Handling requests

1. request → **Dispatcher Servlet (front**

2. delegate request →

Control

3. model and view

6. response

5. return control

4. model

View Template (JSP)
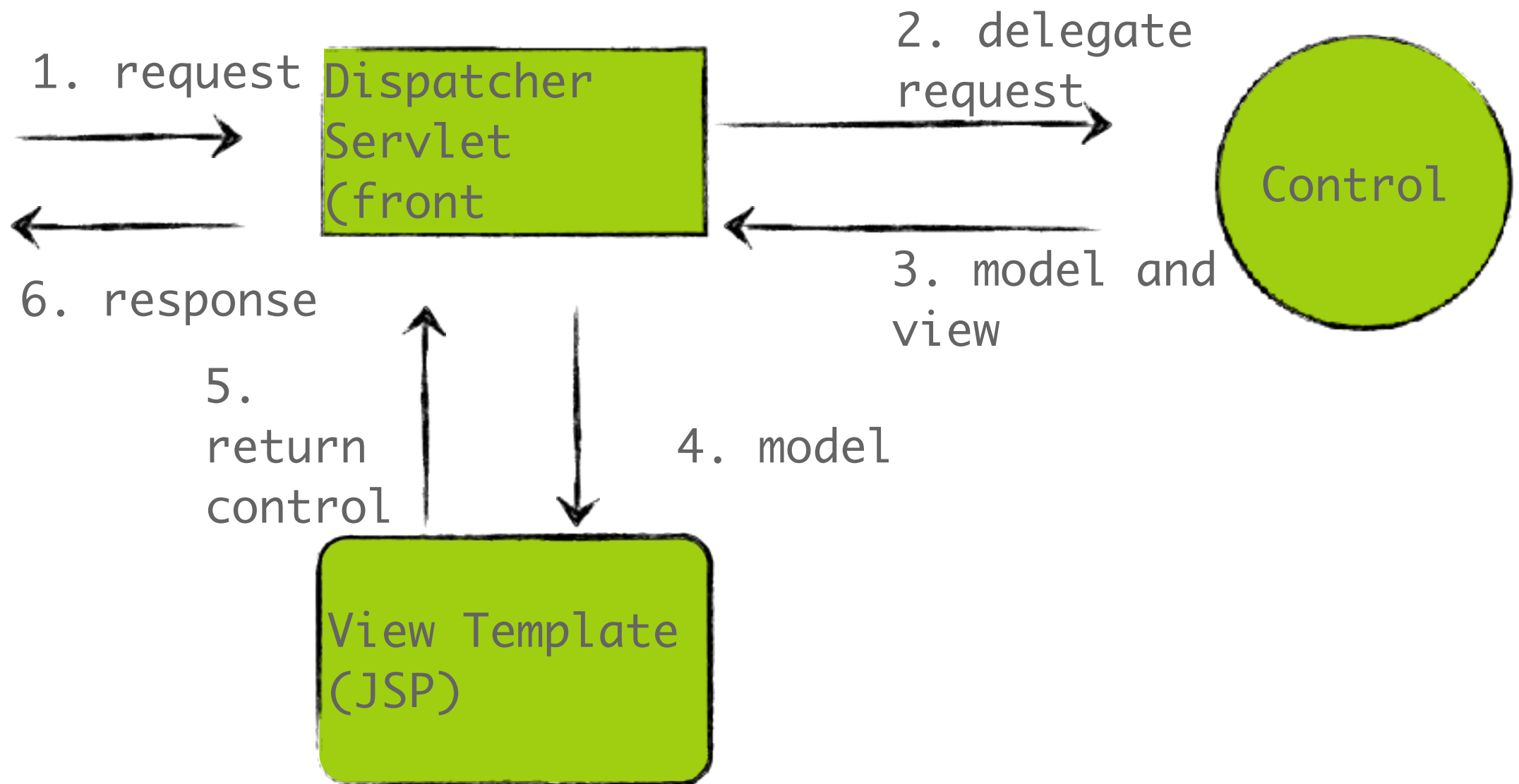
Configuring Web MVC

Add the Dispatcher Servlet to web.xml

Needs a `WEB-INF/spring-servlet.xml` config file

```xml
<servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>/spring/*</url-pattern>
</servlet-mapping>
```

Additional configuration files

web.xml

```xml
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/applicationContext.xml
        /WEB-INF/applicationContext-security.xml
    </param-value>
</context-param>

<listener>
    <listener-class>
        org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
```
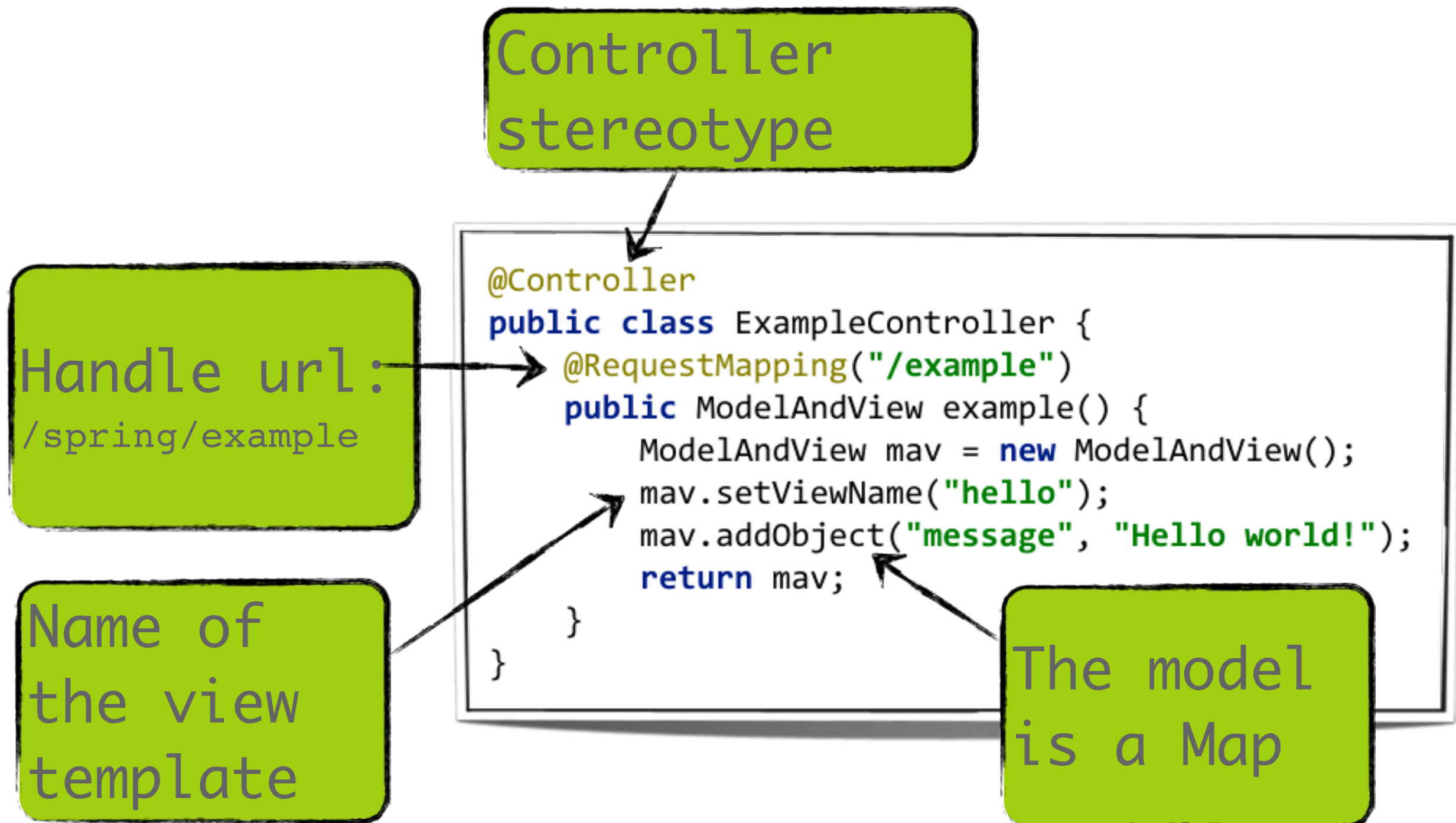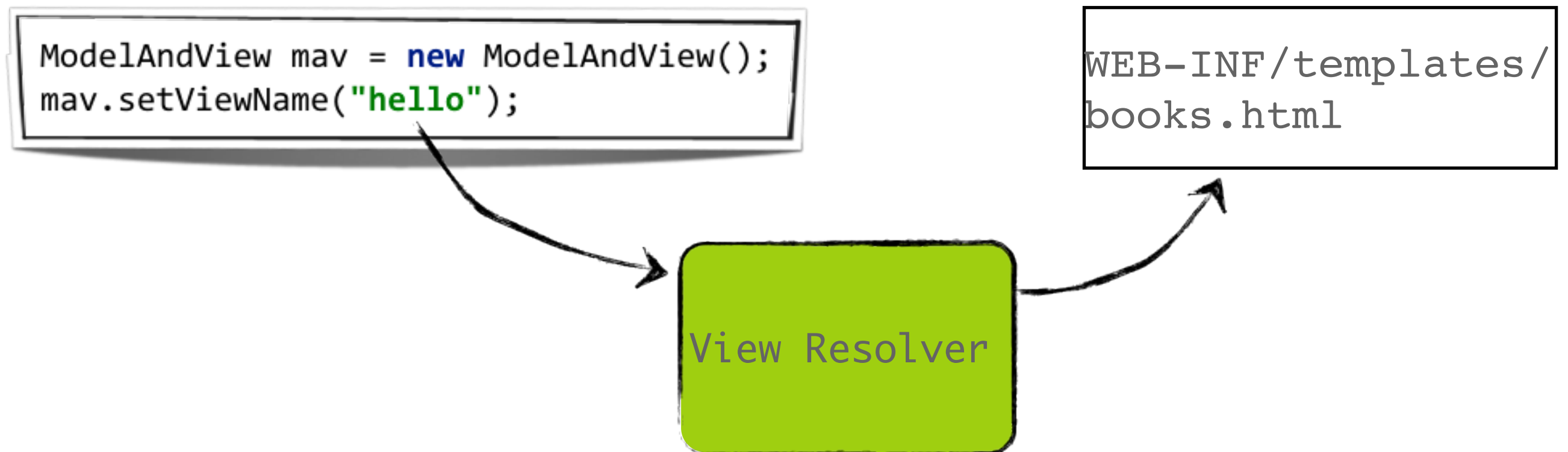
.infoSupport
Solid Innovator

Implementing controllers

Controller stereotype

Handle url:
/spring/example

Name of the view template

The model is a Map

```java
@Controller
public class ExampleController {
    @RequestMapping("/example")
    public ModelAndView example() {
        ModelAndView mav = new ModelAndView();
        mav.setViewName("hello");
        mav.addObject("message", "Hello world!");
        return mav;
    }
}
```

Thymeleaf page

- Use Expression Language to render model values

- Use thymeleaf and Spring tags to make pages dynamic

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<body>
 <table >
    <tr>
        <th>Id</th><th>Title</th>
    </tr>
    <tr th:each="book : ${books}">
        <td th:text="${book.id}">1</td>
        <td th:text="${book.title}">Java 10</td>
    </tr>
  </table>
</body>
</html>
```

View resolvers

- View names are resolved to views using view resolvers

```
ModelAndView mav = new ModelAndView();
mav.setViewName("hello");
```

View Resolver

```
WEB-INF/templates/
books.html
```

Most useful View resolvers

View resolver	Explanation
`InternalResourceViewResolver`	Servlet and JSP view resolver. View names are prefixed and suffixed to generate file name
`ResourceBundleViewResolver`	Views are defined in views.properties. Supports multiple kinds of views working together (e.g. JSP and Excel)
`ContentNegotiatingViewResolver`	Resolve views based on the *accept* header in the HTTP request. Useful for RESTful web services

There are more view resolvers,
but are less likely to be used

Resolving Thymeleaf views

```java
@Bean
public SpringResourceTemplateResolver templateResolver(){

    SpringResourceTemplateResolver templateResolver =
                        new SpringResourceTemplateResolver();
    templateResolver.setApplicationContext(this.applicationContext);
    templateResolver.setPrefix("/WEB-INF/templates/");
    templateResolver.setSuffix(".html");

    templateResolver.setTemplateMode(TemplateMode.HTML);

    return templateResolver;
}
```

Resolving Thymeleaf views cont.

```java
@Bean
public SpringTemplateEngine templateEngine(){
    // SpringTemplateEngine automatically applies SpringStandardDialect and
    // enables Spring's own MessageSource message resolution mechanisms.
    SpringTemplateEngine templateEngine = new SpringTemplateEngine();
    templateEngine.setTemplateResolver(templateResolver());

    templateEngine.setEnableSpringELCompiler(true);
    return templateEngine;
}

@Bean
public ThymeleafViewResolver viewResolver(){
    ThymeleafViewResolver viewResolver = new ThymeleafViewResolver();
    viewResolver.setTemplateEngine(templateEngine());
    return viewResolver;
}
```

Creating PDF documents

```java
public class MyPdfView extends AbstractPdfView{
    @Override
    protected void buildPdfDocument(
            Map<String, Object> model,
            com.lowagie.text.Document document,
            com.lowagie.text.pdf.PdfWriter pdfWriter,
            HttpServletRequest httpServletRequest,
            HttpServletResponse httpServletResponse) throws Exception {
        document.add(new Paragraph((String)model.get("message")));
    }
}
```

Resolving PDF views

```xml
<bean id="pdfViewResolver"
      class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
    <property name="order" value="1"/>
    <property name="basename" value="views"/>
</bean>
```

views.properties

```properties
mypdf.(class)=dvdstore.views.MyPdfView
```

controller

```java
ModelAndView mav = new ModelAndView();
mav.setViewName("mypdf");
```

URI Templates

■ URLs can contain variable data
  – /products/{productId}
  – /products/{category}/all

■ Use parameter in request handling

```java
@RequestMapping("products/{productId}")
public ModelAndView productDetails(@PathVariable int productId) {
    ModelMap model = new ModelMap();
    model.addAttribute("productId", productId);
    return new ModelAndView("product", model);
}
```

Accessing request parameters

- /search?query=Spring

```java
@RequestMapping("search")
public void search(@RequestParam(required = true) String query) {
    System.out.println("Query: " + query);
}
```

Accessing request headers

Request Headers

Name	Value
Accept	text/html

```java
@RequestMapping("/example")
public ModelAndView example(@RequestHeader String accept) {
    System.out.println("Accept header: " + accept);
```

Accessing request cookies

```java
@RequestMapping("/example")
public ModelAndView example(@CookieValue String userId) {
    System.out.println("Cookie value: " + userId);
```

Request method arguments

Argument Type	Explanation
`@RequestBody`	The body of the request mapped by a HttpMessageConverter (for RESTful web services)
`HttpServletRequest / HttpServletResponse`	Plain Servlet API request and response
`HttpSession`	The Servlet API session object (think about thread safety)
`Locale`	The current request's locale
`InputStream / Reader`	Raw inputstream to access the request's content
`OutputStream / Write`	Raw outputstream to write response content
`Map / Model / ModelMap`	Implicit model that's exposed to the web view
`Command objects`	Command objects available in the model
`Errors / BindingResult`	Validation results for the preceding command object
`SessionStatus`	Status handle for marking form processing complete, triggering session cleanup

Request method return types

Return types	Explanation
`ModelAndView`	Combination of explicit model and view object
`Model / Map`	View name is resolved using the `RequestToViewNameTranslator`
`View`	Model resolved using implicit model command objects
`String`	View name. Model is resolved using implicit model command objects
`Void`	Model resolved using implicit model command objects. View name is resolved using the `RequestToViewNameTranslator`
`@ResponseBody`	Object converted using a HttpMessageConverter. Useful for RESTful web services

Showing a form

Handle only GET requests

```java
@RequestMapping(value = "books/edit", method = RequestMethod.GET)
public String editBook(Book book) {
    return "books/edit";
}
```

Create implicit 'book' command object

HTML Form

Form fields will be saved to

Property on

```html
<form:form commandName="book">
    <table>
        <tr>
            <td>Titel</td>
            <td><form:input path="title"/></td>
        </tr>
        <tr>
            <td colspan="2" align="right">
                <input type="submit" value="Save"/>
            </td>
        </tr>
    </table>
</form:form>
```

HTML Form

Form fields will be saved to

Property on

```html
<form  th:object="${book}" th:action="@{/book}" method="post">

    <input type="hidden" th:field="*{id}"/>

    <input type="text" class="form-control" th:field="*{title}"/>

    <button type="submit">Submit</button>

</form>
```

Submitting a form

All properties will be set on

```java
@RequestMapping(value = "books/edit", method = RequestMethod.POST)
public String saveBook(Book book) {
    bookCatalog.saveBook(book);
    return "redirect:/spring/books";
}
```

Redirect to overview page

Bean Validation

- Bean Validation API (JSR-303)
- Define field constraints on Java classes
- Integrates with other frameworks
  - e.g. JSF 2.0, Spring 3, JPA 2

Validation Example

- @Null
- @NotNull
- @AssertTrue
- @AssertFalse
- @Min
- @Max
- @DecimalMin
- @DecimalMax
- @Size
- @Digits
- @Past
- @Future
- @Pattern

```java
@Entity
public class Employee {
    @Id
    private Long id;

    @NotNull @Size(min = 2, max = 20)
    private String name;

    @Past
    private Date birthDate;

    @Min(value = 1000)
    private double salary;
```

Triggering validation

Trigger bean validation

Contains error information

```java
@RequestMapping(value = "employees/edit", method = RequestMethod.POST)
public String saveBook(@Valid Employee employee,
                       BindingResult result) {
    if (result.hasErrors()) {
        return "employee/addForm";
    } else {
        //Save employee
        System.out.println("Employee: " + employee.getFirstname());
        return "redirect:/spring/employees";
    }
}
```

Rendering validation errors

Check if there are errors

Render error for property

```html
<form  th:action="@{/}" th:object="${book}" method="post">

    <ul th:if="${#fields.hasErrors('*')}">
      <li th:each="err : ${#fields.errors('*')}" th:text="${err}">Input is incorrect</li>
    </ul>
      <table>
        <tr>
          <td>Title:</td>
          <td><input type="text" th:field="*{title}" /></td>
          <td th:if="${#fields.hasErrors('title')}" th:errors="*{title}">Title Error</td>
        </tr>
        <tr>
          <td><button type="submit">Submit</button></td>
        </tr>
      </table>
</form>
```

Custom bean validation invocation

```
validator.validate(Object instance, Class<?>...
groups)
```

```java
ValidatorFactory factory = Validation.buildDefaultValidatorFactory();
Validator validator = factory.getValidator();

Set<ConstraintViolation<Employee>> violations =
        validator.validate(emp1, Communicating.class);

for (ConstraintViolation<Employee> violation : violations) {
    System.out.println("Error: " + violation.getMessage());
}
```

Creating constraints

- Create annotation
- Implement validator class

```java
@ElementCollection
@NonEmptyCollection
private Set<String> emailAddresses;
```

```java
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD, ElementType.FIELD})
@Constraint(validatedBy = NonEmptyCollectionValidator.class)
public @interface NonEmptyCollection {
    String message() default "Collection may not be empty";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}
```

Creating constraints

```java
public class NonEmptyCollectionValidator implements
        ConstraintValidator<NonEmptyCollection, Collection<?>> {
    @Override
    public void initialize(NonEmptyCollection nonEmptyCollection) {

    }


    @Override
    public boolean isValid(Collection<?> objects,
                           ConstraintValidatorContext
                                   constraintValidatorContext) {
        return objects != null && objects.size() > 0;
    }
}
```

Validation Groups

- Define different sets of constraints for different situations
- Trigger validation only for a certain constraints

```java
@ElementCollection
@NonEmptyCollection(groups = Communicating.class)
private Set<String> emailAddresses;
```

```java
public interface Communicating {
}
```

```java
validator.validate(emp1, Communicating.class);
```

ModelAttribute

Method to produce model

```java
@ModelAttribute("roles")
public List<String> listRoles() {
    return Arrays.asList("Developer", "Sales", "Management");
}
```

Handling exceptions

Map exceptions to custom error pages

```xml
<bean class="....SimpleMappingExceptionResolver">
    <property name="defaultErrorView" value="error"/>
    <property name="exceptionMappings">
        <util:map>
            <entry key="dvdstore.controllers.MyCustomException"
                   value="customerror"/>
        </util:map>
    </property>
</bean>
```

Handling exceptions

Handle exceptions from code

```java
@Component
public class CustomErrorResolver
        implements HandlerExceptionResolver {
    @Override
    public ModelAndView resolveException(HttpServletRequest request,
                                         HttpServletResponse response,
                                         Object handler,
                                         Exception ex) {
        if (ex instanceof MyCustomException) {
            return new ModelAndView("customerror");
        } else {
            return new ModelAndView("error");
        }
    }
}
```

Interceptors

- Intercept requests before handling
- Similar to Servlet Filters but within the Spring context
- Implement HandlerInterceptor interface
  - or extend HandlerInterceptorAdaptor convenience class

Interceptor example

```java
public class TimeBasedAccessInterceptor extends
        HandlerInterceptorAdapter {

    @Override
    public boolean preHandle(HttpServletRequest request,
                             HttpServletResponse response,
                             Object handler) throws Exception {
        //implement intercepter logic here
        if (1 > 2) {
            return true;
        } else {
            response.setStatus(403);
            return false;
        }

    }
}
```

Configuring interceptors

```xml
<mvc:interceptors>
    <bean class="....TimeBasedAccessInterceptor"/>

    <bean class="....ThemeChangeInterceptor">
        <property name="paramName" value="theme"/>
    </bean>

    <bean class="....LocaleChangeInterceptor">
        <property name="paramName" value="lang"/>
    </bean>
</mvc:interceptors>
```

Intermezzo Thymeleaf

- Thymeleaf is a Java library
- An XML/XHTML/HTML5 template engine
- Transforms template files to display data / text produced applications
- Most of the processors of the Standard Dialect are attribute processors
  - Allows browsers to correctly display HTML5 template files before processing because they will simply ignore the additional attributes

Thymeleaf details

```
<input type="text" name="title" value="James Carrot"
                th:value="${book.title}" />
```

■ Allows to (optionally) specify a value attribute, here "James Carrot", that will be displayed when statically opened

■ It is substituted by the value resulting from the evaluation of ${book.title} during Thymeleaf processing of the template