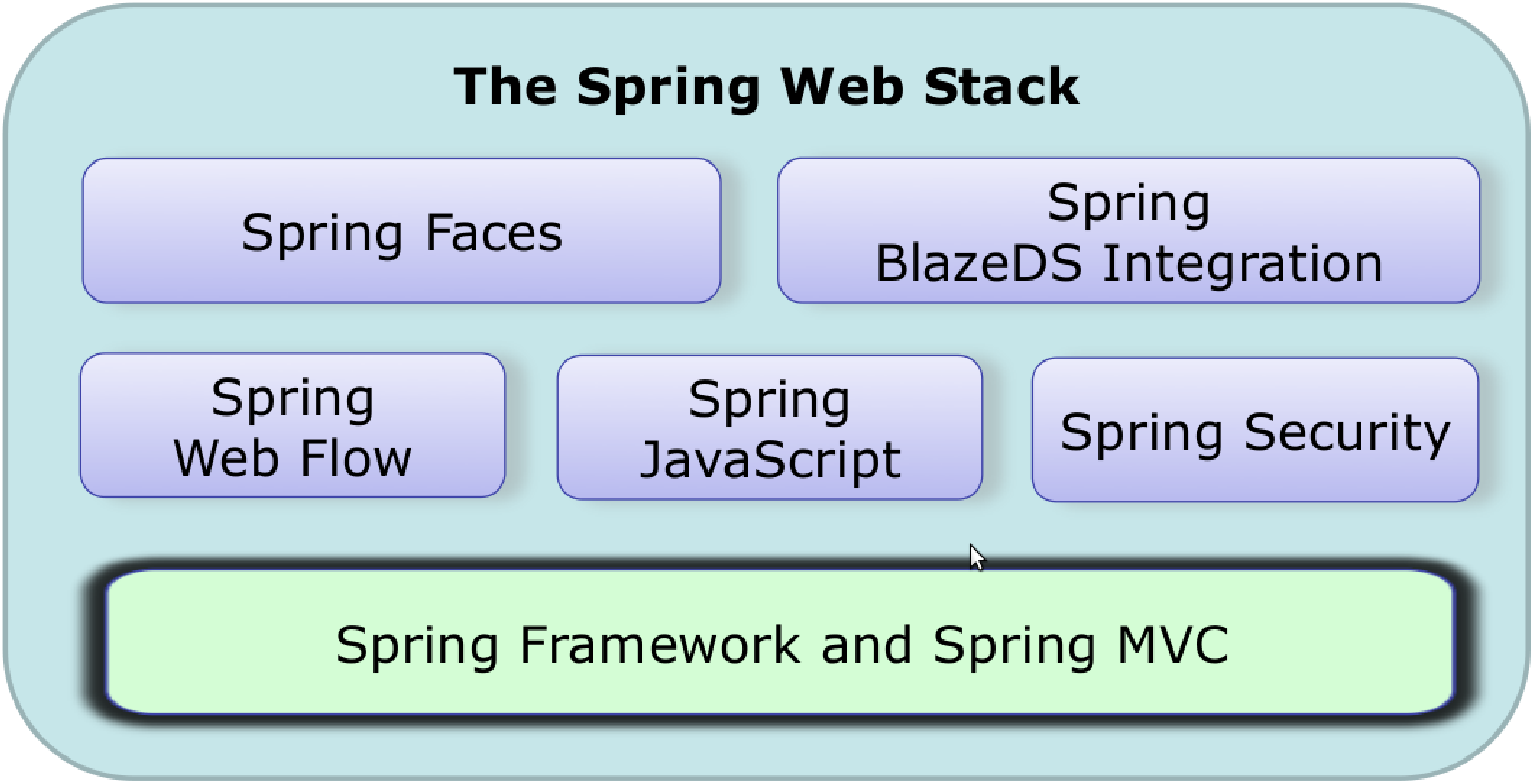
Spring-Hibernate Training Program
Session 3

BoTree Consulting

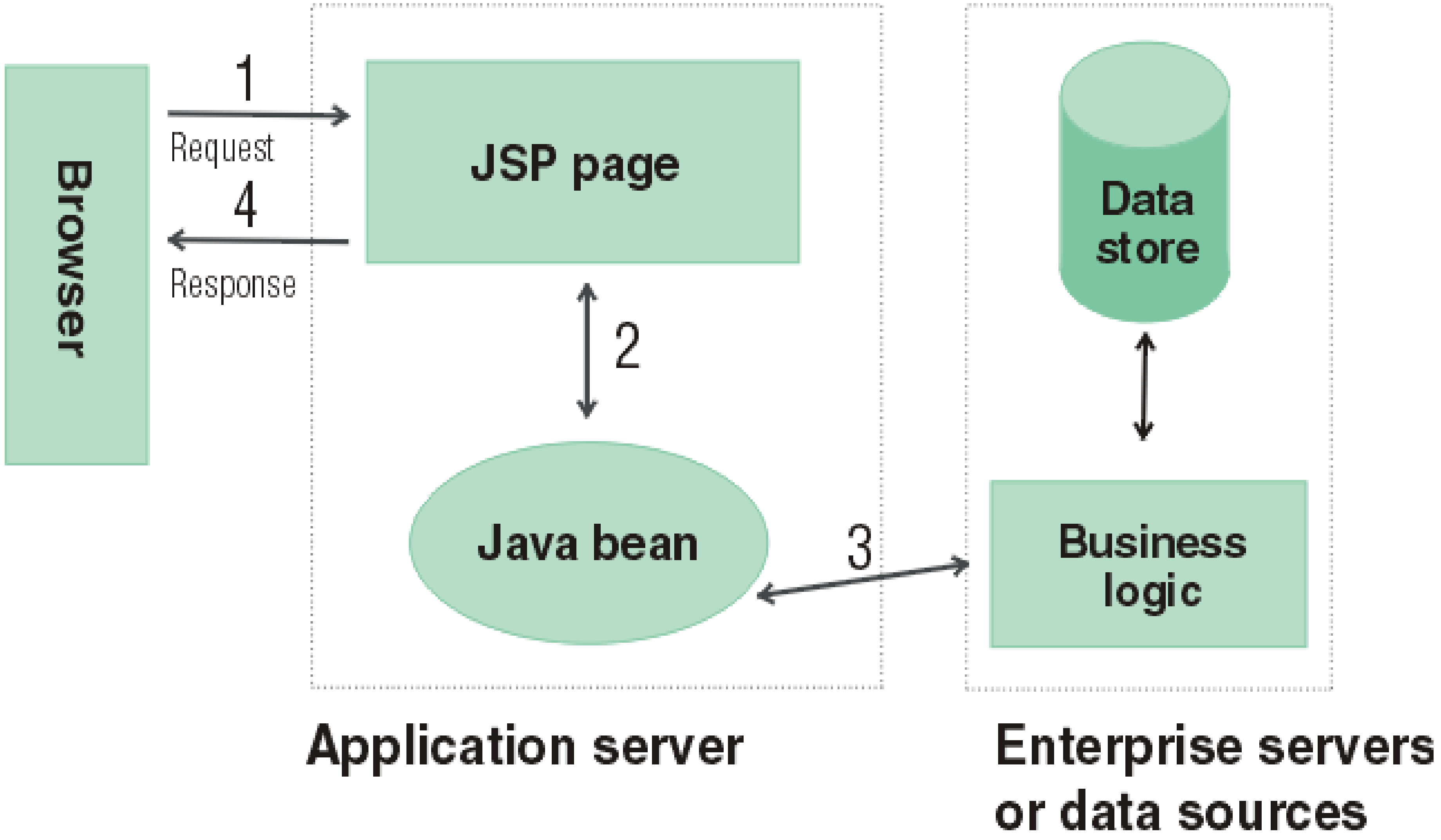
Spring Web Stack



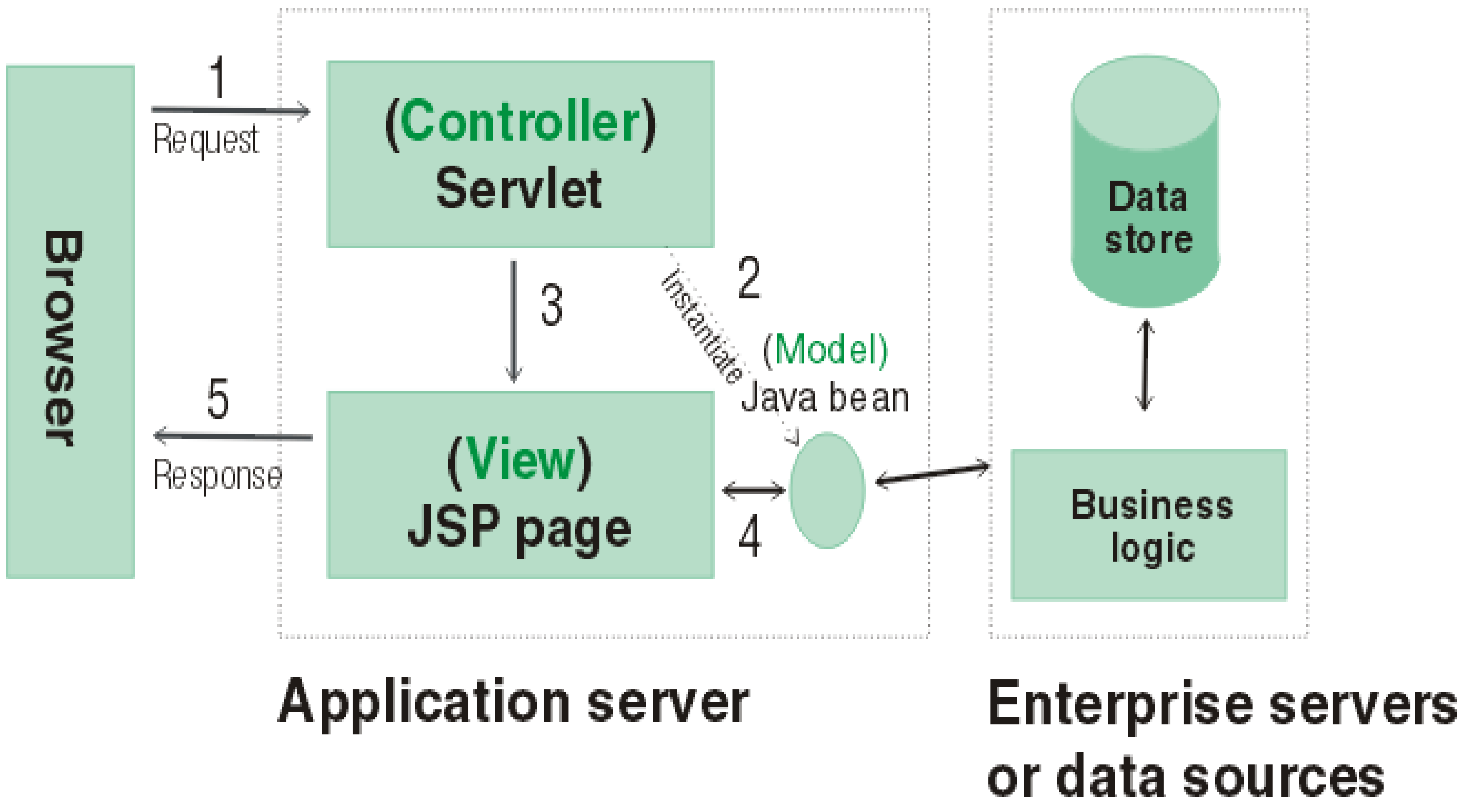
What is MVC?

- **Model** - Represents enterprise data and the business rules that govern access to and updates of this data.
- **View** - The view renders the contents of a model. It accesses enterprise data through the model and specifies how that data should be presented. It is the view's responsibility to maintain consistency in its presentation when the model changes.
- **Controller** - The controller translates interactions with the view into actions to be performed by the model. In a Web application, they appear as GET and POST HTTP requests.

MVC 1


Browser
1
Request
4
Response
JSP page
2
Java bean
3
Data store
Business logic
Application server
Enterprise servers or data sources

MVC 2


Browser
1
Request
(Controller)
Servlet
3
2
Instantiate
(Model)
Java bean
5
Response
(View)
JSP page
4
Data store
Business logic
Application server
Enterprise servers or data sources

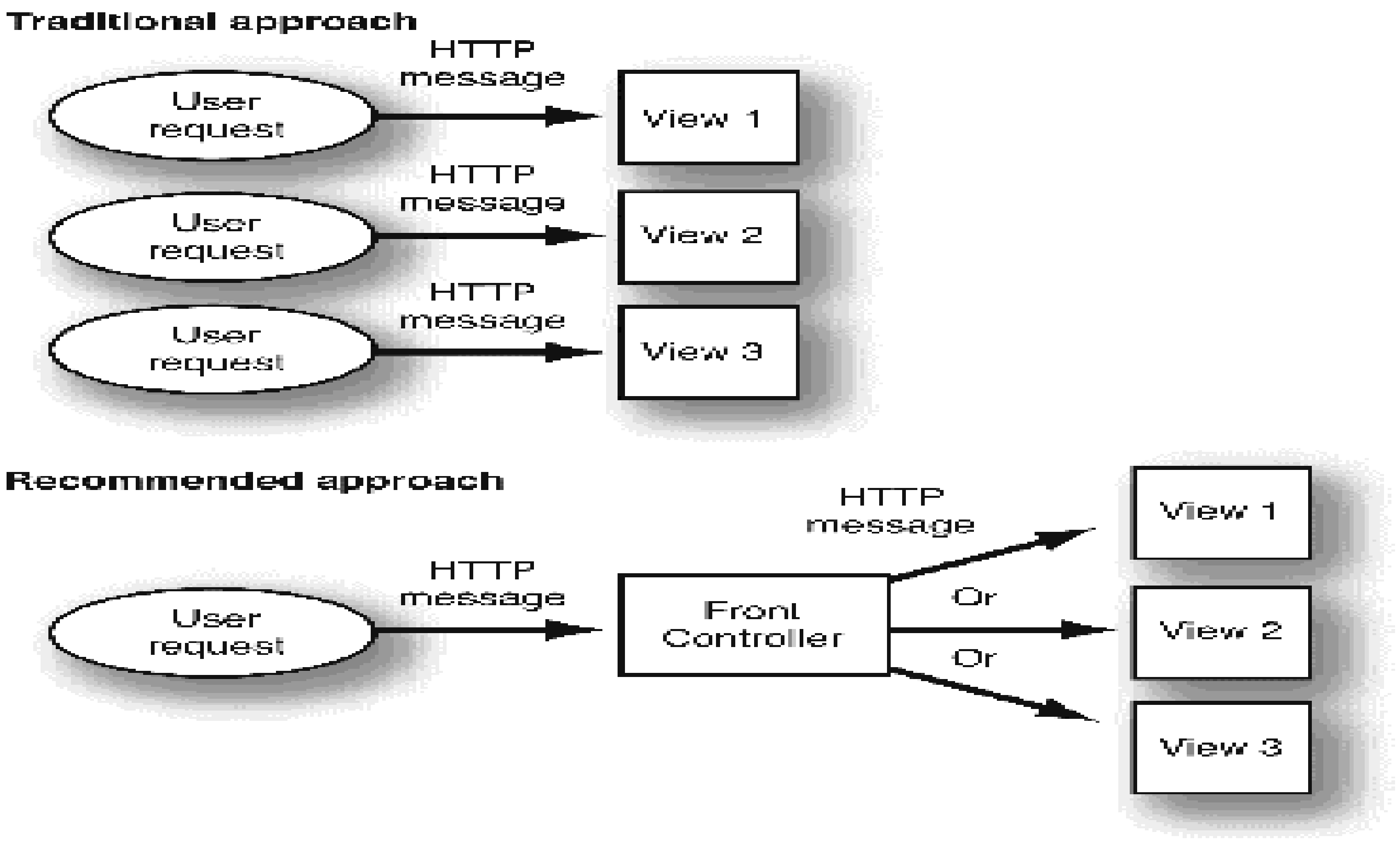
Why MVC?

- Reduce Coupling
- Flexibility and Maintainability
- Testability
- Productivity

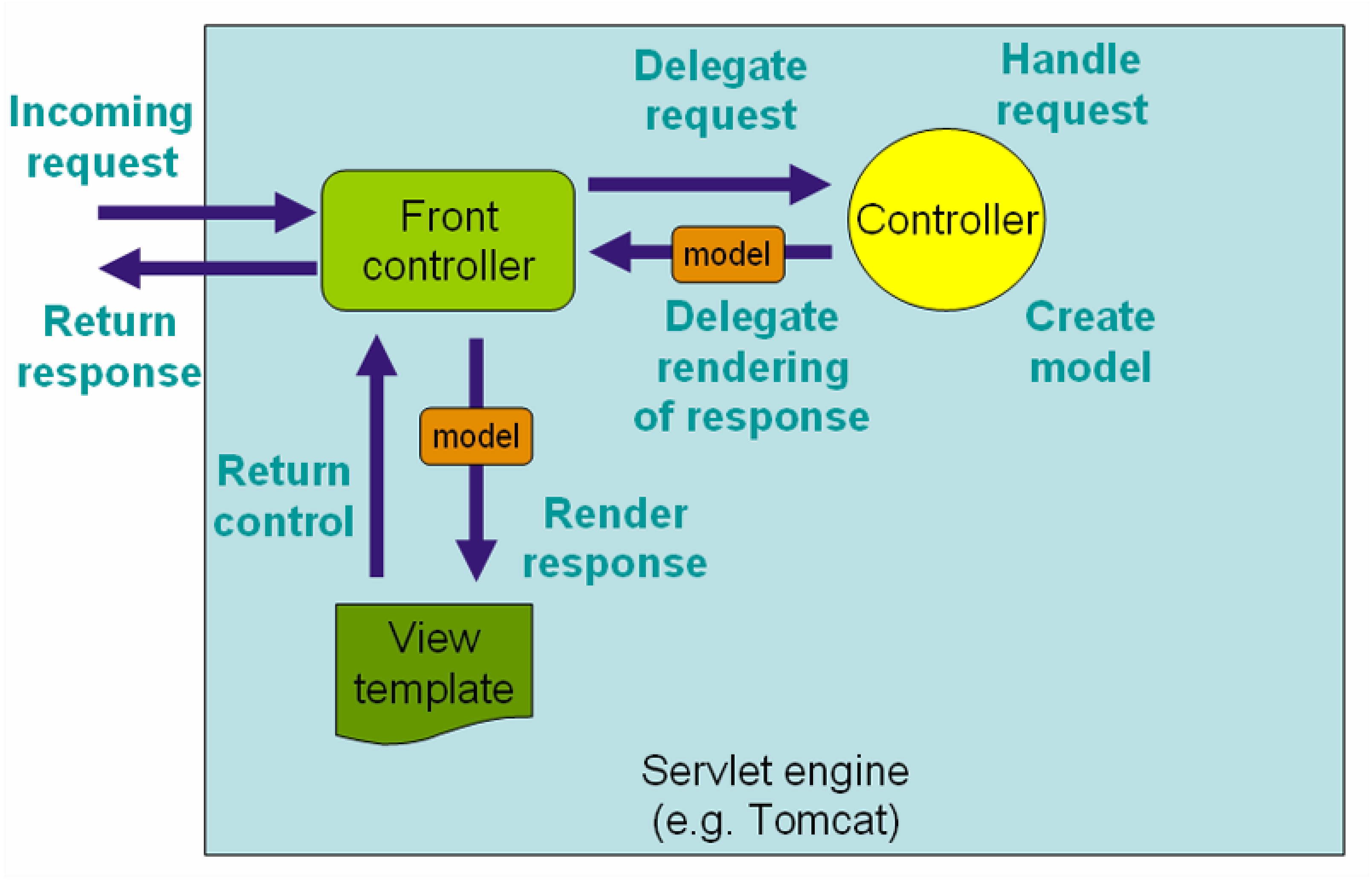
Spring MVC Overview

- Front Controller
- Controller
- Model
- View

Front Controller

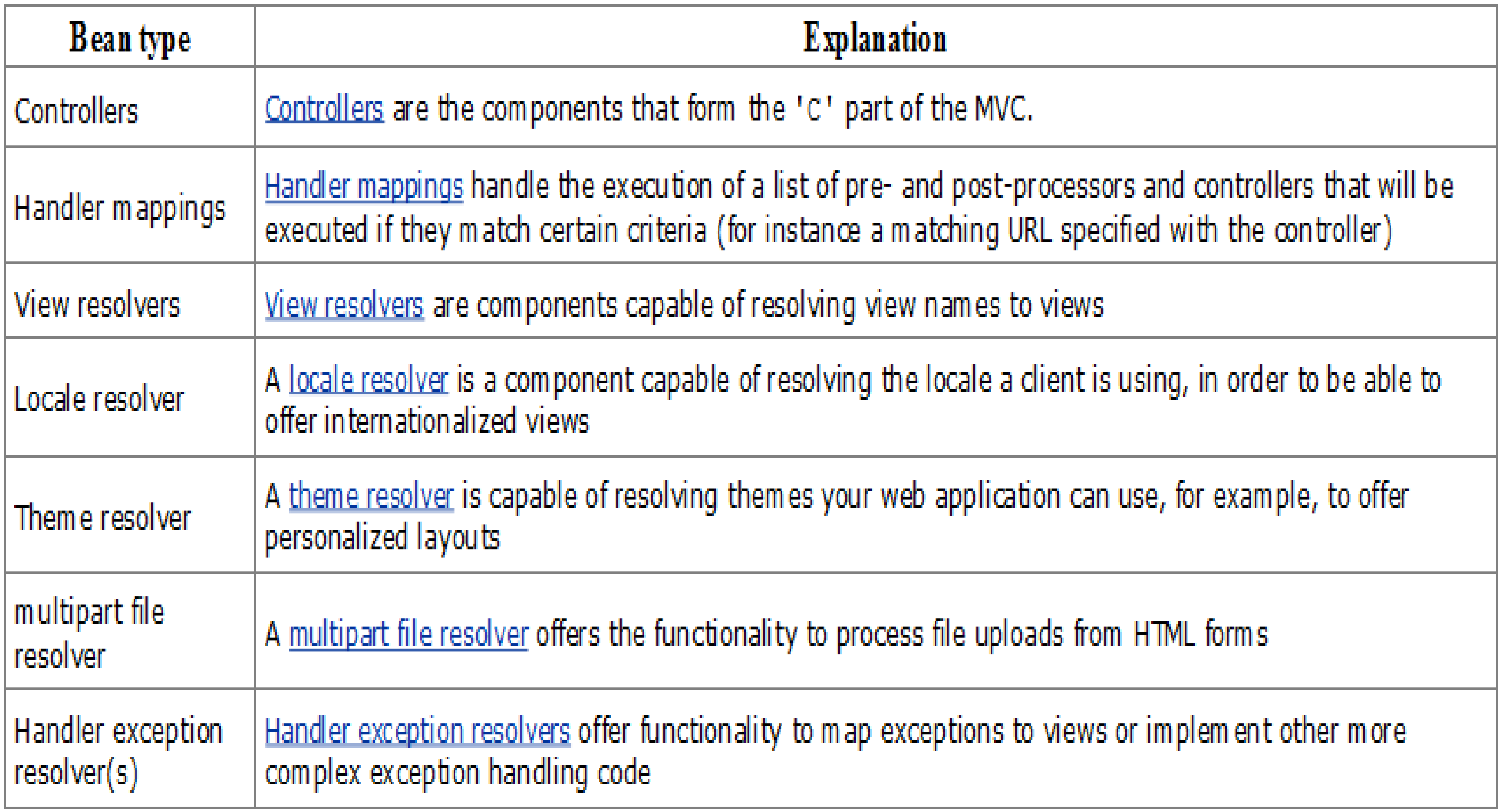


Spring MVC



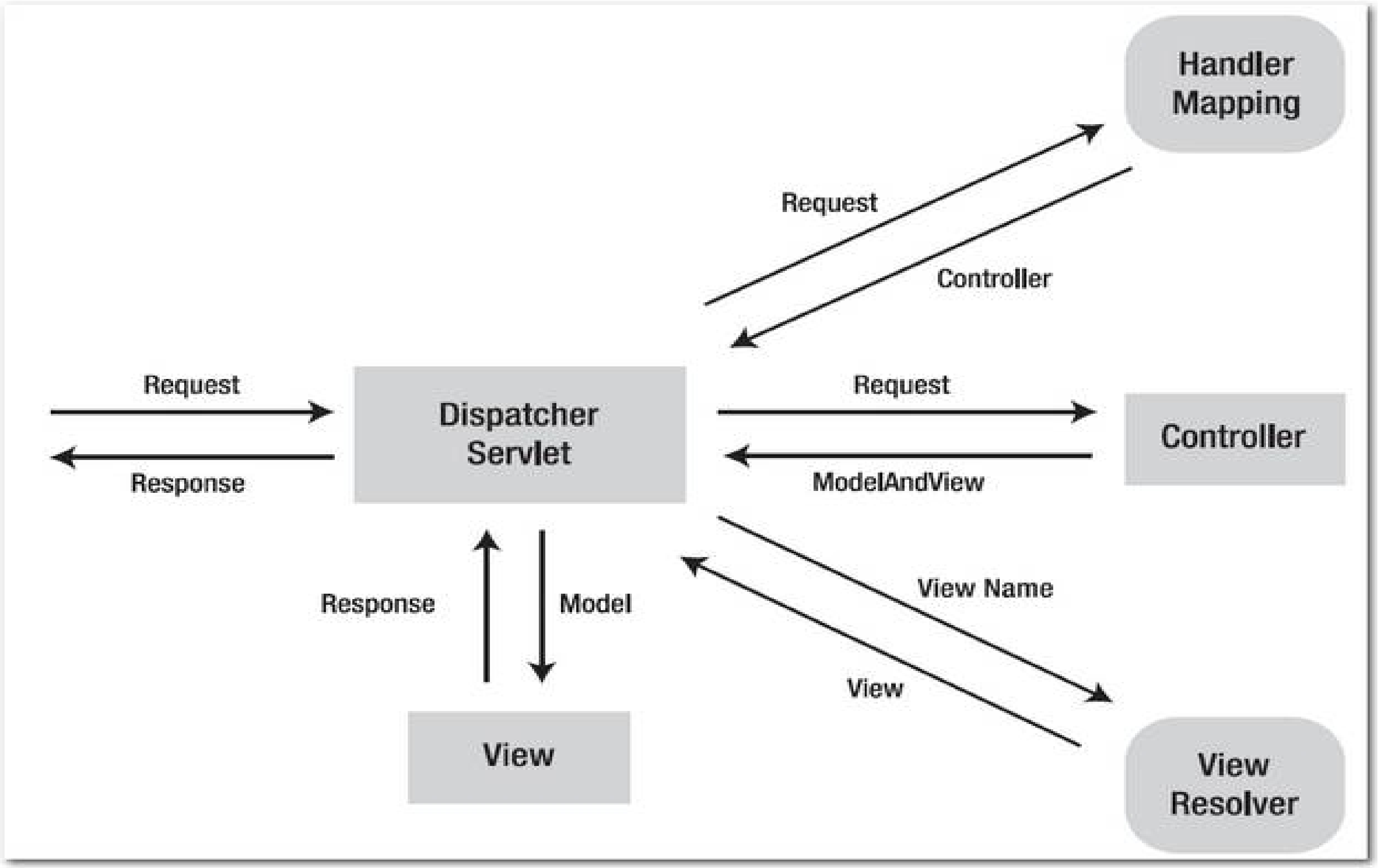
Incoming request
Return response
Front controller
Delegate request
Handle request
Controller
model
Delegate rendering of response
Create model
model
Return control
Render response
View template
Servlet engine (e.g. Tomcat)

SpecialBeans in WebApplicationContext



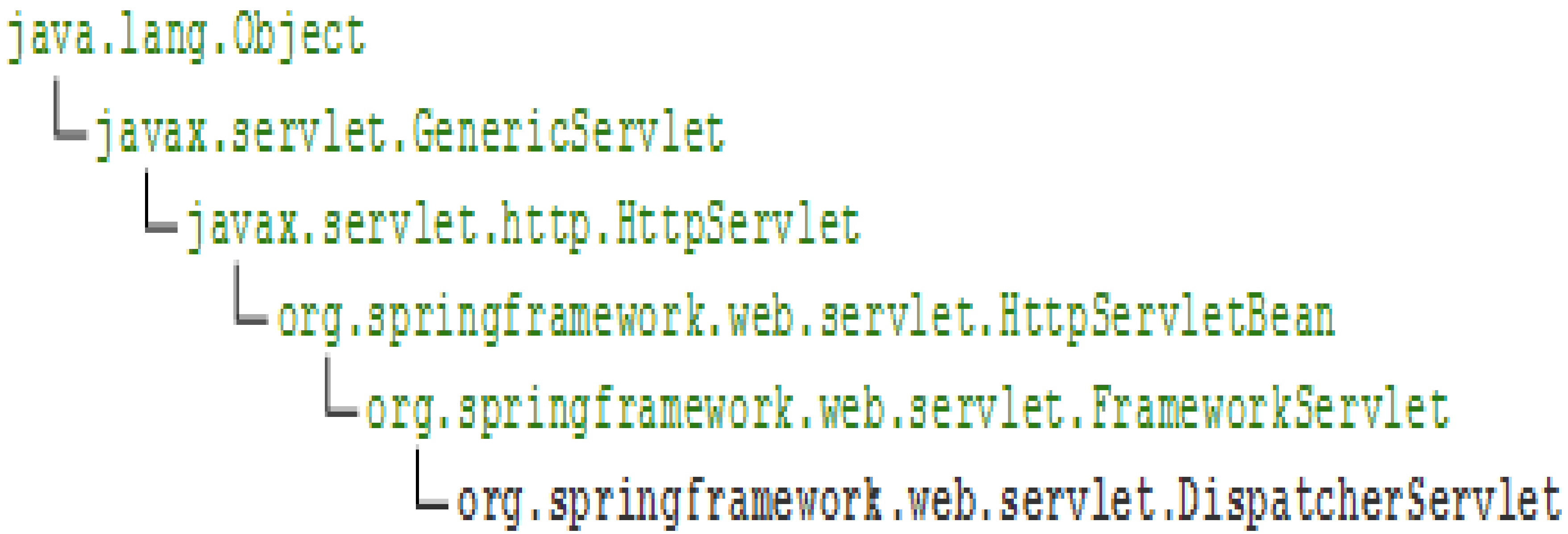
Bean type	Explanation
Controllers	Controllers are the components that form the 'C' part of the MVC.
Handler mappings	Handler mappings handle the execution of a list of pre- and post-processors and controllers that will be executed if they match certain criteria (for instance a matching URL specified with the controller)
View resolvers	View resolvers are components capable of resolving view names to views
Locale resolver	A locale resolver is a component capable of resolving the locale a client is using, in order to be able to offer internationalized views
Theme resolver	A theme resolver is capable of resolving themes your web application can use, for example, to offer personalized layouts
multipart file resolver	A multipart file resolver offers the functionality to process file uploads from HTML forms
Handler exception resolver(s)	Handler exception resolvers offer functionality to map exceptions to views or implement other more complex exception handling code

Spring MVC


Handler Mapping
Request
Controller
Request
Response
Dispatcher Servlet
Request
ModelAndView
Controller
View Name
View
Response
Model
View
View Resolver

Dispatcher Servlet

- DispatcherServlet is an expression of the “FrontController” Design Pattern.



```
java.lang.Object
  └javax.servlet.GenericServlet
      └javax.servlet.http.HttpServlet
          └org.springframework.web.servlet.HttpServletBean
              └org.springframework.web.servlet.FrameworkServlet
                  └org.springframework.web.servlet.DispatcherServlet
```

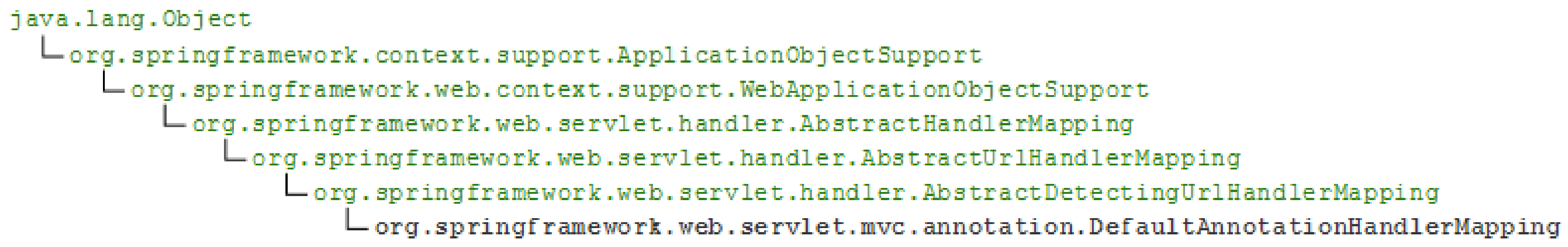
DispatcherServlet Mapping

```
<web-app>
  <servlet>
    <servlet-name>example</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>example</servlet-name>
    <url-pattern>*.form</url-pattern>
  </servlet-mapping>
</web-app>
```

```
<web-app>
  <servlet>
    <servlet-name>example</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>example</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

Mapping Handler

- Objects that define a mapping between requests and handler objects

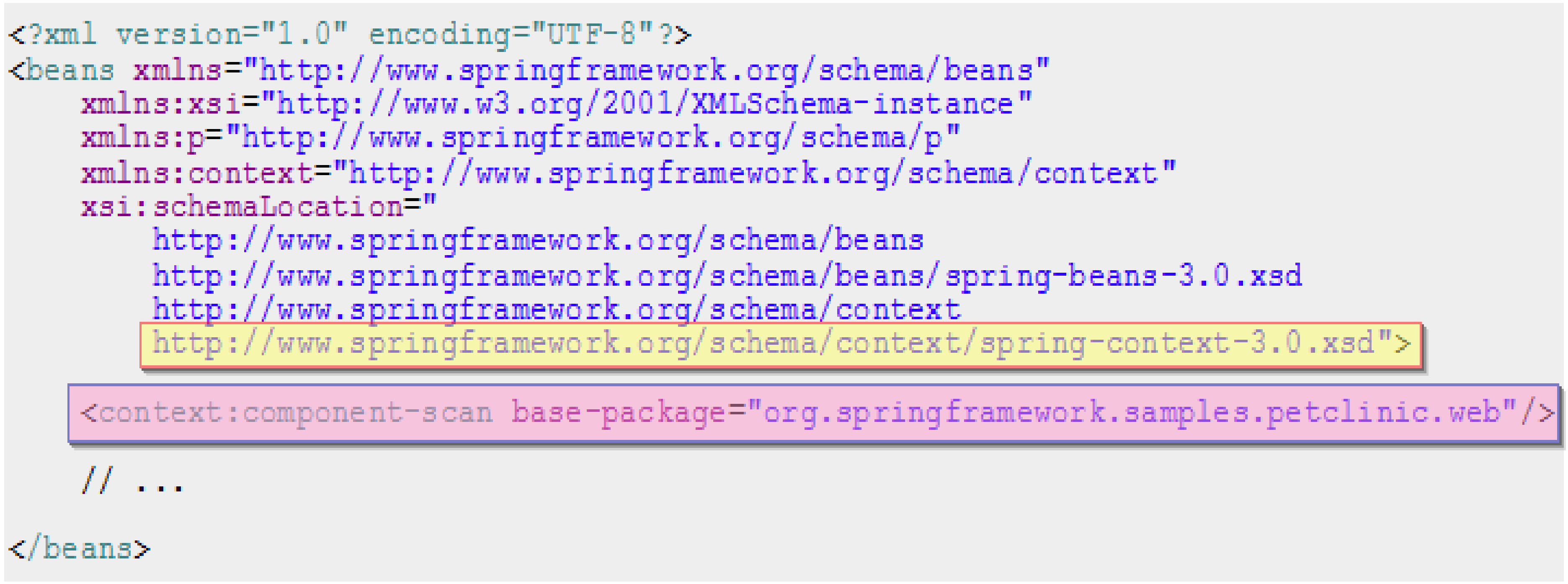


```
java.lang.Object
  └org.springframework.context.support.ApplicationObjectSupport
      └org.springframework.web.context.support.WebApplicationObjectSupport
          └org.springframework.web.servlet.handler.AbstractHandlerMapping
              └org.springframework.web.servlet.handler.AbstractUrlHandlerMapping
                  └org.springframework.web.servlet.handler.AbstractDetectingUrlHandlerMapping
                      └org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping
```

- The *AnnotationMethodHandlerAdapter* is responsible for processing @RequestMapping annotated handler methods
- *SimpleControllerHandlerAdapter* is responsible for processing Type(Controller) level @RequestMapping

Controller

- The *@Controller* annotation indicates that a particular class serves the role of a Controller.
- For configuring Controllers detection of annotations,



```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <context:component-scan base-package="org.springframework.samples.petclinic.web"/>

    // ...

</beans>
```

Controller

- Controllers can be viewed as interfaces for providing access to the the services offered by the application.
- If we consider the UI(HTML) as the boundary of an application View Layer, the Controller is the other end.
- Controllers do not contain any business logic.
- Controllers are responsible for handling the Request and Response objects.
- Controllers are also responsible for forwarding the response to specific views.

Mapping Requests

- @RequestMapping annotation
  - Define mapping rules
- Different Levels
  - Class Level.
  - Method Level.

@RequestMapping Rules

- By path [Primary Mapping]

@RequestMapping(“<PATH>”)

- By HTTP method

@RequestMapping(“<PATH>”, method=RequestMethod.GET)

- By presence of parameter

@RequestMapping(“<PATH>”, method=RequestMethod.GET, params=”foo”)

  - Negation also supported: params={ “foo”, “!bar” })

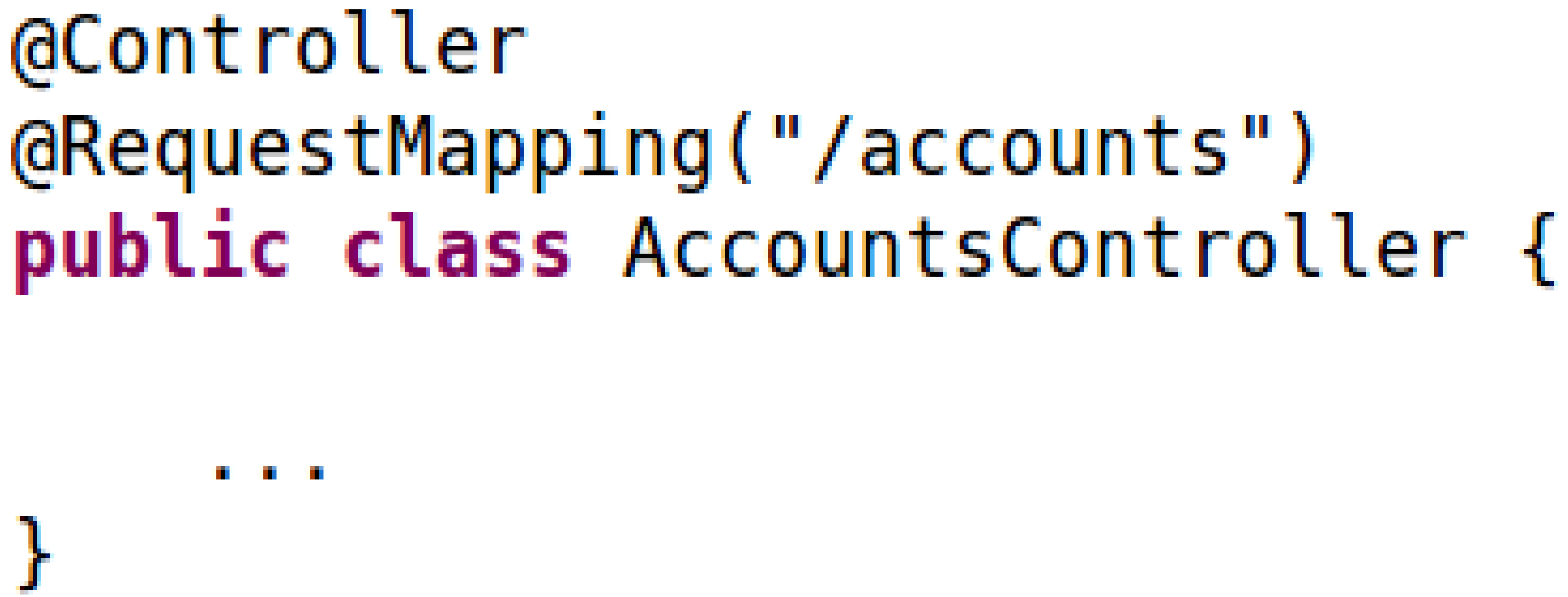
- By presence of request header

@RequestMapping(“<PATH>”, header=”content-type=text/*”)

  - Negation also supported

Mapping – Class Level

- Optional
- To group related actions in single controller
- Concise way to map all requests within a path to a @Controller



```java
@Controller
@RequestMapping("/accounts")
public class AccountsController {


    ...
}
```

Mapping – Method Level

- **Absolute** if Class Level mapping is not specified otherwise **Relative**

```
@Controller
@RequestMapping("/accounts")
public class AccountsController {

    @RequestMapping("active")
    public @ResponseBody List<Account> active() { ... }

    @RequestMapping("inactive")
    public @ResponseBody List<Account> inactive() { ... }
}
```

Flexible Handler Method Signature

- Method Parameters
  - Standard Objects
    - ServletRequest / HttpServletRequest
    - ServletResponse / HttpServletResponse
    - HttpSession
    - InputStream / Reader
    - OutputStream / Writer
    - Locale
  - Special Objects
    - @RequestParam / @PathParam / @RequestHeader / @CookieValue annotated parameters
    - @ModelAttribute annotated command / form objects
    - Map / Model / ModelMap
    - WebRequest or NativeWebRequest.
    - Errors / BindingResult
    - SessionStatus

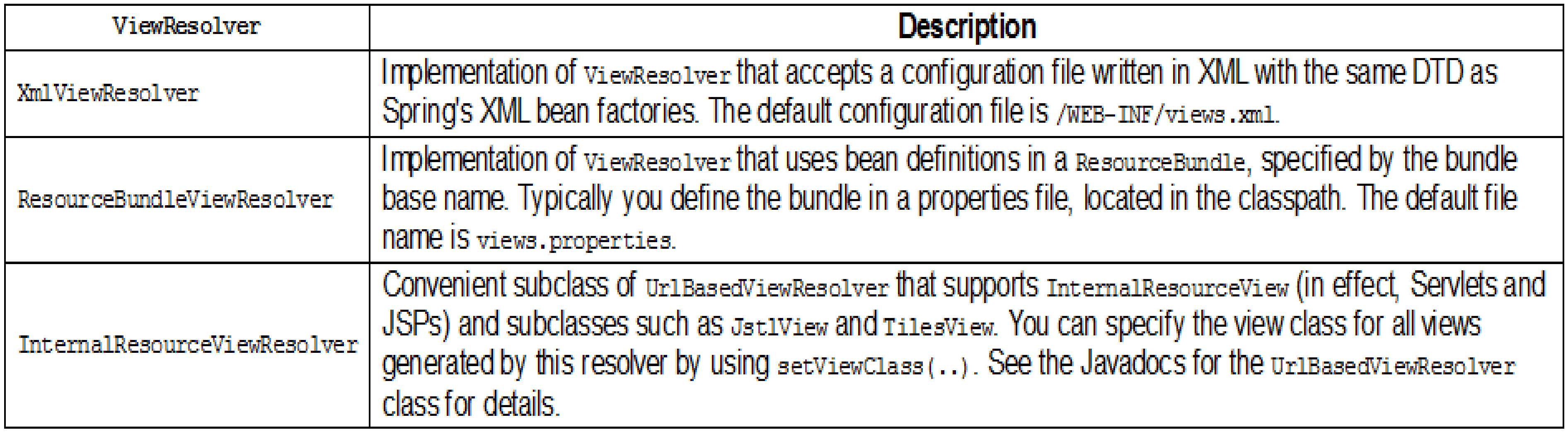
Flexible Handler Method Signature

- Method Return types
  - Model
  - Map
  - View
  - String
  - Void
  - Custom Class

View Resolvers

- Decouples View Technology
- Interfaces ViewResolver and View
- ViewResolver Interface provides a mapping between logical view names and actual views.
- View Interface addresses the preparation of request and hands over the request to one of the view technologies.

View Resolvers



ViewResolver	Description
`XmlViewResolver`	Implementation of `ViewResolver` that accepts a configuration file written in XML with the same DTD as Spring's XML bean factories. The default configuration file is `/WEB-INF/views.xml`.
`ResourceBundleViewResolver`	Implementation of `ViewResolver` that uses bean definitions in a `ResourceBundle`, specified by the bundle base name. Typically you define the bundle in a properties file, located in the classpath. The default file name is `views.properties`.
`InternalResourceViewResolver`	Convenient subclass of `UrlBasedViewResolver` that supports `InternalResourceView` (in effect, Servlets and JSPs) and subclasses such as `JstlView` and `TilesView`. You can specify the view class for all views generated by this resolver by using `setViewClass(..)`. See the Javadocs for the `UrlBasedViewResolver` class for details.

InternalResourceViewResolver

```
<bean id="viewResolver"
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">

        <property name="prefix">
            <value>/WEB-INF/views/</value>
        </property>
        <property name="suffix">
            <value>.jsp</value>
        </property>
   </bean>
```

XmlViewResolver

```xml
<bean class="org.springframework.web.servlet.view.XmlViewResolver">
    <property name="location">
        <value>/WEB-INF/xml-views.xml</value>
    </property>
</bean>
```

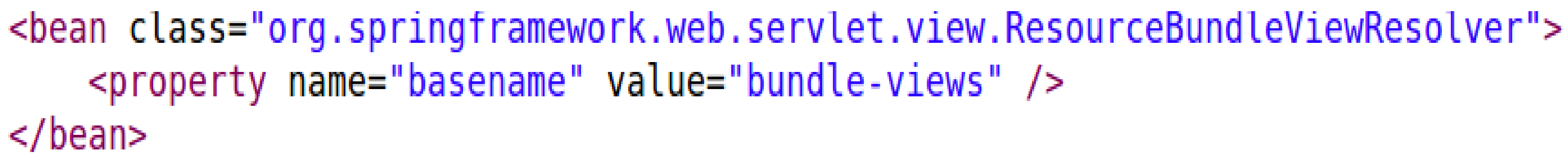
bundle-views.xml

```xml
<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-3.0.xsd">

    <bean id="WelcomePage"
        class="org.springframework.web.servlet.view.JstlView">
        <property name="url" value="/WEB-INF/views/WelcomePage.jsp" />
    </bean>

</beans>
```

ResourceBundleResolver



```xml
<bean class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
    <property name="basename" value="bundle-views" />
</bean>
```

bundle-views.properties

```
WelcomePage.(class)=org.springframework.web.servlet.view.JstlView
WelcomePage.url=/WEB-INF/views/WelcomePage.jsp
```

Chaining Resolvers

- Multiple View Resolvers using *order*
- Override specific view resolvers in certain conditions
- Example

Chaining Resolvers

```
<bean class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
    <property name="basename" value="bundle-views" />
    <property name="order" value="0" />

</bean>

<bean class="org.springframework.web.servlet.view.XmlViewResolver">
    <property name="location">
        <value>/WEB-INF/xml-views.xml</value>
    </property>
    <property name="order" value="1" />
</bean>

<bean id="viewResolver"
       class="org.springframework.web.servlet.view.InternalResourceViewResolver">

       <property name="prefix">
           <value>/WEB-INF/views/</value>
       </property>
       <property name="suffix">
           <value>.jsp</value>
       </property>
    <property name="order" value="2" />
  </bean>
```

Redirecting

- Used for HTTP redirect back to client before the view is rendered.
- Used to avoid duplicate submit.
- RedirectView
- redirect:
- forward:

End of
Spring – Hibernate Training Program
Session 3