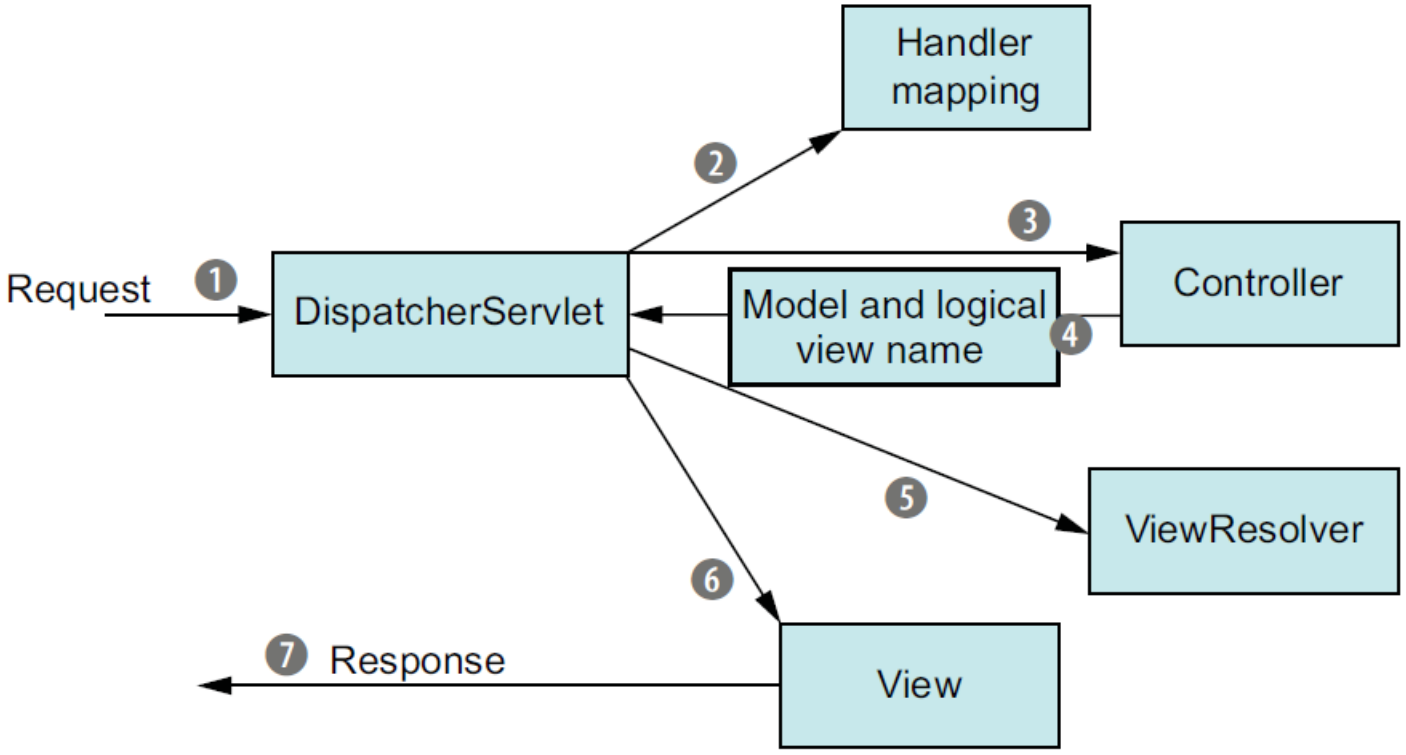
Spring in Web Applications

- ✓ Spring MVC
- ✓ Spring Security
- ✓ JSF with Spring
- ✓ Unit testing


Handler mapping
2
3
Controller
Request
1
DispatcherServlet
Model and logical view name
4
5
ViewResolver
6
7
Response
View

Define an app initializer

```
public class MyAppInitializer extends
        AbstractAnnotationConfigDispatcherServletInitializer{

        // … app initializer definition here
}
```

Define an app initializer

```
@Override
 protected Class<?>[] getRootConfigClasses() {
  return new Class[]{RootConfig.class};
 }


@Override
 protected Class<?>[] getServletConfigClasses() {
  return new Class[]{WebConfig.class};
 }
```

Define an app initializer

```
@Override
 protected String[] getServletMappings() {
  return new String[]{"/"};
 }
```

Define an app initializer

```
public class MyAppInitializer extends AbstractAnnotationConfigDispatcherServletInitializer{

  @Override
  protected Class<?>[] getRootConfigClasses() {
    return new Class[]{RootConfig.class};
  }

  @Override
  protected Class<?>[] getServletConfigClasses() {
    return new Class[]{WebConfig.class};
  }

  @Override
  protected String[] getServletMappings() {
    return new String[]{"/"};
  }

}
```

Create the web configuration

```
@Configuration
@EnableWebMvc
@ComponentScan("ro.laurentiuspilca.*")
public class WebConfig extends WebMvcConfigurerAdapter
{

}
```

Create the web configuration

```
@Bean
 public ViewResolver viewResolver() {

  InternalResourceViewResolver resolver
                              = new InternalResourceViewResolver();

  resolver.setPrefix("/WEB-INF/views/");
  resolver.setSuffix(".jsp");
  resolver.setExposeContextBeansAsAttributes(true);

  return resolver;
 }
```

With XML:

```
<bean id="viewResolver"
        class="org.springframework.web.servlet.view.
                                        InternalResourceViewResolver"
                p:prefix="/WEB-INF/views/"
                p:suffix=".jsp“
 />
```

Create the web configuration

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

Create the root configuration

```
@Configuration
@ComponentScan(basePackages = {"ro.laurentiuspilca.*"},
      excludeFilters = {
        @Filter(type = FilterType.ANNOTATION, value =
                    EnableWebMvc.class)
      })
public class RootConfig {

}
```

Define a controller

```
@Controller
public class HomeController {

  @RequestMapping(value = "/", method = RequestMethod.GET)
  public String home() {
    return "home";
  }
}
```

The View

home.jsp

Source History

```
<%--
    Document   : home
    Created on : Apr 10, 2016, 10:33:59 PM
    Author     : Laurentiu.S
--%>

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

Link the model to the view

```
@RequestMapping(value = "/", method = RequestMethod.GET)
 public String home( Model model ) {
  model.addAttribute("name", "John");
  return "home";
 }
```

Link the model to the view

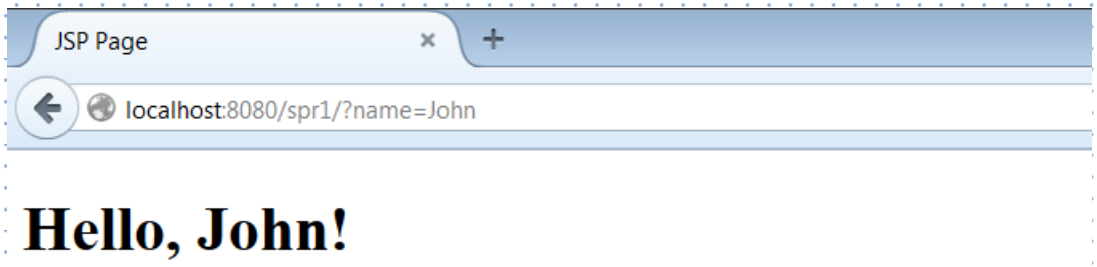
```
@RequestMapping(value = "/", method = RequestMethod.GET)
 public String home( Map<String, String> model ) {
  model.put("name", "John");
  return "home";
 }
```

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

Taking input parameters

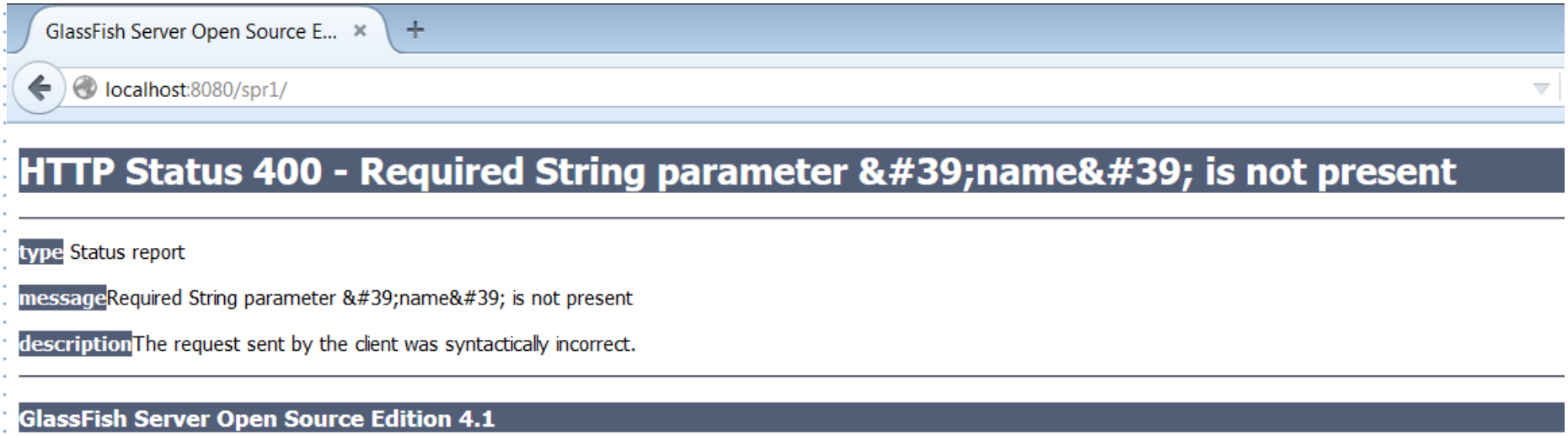
```
@RequestMapping(value = "/", method = RequestMethod.GET)
 public String home(
                        @RequestParam("name") String name,
                        Model model
                        ) {
  model.addAttribute("name", name);
  return "home";
 }
```

http://localhost:8080/spr1/?name=John



Hello, John!

http://localhost:8080/spr1/



Path Variable

```
@RequestMapping( value = "/{name}",
                 method = RequestMethod.GET)
 public String home( @PathVariable("name") String name,
                 Model model) {

  model.addAttribute("name", name);
  return "home";

 }
```

http://localhost:8080/spr1/**John**

Forms and the POST method

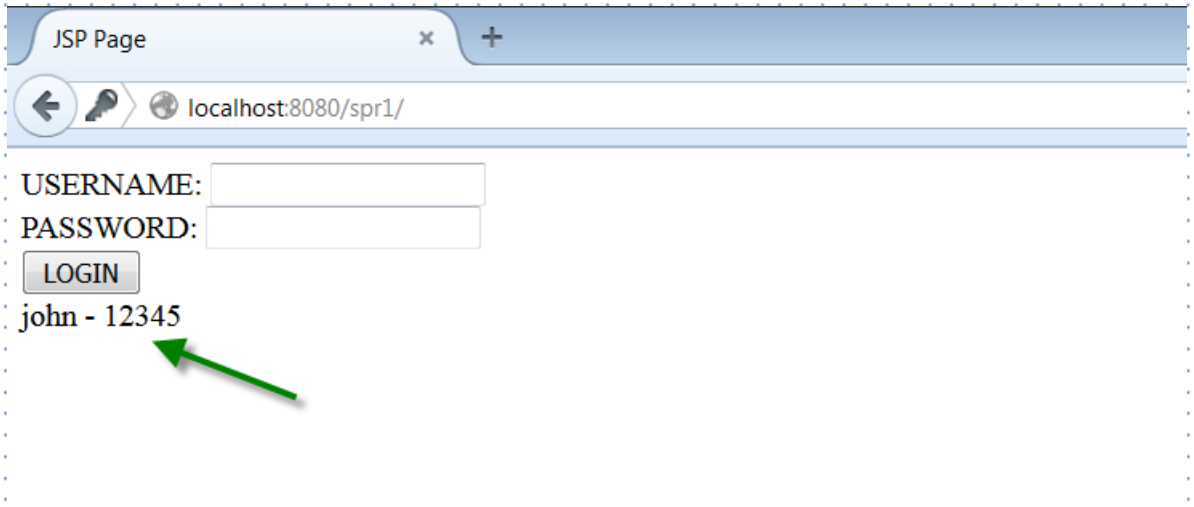
```
@RequestMapping(value = "/", method =
                        RequestMethod.POST)
 public String home( String name, Model model ) {
  model.addAttribute("name", name);
  return "home";
 }
```

```
<form method="post">
    NAME: <input type="text" name="name" />
    <input type="submit" name="sayHello" value="Say Hello" />
</form>

${name}
```

Mapping an object

```
public class User implements java.io.Serializable{

  private String username;
  private String password;

  // getter and setters here
}
```

```
@RequestMapping(value = "/", method = RequestMethod.POST)
 public String home( User user , Model model) {

  model.addAttribute("username", user.getUsername());
  model.addAttribute("password", user.getPassword());
  return "home";

 }
```


JSP Page
localhost:8080/spr1/
USERNAME:
PASSWORD:
LOGIN
john - 12345

Validations

Annotation	Description
`@AssertFalse`	The annotated element must be a Boolean type and be `false`.
`@AssertTrue`	The annotated element must be a Boolean type and be `true`.
`@DecimalMax`	The annotated element must be a number whose value is less than or equal to a given `BigDecimalString` value.
`@DecimalMin`	The annotated element must be a number whose value is greater than or equal to a given `BigDecimalString` value.
`@Digits`	The annotated element must be a number whose value has a specified number of digits.
`@Future`	The value of the annotated element must be a date in the future.
`@Max`	The annotated element must be a number whose value is less than or equal to a given value.
`@Min`	The annotated element must be a number whose value is greater than or equal to a given value.
`@NotNull`	The value of the annotated element must not be `null`.
`@Null`	The value of the annotated element must be `null`.
`@Size`	The value of the annotated element must be either a `String`, a collection, or an array whose length fits within the given range.

Validations

```
public class User implements java.io.Serializable{

  @NotNull
  @Size(min=5, max=50)
  private String username;

  @NotNull
  @Size(min=5, max=50)
  private String password;

// getters and setters here
}
```

```
@RequestMapping(value = "/", method = RequestMethod.POST)
 public String home(@Valid User user,Errors errors,Model model) {

  model.addAttribute("username", user.getUsername());
  model.addAttribute("password", user.getPassword());

  if(errors.hasErrors()){
   List<ObjectError> errorList = errors.getAllErrors();
   errorList.forEach(System.out::println);
  }

  return "home";
 }
```

Now… try to insert a non valid user…

Field error in object 'user' on field 'username': rejected value [john]; codes … blah blah blah

JstlView Resolver

```
@Bean
public ViewResolver viewResolver() {
  InternalResourceViewResolver resolver
                              = new InternalResourceViewResolver();

  resolver.setPrefix("/WEB-INF/views/");
  resolver.setSuffix(".jsp");

  resolver.setViewClass(
        org.springframework.web.servlet.view.JstlView.class);

  resolver.setExposeContextBeansAsAttributes(true);
  return resolver;
}
```

JSP tag	Description
`<sf:checkbox>`	Renders an HTML `<input>` tag with `type` set to `checkbox`.
`<sf:checkboxes>`	Renders multiple HTML `<input>` tags with `type` set to `checkbox`.
`<sf:errors>`	Renders field errors in an HTML `<span>` tag.
`<sf:form>`	Renders an HTML `<form>` tag and exposed binding path to inner tags for data-binding.
`<sf:hidden>`	Renders an HTML `<input>` tag with `type` set to `hidden`.
`<sf:input>`	Renders an HTML `<input>` tag with `type` set to `text`.
`<sf:label>`	Renders an HTML `<label>` tag.
`<sf:option>`	Renders an HTML `<option>` tag. The `selected` attribute is set according to the `bound` value.
`<sf:options>`	Renders a list of HTML `<option>` tags corresponding to the bound collection, array, or map.
`<sf:password>`	Renders an HTML `<input>` tag with `type` set to `password`.
`<sf:radiobutton>`	Renders an HTML `<input>` tag with `type` set to `radio`.
`<sf:radiobuttons>`	Renders multiple HTML `<input>` tags with `type` set to `radio`.
`<sf:select>`	Renders an HTML `<select>` tag.
`<sf:textarea>`	Renders an HTML `<textarea>` tag.

```
<%@ taglib uri="http://www.springframework.org/tags/form"
prefix="sf"
%>
```

In view:

```
<sf:form method="POST" commandName="login">

    Username: <sf:input path="username" /><br/>
    Password: <sf:password path="password" /><br/>
    <input type="submit" value="Login" />

</sf:form>
```

In controller:

```
@RequestMapping(value = "/", method = RequestMethod.GET)
 public String home(Model model) {
  model.addAttribute(new User());
  return "home";
 }
```

Spring Security

Spring Security is a **security framework** that provides **declarative security** for your Spring-based applications.

The security-config.xml

```
<beans:beans xmlns="http://www.springframework.org/schema/security"
        xmlns:beans="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-3.0.xsd
http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security-3.2.xsd">



</beans:beans>
```

Defining and HTTP security configuration

```
<http auto-config="true" use-expressions="true">
    <intercept-url pattern="/**" access="permitAll" />
</http>
```

Login action

```
<http auto-config="true" use-expressions="true">
    <intercept-url pattern="/**" access="permitAll" />
    <form-login
      login-page="/login"
      default-target-url="/success.htm"
      authentication-failure-url="/login.htm?error"
      username-parameter="j_username"
      password-parameter="j_password" />


</http>
```

Logout action

```
<http auto-config="true" use-expressions="true">
    <intercept-url pattern="/**" access="permitAll" />
    <form-login
      login-page="/login"
      default-target-url="/success.htm"
      authentication-failure-url="/login.htm?error"
      username-parameter="j_username"
      password-parameter="j_password" />
    <logout logout-success-url="/login.htm" />
  </http>
```

Authentication manager

```
<authentication-manager>
        <authentication-provider>

        <password-encoder hash="md5"/>

        </authentication-provider>
 </authentication-manager>
```

Authentication manager

```
<jdbc-user-service
    data-source-ref="dataSource"
    users-by-username-query=
        "SELECT u.username, u.password, u.enabled from users
        u where u.username=?"
    authorities-by-username-query=
      "SELECT u.username, ur.role FROM users u left join
   user_roles ur on u.id=ur.username
   WHERE u.username=?"

/>
```

```
<form:form action='j_spring_security_check' method='POST'>
    <table>
        <tr>
            <td>User Name:</td>
            <td><input type='text' name='j_username'/></td>
        </tr>
        <tr>
            <td>Password:</td>
            <td><input type='password' name='j_password'/></td>
        </tr>
        <tr>
            <td colspan='2'><input name="submit" type="submit" value="Login"/></td>
        </tr>
    </table>
</form:form>
```

- **@PreAuthorize** – the check is done before the method call.

- **@PostAuthorize** – the check is done after the method is called and return type can be used in the check

```
public interface IService {

    @PreAuthorize("hasRole(1)")
    public void serviceMethodExample();

    @PostAuthorize ("returnObject == authentication.name")
    public String postAuthorizedExampleMethod();
}
```

Defining security in Java

- **@EnableWebMvcSecurity** – over a configuration class for web MVC project security
- Extend **WebSecurityConfigurerAdapter**
- Override on of the three **configure()** methods

protected void configure(**AuthenticationManagerBuilder** auth)
throws Exception

protected void configure(**HttpSecurity** http)
throws Exception

protected void configure(**WebSecurity** web)
throws Exception

```
protected void configure( AuthenticationManagerBuilder auth)
        throws Exception


auth.
   jdbcAuthentication()
        .dataSource(dataSource)
                .usersByUsernameQuery(query1)
        .authoritiesByUsernameQuery(query2)
        .passwordEncoder(new Md5PasswordEncoder());
```

JSF - Java Server Faces

- **Faces Servlet**
  - Servlet that managed and translates facelets into functionality
- **Facelets**
  - XML tags defined by a library that implements functionality on both server and client side

JSF - Java Server Faces

Basic parts of a JSF project

1. JSF library – 1.x / 2.x (mandatory)
2. Component library (E. g. Primefaces or RichFaces) – optional
3. Utility library (E. g. Omnifaces) – optional
4. Server side business implementation (JEE or Spring) - optional

The faces-config.xml file

- Configuration file
- Similar to Spring XML file
- Used to define beans in JSF context
- Used to define navigation cases

Starting a Spring based JSF web project

Inside faces-config.xml file define a Spring bean faces EL resolver:

```
<application>
    <el-resolver>
        org.springframework.web.jsf.el.SpringBeanFacesELResolver
    </el-resolver>
</application>
```

Define Spring context listeners in web.xml

```
<listener>
    <listener-class>
        org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
<listener>
    <listener-class>
        org.springframework.web.context.request.RequestContextListener
    </listener-class>
</listener>
```

Anatomy of a facelet

```
<h:inputText value="#{bean.attribute}" />
```

The JSF View

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/x
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:p="http://primefaces.org/ui">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        <h:form>
            <p><p:inputText value="#{product.name}" /></p>
            <p><p:commandButton value="ADD" actionListener="#{product.addProduct()}" /></p>
        </h:form>
    </h:body>
</html>
```

JSF Context

- JSF uses its own context independent of the Spring/EE context used
- JSF context does not know the beans in Spring context
- JSF context can be replaced by CDI since JSF 2.x
- Which means that an application using JSF 2.x and Spring > 3 can have the same context if CDI annotations are used

JSF Context

- When using CDI, the CDI annotations (@Named, @Inject etc) only these are enough to mark beans in both contexts – JSF and Spring

- When using JSF and Spring context individually, both annotation can and must be use on the class

- Alternatively, in both JSF and Spring XML can be used for bean definition instead of annotations.

JSF Context - Annotations

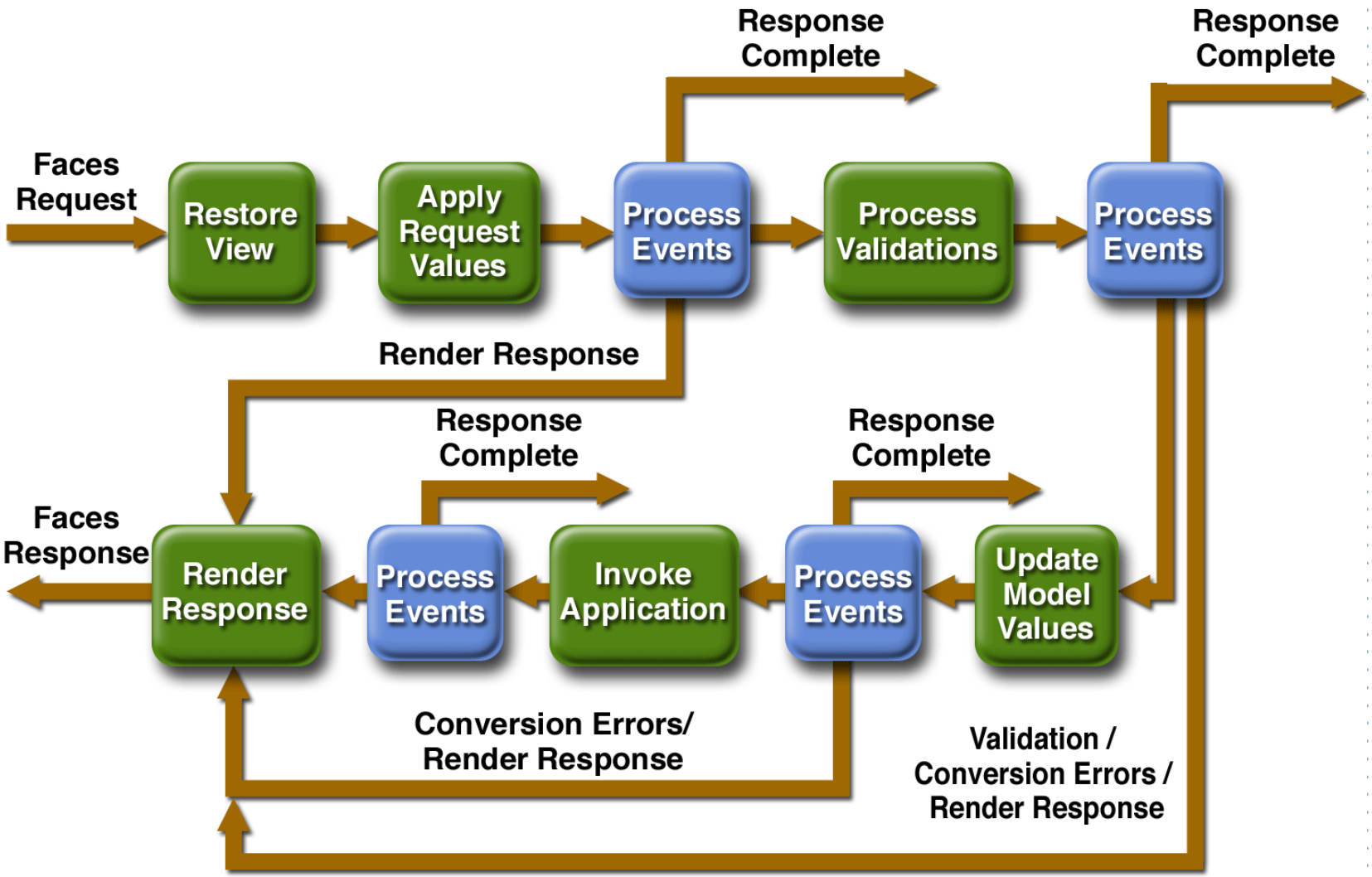
For bean definition and Dependency Injection

- @ManagedBean
- @ManagedProperty

For bean scope

- @RequestScoped
- @ViewScoped
- @SessionScoped
- @ApplicationScoped

JSF request life cycle



Unit Testing

- One test covers one elementary part of the implementation
- We call one of these tests a unit test
- We call a set of tests a test suite
- First level of testing implementation
- Other levels of testing follow the unit testing:
  - Component testing
  - Integration testing
  - Performance testing
  - Acceptance testing

Unit Testing

Some methodologies use test as the center of implementation – **first stage**

We call this TDD – test driven development

Unit testing in Spring

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>4.1.7.RELEASE</version>
    <type>jar</type>
</dependency>
```

Definition of a simple unit test

@RunWith – defines the used engine for unit testing

@ContextConfiguration – defines the configuration of the context. Can be supplied a set of configuration files, a set of configuration classes or both

@Test – standard junit annotation to define a test method

Definition of a simple unit test

First write the test suite:

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath: config.xml")
public class MyFirstTestSuite{
}
```

Definition of a simple unit test

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath: config.xml")
public class MyFirstTestSuite{

        @Autowired
        private Obj obj;

        @Test
        public void myTest(){
                Assert.notNull(obj);
        }
}
```

Repeating a test case

If you want to check that a test case does not fail if repeated multiple times, Spring can be instructed to repeat it using the **@Repeat** annotation.

```
@Test
@Repeat(10)
public void myTest(){
}
```

Add a time limit

If the test should be finished in a defined time frame **@Timed** annotation can be used to instruct Spring the test should end in some specific time limit.

```
@Test
@Timed(millis = 10000) // no more than 10 seconds
public void myTest(){
}
```

Transactions in test cases

@TransactionConfiguration annotation can be used to specify a test suite which is the transaction manager to be used

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath: config.xml")
@TransactionConfiguration(transactionManager = "tx")
@Transactional
public class MyTestSuite{
}
```

Test Suite life cycle

@Before

- marks method to be executed before the test case

@BeforeTransaction

- marks method to be executed before the transaction starts

@After

- marks method to be executed after the test case

@AfterTransaction

- marks method to be executed after the transaction ends