisogno di nominare la request oppure neanche di usare il getName() dell'entità User, e neanche di importare User.

Fatto questo possiamo avviare Spring. Facciamo partire la DispatcherServlet da DispatcherInitializer e usiamo la configurazione dei beans da java.

Però prima di tutto questo dobbiamo aggiornare le dipendenze del **pom.xml di maven**:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <groupId>it.euris.example</groupId>
    <artifactId>SpringMeToLife</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging>

    <properties>
        <java-version>1.7</java-version>
        <springframework.version>4.3.1.RELEASE</springframework.version>
        <jackson.version>2.7.5</jackson.version>
    </properties>

    <dependencies>
        <!-- J2EE Classic -->
```

```
<dependency>
  <groupId>javax</groupId>
  <artifactId>javaee-web-api</artifactId>
  <version>7.0</version>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>

<!-- Spring -->

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-core</artifactId>
  <version>${springframework.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-web</artifactId>
  <version>${springframework.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>${springframework.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-tx</artifactId>
  <version>${springframework.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-orm</artifactId>
```

```
<version>${springframework.version}</version>

</dependency>

</dependencies>

<build>

    <finalName>SpringMeToLife</finalName>

    <pluginManagement>

        <plugins>

            <plugin>

                <groupId>org.apache.maven.plugins</groupId>

                <artifactId>maven-compiler-plugin</artifactId>

                <version>2.3.2</version>

                <configuration>

                    <source>${java-version}</source>

                    <target>${java-version}</target>

                </configuration>

            </plugin>

            <plugin>

                <groupId>org.apache.maven.plugins</groupId>

                <artifactId>maven-war-plugin</artifactId>

                <version>2.4</version>

                <configuration>

                    <warSourceDirectory>src/main/webapp</warSourceDirectory>

                    <warName>SpringMeToLife</warName>

                    <failOnMissingWebXml>false</failOnMissingWebXml>

                </configuration>

            </plugin>

        </plugins>

    </pluginManagement>

</build>

</project>
```

procediamo ad aggiornare le dipendenze di progetto dopo aver salvato le modifiche del pom.xml con **Maven>Update Project** cliccando col destro sul nome del progetto.

Adesso creiamo il file di Configurazione Java contentente l'istanziazione dei vari Java Beans che Spring andrà ad iniettare.

Facciamo main>java> new Class:

package: it.euris.example.configurations

class: SpringConfiguration.java

```
package it.euris.example.configurations;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.ViewResolver;
import org.springframework.web.servlet.config.annotation.DefaultServletHandlerConfigurer;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
import org.springframework.web.servlet.view.JstlView;

import it.euris.example.models.User;

/**Classe che istanzia i vari Beans. Si definisce un package in cui Spring deve scansionare per trovare le classi entità dalle quali creare i Bean qui definiti **/
```

```
@Configuration
@EnableWebMvc
@ComponentScan(basePackages = "it.euris.example")

public class SpringConfiguration extends WebMvcConfigurerAdapter{

    @Override
    public void configureDefaultServletHandling(DefaultServletHandlerConfigurer configurer) {
        configurer.enable();
    }
}
```

```
/**Questo BEAN si occupa di collegare il nome logico nel Controller ad una View esistente: NON DOBBIAMO
INIETTARLO NOI**/

@Bean

public ViewResolver viewResolver() {

    InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();

    viewResolver.setViewClass(JstlView.class);

    viewResolver.setPrefix("/WEB-INF/views/");

    viewResolver.setSuffix(".jsp");

    return viewResolver;

}

/** INIZIO DEFINIZIONE DEI BEANS CHE CI INTERESSANO **/

@Bean

public User mioUtente(){

    return new User();

}

@Bean

public User luciaUtente(){

    return new User("Lucia", "Distante");

}

}
```

Adesso andiamo a far partire la DispatcherServlet di Spring (non possiamo usare la annotation sulla definizione della classe DispatcherServlet purtroppo), quindi facciamo partire questa servlet speciale con questa classe.

Facciamo main>java> new Class:

```
package: it.euris.example.configurations

class:    DispatcherInitializer
```

```
package it.euris.example.configurations;
```

```
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
```

```

import javax.servlet.ServletRegistration;

import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;

/**Classe che si occupa di inizializzare la Dispatcher Servlet di Spring senza dover usare il Deployment Descriptor*/
public class DispatcherInitializer implements WebApplicationInitializer {

    public void onStartup(ServletContext container) throws ServletException {
        AnnotationConfigWebApplicationContext ctx = new AnnotationConfigWebApplicationContext();
        ctx.register(SpringConfiguration.class);
        ctx.setServletContext(container);

        ServletRegistration.Dynamic servlet = container.addServlet("dispatcher", new DispatcherServlet(ctx));

        servlet.setLoadOnStartup(1);
        servlet.addMapping("/");
    }
}

```

Adesso manca da definire il Controller di Spring, una classe annotata a cui la DispatcherServlet va ad inoltrare tutte le richieste http a seconda del request mapping, ovvero dell'url della richiesta.

Creiamo una nuova classe in: main>java> new Class:

```

package: it.euris.example.controllers
class:      SpringController.java

```

```
package it.euris.example.controllers;
```

```

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

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.servlet.ModelAndView;

import it.euris.example.models.User;

@Controller

public class SpringController {

    @Autowired

    private User mioUtente;

    @Autowired

    public User luciaUser;

    @RequestMapping(value={"/ciaoSpring", "/salveSpring"}, method=RequestMethod.GET)

    public String goToPage(){

        return "hiddenPage";

    }

    @RequestMapping(value={"/ciaoSpringUser", "/salveSpringUser"}, method=RequestMethod.GET)

    public ModelAndView goToPage(@RequestParam("name") String nome, @RequestParam("surname") String cognome){

        //istanza/bean iniettata

        mioUtente.setName(nome);

        mioUtente.setSurname(cognome);

        ModelAndView modAndView = new ModelAndView("hiddenPage"); //pagina jsp associata al modello

        modAndView.addObject("mioUtente", mioUtente); //aggiungiamo i dati col nome

        mioUtente

        return modAndView;

    }

    @RequestMapping(value={"/tuuts"}, method=RequestMethod.GET)

    public ModelAndView goToLucia(){

}

```

```

ModelAndView modAndView = new ModelAndView("hiddenPage");      //pagina jsp associata al
modello

modAndView.addObject("mioUtente", luciaUser);                  //aggiungiamo i dati col nome
mioUtente

return modAndView;

}

}

```

Ricordiamo che la pagina jsp in main>webapp>WEB-INF>views>hiddenPage.jsp

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<%@ page isELIgnored="false" %>
<!DOCTYPE html>
<html>

<head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
    <title>Hidden Page</title>
</head>
<body>
    <p> Hidden Page </p>
    <p> ${mioUtente.name} </p>
</body>
</html>

```

Le chiamate a dover esser provate sono:

```

http://localhost:8080/SpringMeToLife/ciaoSpringUser?name=Andrea&surname=Fornaro
http://localhost:8080/SpringMeToLife/tuuts

```

