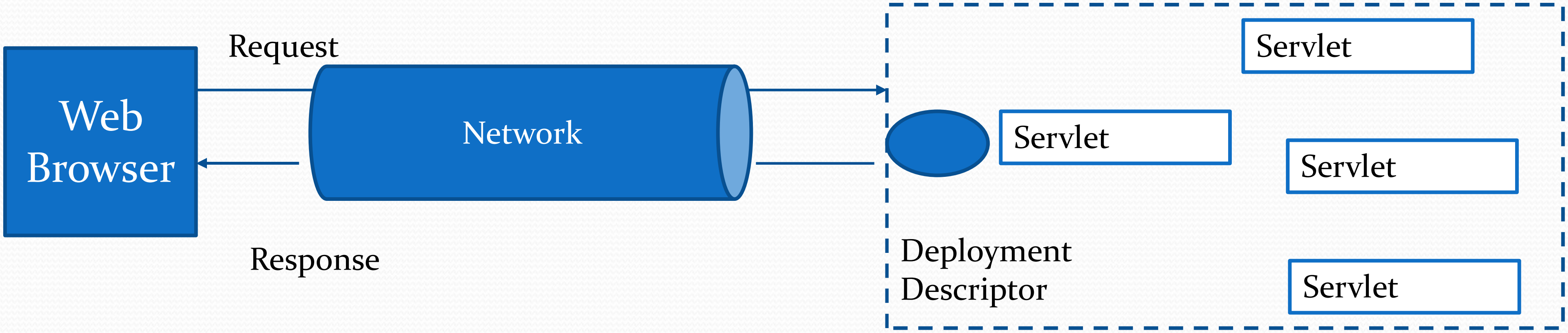
Web Frameworks: Spring

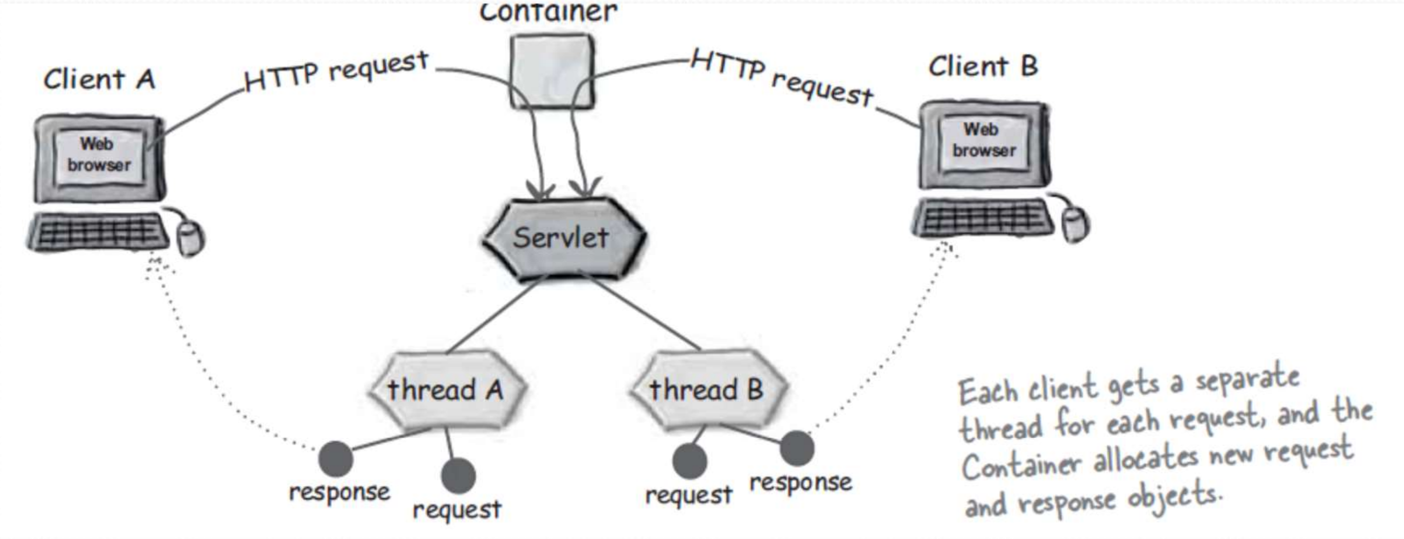
An Introduction

Web Container

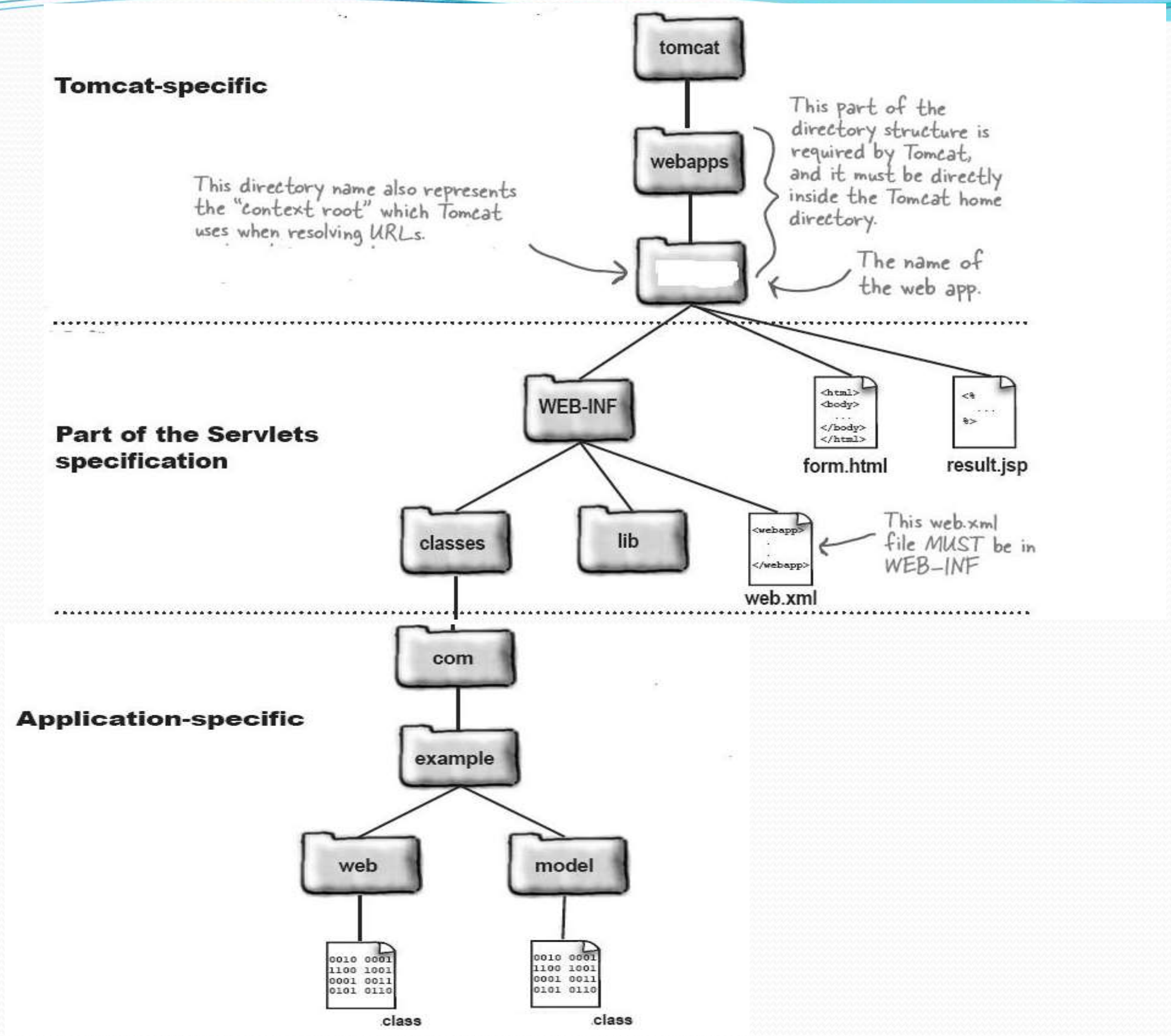
```
@WebServlet("/SelectCoffeeMVC")
public class CoffeeSelectMVC extends HttpServlet {

       public void doPost(HttpServletRequest request,
HttpServletResponse response)
           throws IOException, ServletException {
```



Handling Multiple Clients



Each client gets a separate thread for each request, and the Container allocates new request and response objects.



Tomcat-specific

Part of the Servlets specification

Application-specific

This part of the directory structure is required by Tomcat, and it must be directly inside the Tomcat home directory.

This directory name also represents the "context root" which Tomcat uses when resolving URLs.

The name of the web app.

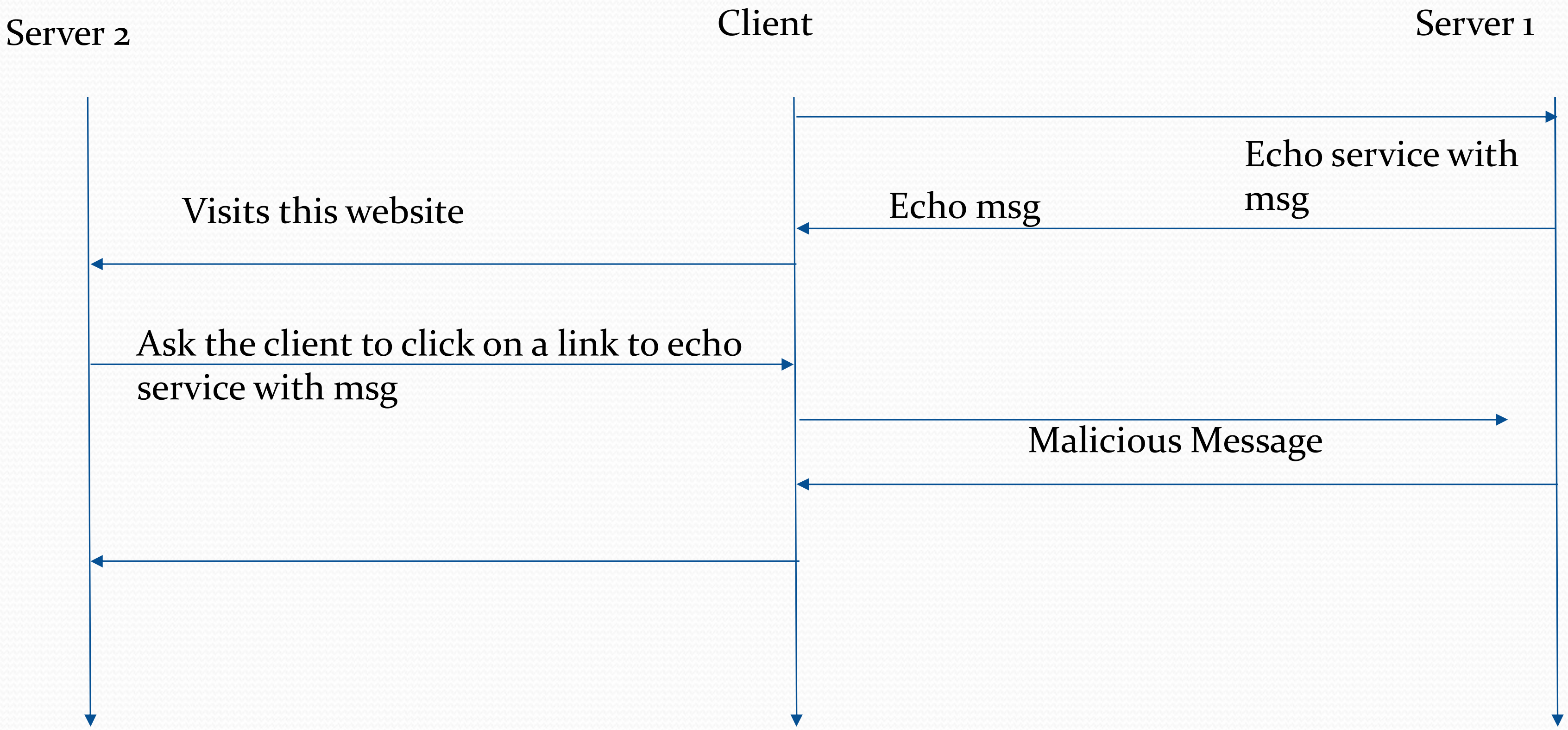
This web.xml file MUST be in WEB-INF

```
@WebServlet("/SelectCoffeeMVC.do")
public class CoffeeSelectMVC extends HttpServlet {

        public void doPost(HttpServletRequest request, HttpServletResponse response)
      throws IOException, ServletException {

                String color = request.getParameter("color");
                String addOn=request.getParameter("addOns");
                if(!color.equals(“”) && ! addOn.equals(“”)) {
                        Coffee c=new Coffee(color, addOn); }

                Cookie ck1;   HttpSession session=request.getSession();

                CoffeeExpert ce = new CoffeeExpert();
                 String result="";
        try{
                Connection con=(Connection)getServletContext().getAttribute("key2");
                result = ce.getBrands(c,con);
        }catch(Exception e){ System.out.println(e);}

                 request.setAttribute("brands", result);
                RequestDispatcher view =    request.getRequestDispatcher("result.jsp");
                view.forward(request, response);
  }}
```

Introduction

- Web.xml routes requests to the individual servlet’s doGet or doPost methods
- doGet(…)
  - //extract parameters from request

Injection Attack



Introduction

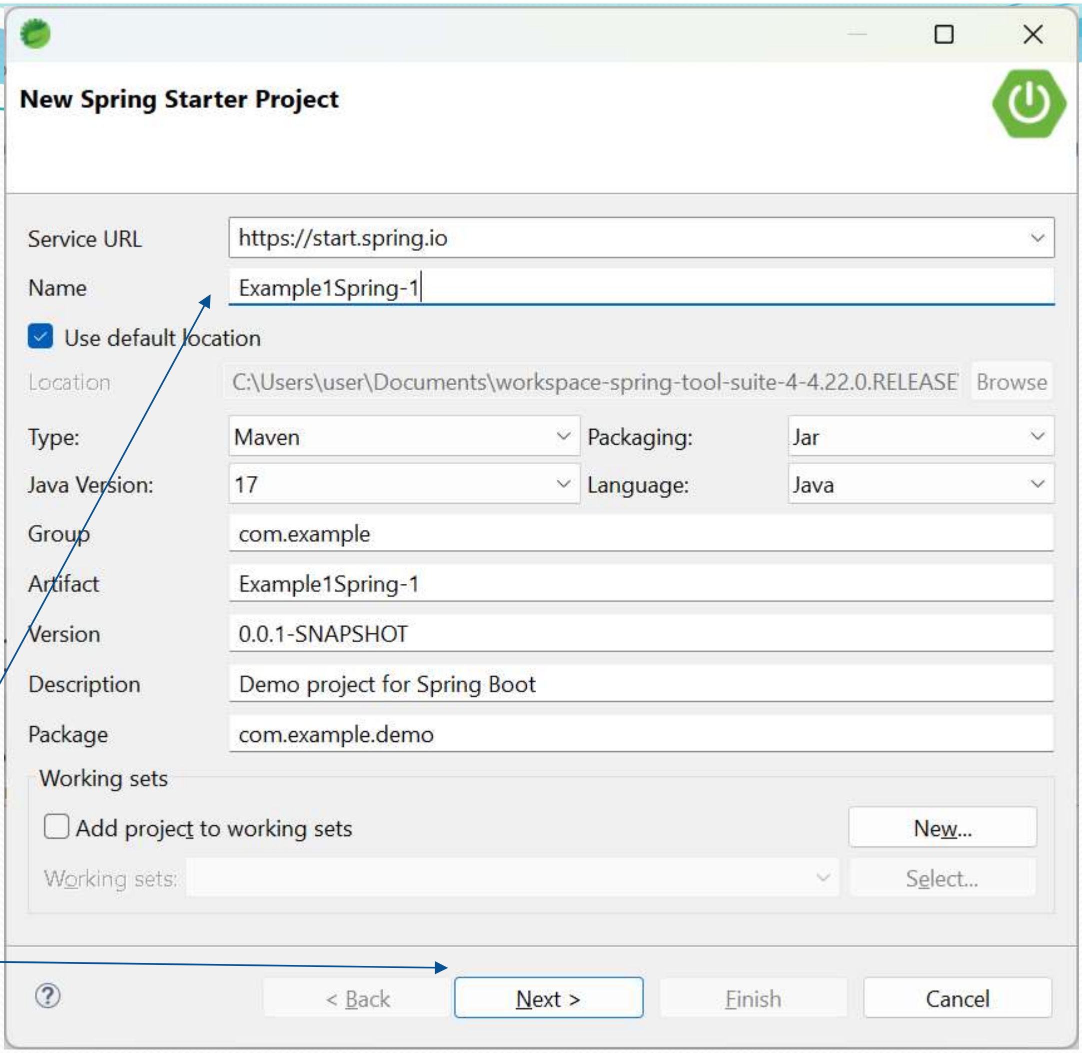
- Web.xml routes requests to the individual servlet’s doGet or doPost methods
- doGet(…)
  - //extract parameters from request
  - //validation
  - //Construct objects with parameters
  - //do the processing

Introduction

- In EJB *public class HelloWorldBean implements SessionBean {*
- Spring avoids (as much as possible) littering your application code with its API
- Spring almost never forces you to implement a Spring-specific interface or extend a Spring-specific class
- Instead, the classes in a Spring-based application often have no indication that they're being used by Spring
- Spring has enabled the return of the plain old Java object (POJO) to enterprise development

Setting up

- Download appropriate STS version from https://spring.io/tools
- Install the jar
- Open STS from the installed folder
- Top left corner→ click on “New”→Spring Starter Project



New Spring Starter Project

Service URL: https://start.spring.io

Name: Example1Spring-1

☑ Use default location

Location: C:\Users\user\Documents\workspace-spring-tool-suite-4-4.22.0.RELEASE

Type: Maven

Packaging: Jar

Java Version: 17

Language: Java

Group: com.example

Artifact: Example1Spring-1

Version: 0.0.1-SNAPSHOT

Description: Demo project for Spring Boot

Package: com.example.demo

Working sets

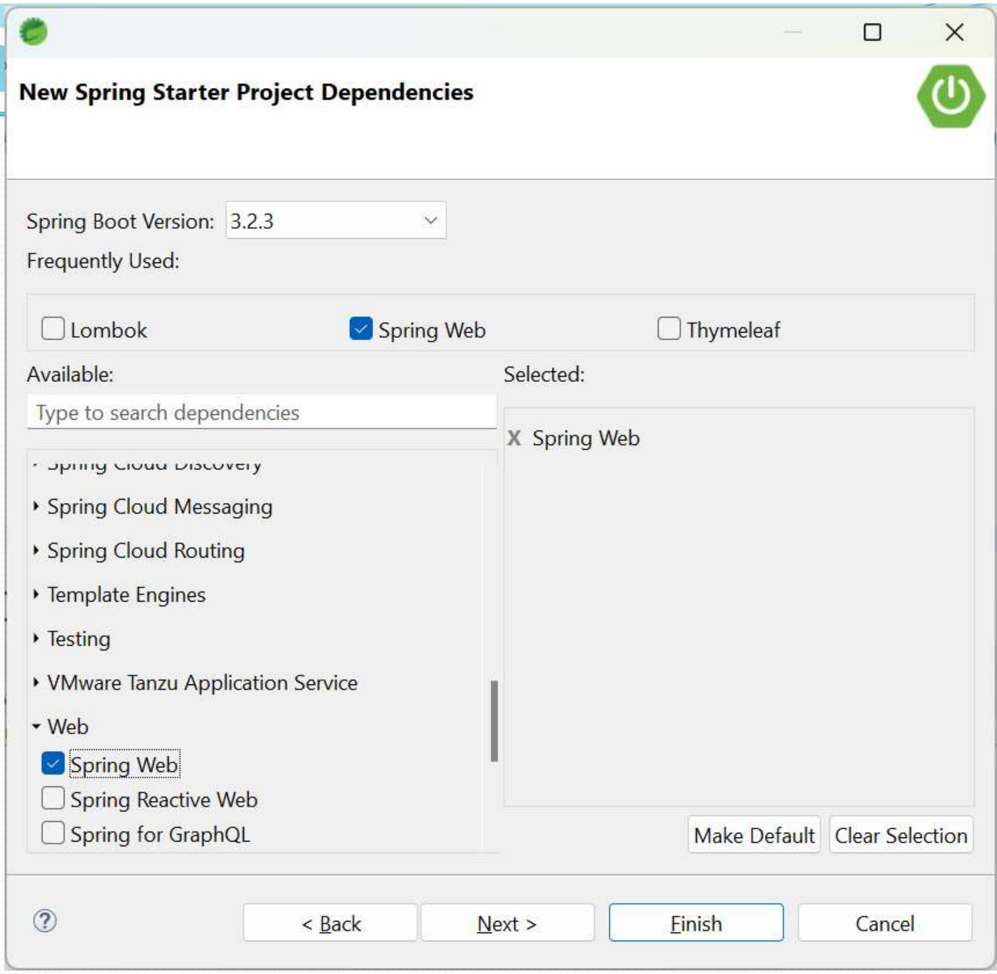
☐ Add project to working sets

Working sets:

Give a name and click next

Setting up

- Select Web→Spring web
- Select Template Engines→Thymeleaf
- Click Next and then Finish



New Spring Starter Project Dependencies

Spring Boot Version: 3.2.3

Frequently Used:

☐ Lombok ☑ Spring Web ☐ Thymeleaf

Available:

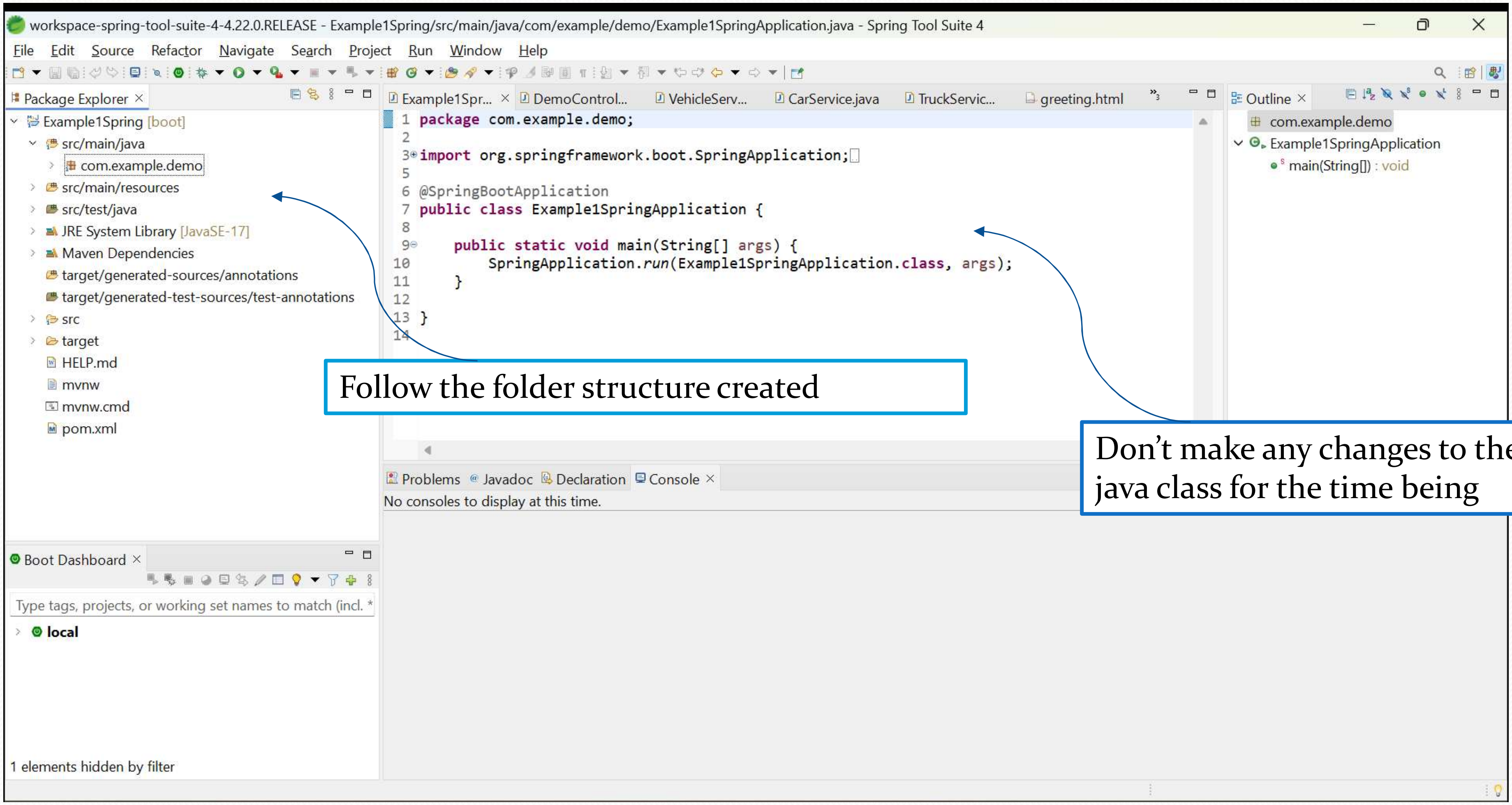
Type to search dependencies

- Spring Cloud Discovery
- Spring Cloud Messaging
- Spring Cloud Routing
- Template Engines
- Testing
- VMware Tanzu Application Service
- Web
  - ☑ Spring Web
  - ☐ Spring Reactive Web
  - ☐ Spring for GraphQL

Selected:

X Spring Web

Setting up



workspace-spring-tool-suite-4-4.22.0.RELEASE - Example1Spring/src/main/java/com/example/demo/Example1SpringApplication.java - Spring Tool Suite 4

```
Example1Spring [boot]
  src/main/java
    com.example.demo
  src/main/resources
  src/test/java
  JRE System Library [JavaSE-17]
  Maven Dependencies
  target/generated-sources/annotations
  target/generated-test-sources/test-annotations
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

```java
package com.example.demo;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class Example1SpringApplication {

    public static void main(String[] args) {
        SpringApplication.run(Example1SpringApplication.class, args);
    }

}
```

```
com.example.demo
Example1SpringApplication
  main(String[]) : void
```

No consoles to display at this time.

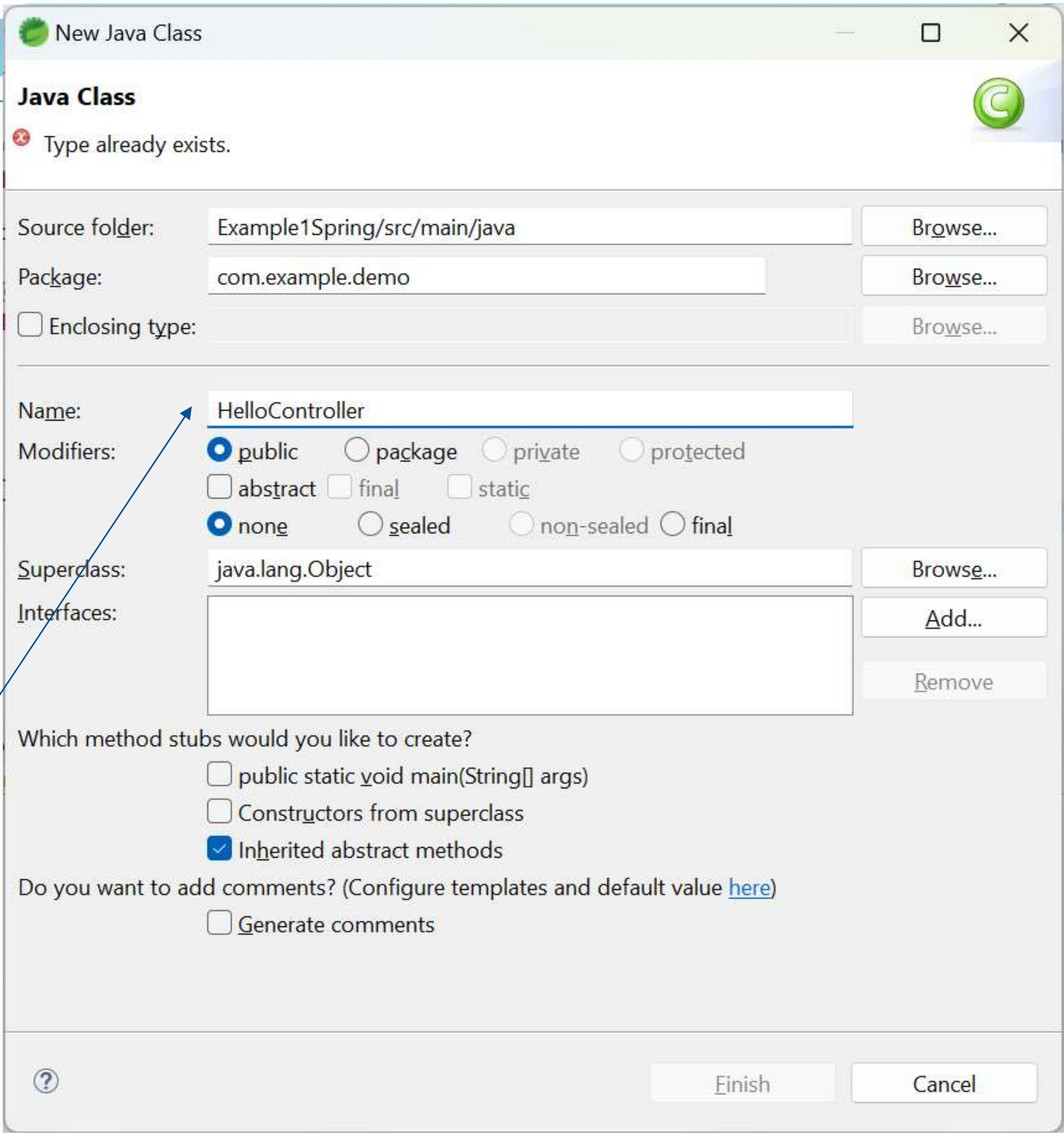
Follow the folder structure created

Don’t make any changes to the java class for the time being

Setting up

- Go to src/main/java→com.example.demo (where you found the java file with main method provided by the framework)
- Right click on the package name→click on new→class

Creating a controller named HelloController



Type already exists.

Source folder: Example1Spring/src/main/java

Package: com.example.demo

□ Enclosing type:

Name: HelloController

Modifiers: ● public ○ package ○ private ○ protected

□ abstract □ final □ static

● none ○ sealed ○ non-sealed ○ final

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

□ public static void main(String[] args)

□ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

□ Generate comments

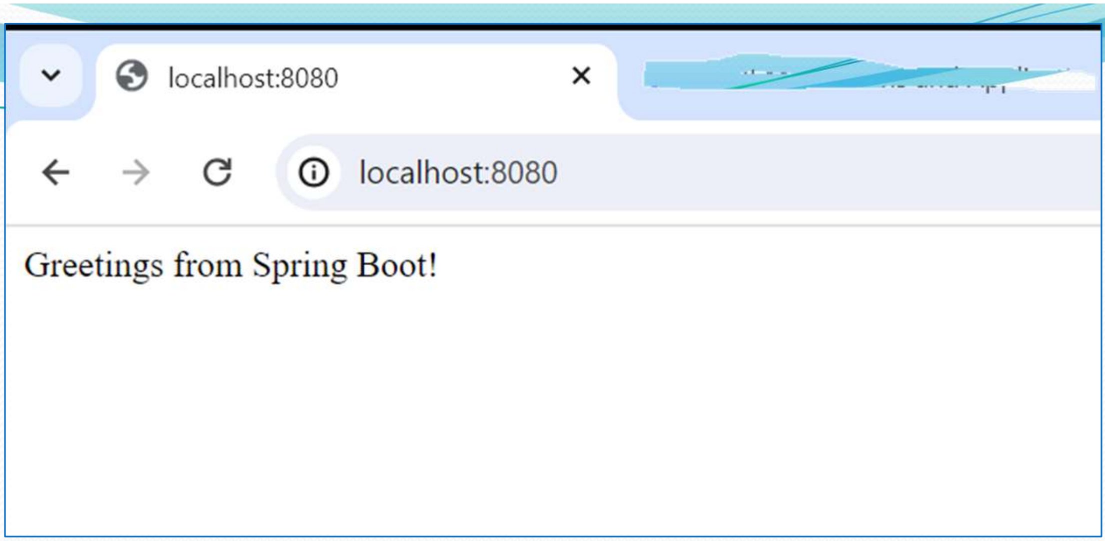
HelloController

```
package com.example.demo;

@RestController
public class HelloController {

      @GetMapping("/")
      public String index() {
            return "Greetings from
                     Spring Boot!";
      }

}
```

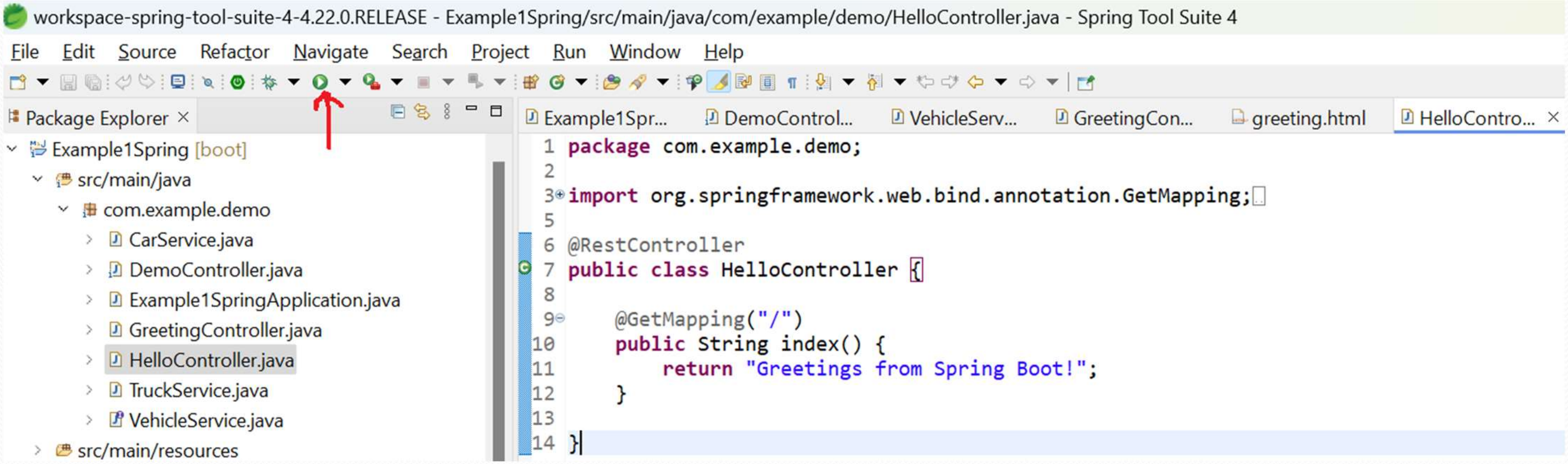
Will be written in the file

Click on annotations (@) and select the import statement; It will be inserted automatically

Executing a Spring Application



Greetings from Spring Boot!



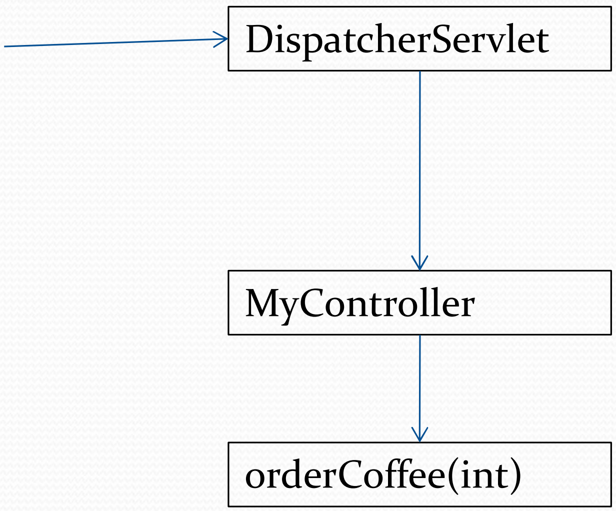
workspace-spring-tool-suite-4-4.22.0.RELEASE - Example1Spring/src/main/java/com/example/demo/HelloController.java - Spring Tool Suite 4

```java
package com.example.demo;

import org.springframework.web.bind.annotation.GetMapping;

@RestController
public class HelloController {

    @GetMapping("/")
    public String index() {
        return "Greetings from Spring Boot!";
    }

}
```

Spring

- In Spring
  - A specialized servlet-DispatcherServlet
  - One or more controllers having simple methods to process HTTP requests
  - The DispatcherServlet routes requests to appropriate controller-individual methods of the controllers
  - DispatcherServlet extracts request parameters, performs data validation and marshalling
  - Provides an extra layer of routing over web.xml

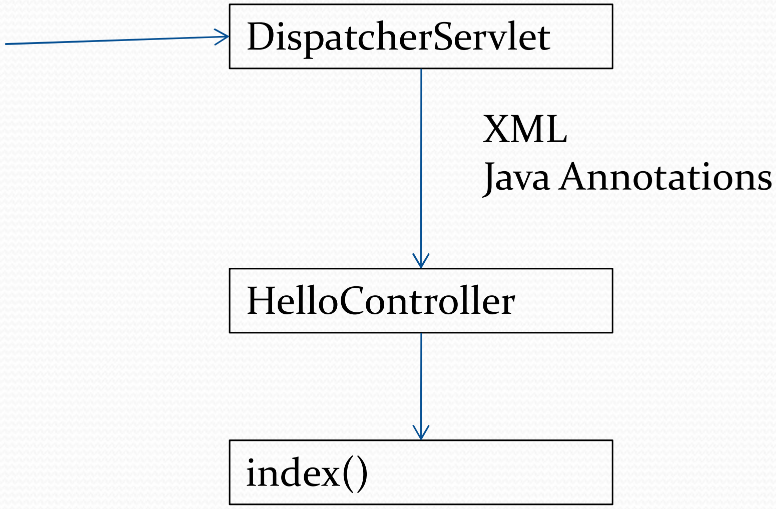
```
public class MyController {
    String orderCoffee(int) {
       ….
return …
}
```



```
@WebServlet("/SelectCoffee")
public class CoffeeSelect extends
HttpServlet {

p.v. doPost(HttpServletRequest
request, HttpServletResponse
response)
            throws IOException,
ServletException {
     //extract parameters from request
    //validation
    //Construct objects with
    parameters
    //do the processing
    }
```

Spring

Spring Controllers are plain java objects
No special interfaces to be implemented or classes to be inherited


DispatcherServlet

XML
Java Annotations

HelloController

index()

- ❑ Routing is possible based on Path like servlets
- ❑ Request parameters using annotations
- ❑ Data validation is taken care of

Routing through DispatherServlet

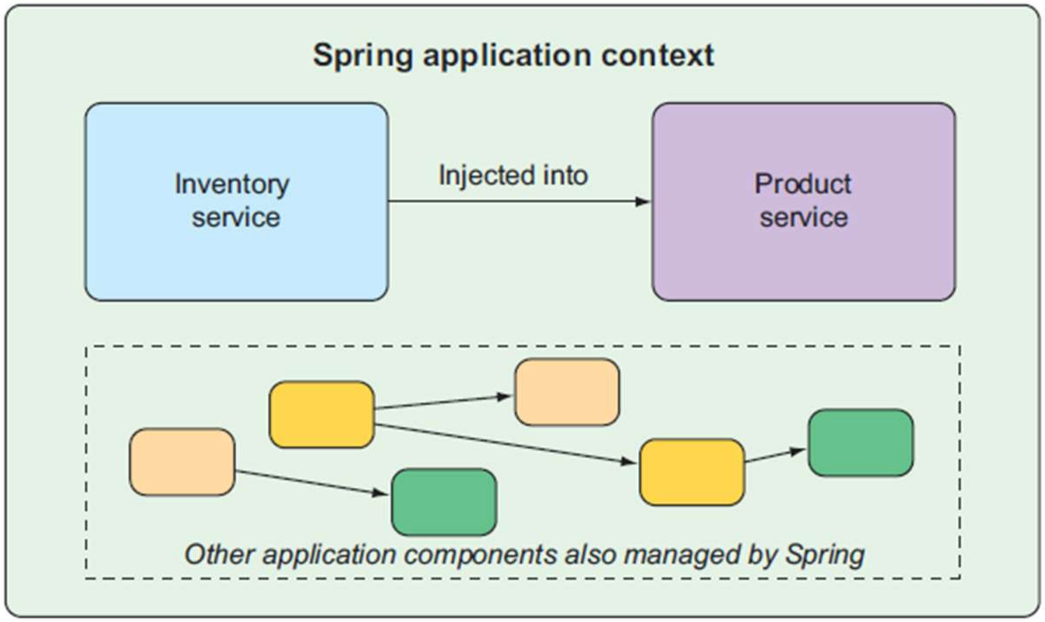
```
@RestController
public class HelloController {


@RequestMapping(“/”)
public String index() {
        return "Greetings from Spring Boot!";
        }
```

A Simple java class-no framework code

Routing through DispatherServlet

```
@RestController
public class HelloController {


@GetMapping(“/”)
public String index() {
       return "Greetings from Spring Boot!";
       }



@RequestMapping(“/friends”)
public String findFriend() {
       return "Greetings !";
       }
```

Spring at a glance


Spring application context
Inventory service
Injected into
Product service
Other application components also managed by Spring

- Spring offers a *container*, often referred to as the *Spring application context*, that creates and manages application components.
- These components, or *beans*, are wired together inside the Spring application context to make a complete application
- The act of wiring beans together is based on a pattern known as *dependency injection* (DI)
- A full portfolio of related libraries offer a web framework, a variety of data persistence options, a security framework, integration with other systems, runtime monitoring, microservice support, a reactive programming model, and many other features necessary for modern application development

Mapping Request parameters to method parameters

Retrieves request parameters and performs basic data validation so that value of *name1* can be mapped to *name*

```
@Controller
public class GreetingController {

        @PostMapping("/greeting")
        public String greeting(@RequestParam(name="name1", defaultValue="World") String
                                                 name, Model model) {

                model.addAttribute("name", name);
                return "greeting";
    }
```

Model object holds the key-value pairs that propagates to the view layer, that is, the html file

“greeting” indicates the name of the html file in the src/main/resources/templates directory

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head></head><body>
        <p th:text="|Hello, ${name}!|" />
</body>
</html>
```

Mapping Request parameters to method parameters

```
@Controller
public class ContactController {

@RequestMapping(value={“/search”}, method = RequestMethod.GET)
   public Contacts searchContacts(
       @RequestParam searchstr String SearchStr) {
           //retrieve contacts
           Contacts c=…
           …
           return c;
      }
```

Mapping Requests

```
@RestController
@RequestMapping(value = "/demo")
public class DemoController {

@RequestMapping(value = "/login")
public String sayHelloWorld() {
 return "Hello World ";

}

@RequestMapping(value = "/dummy")
public String sayHelloDummy() {
 return "Hello World dummy";

}

}
```

- No need to worry about
  - how that request got to the server,
  - what format it got there in,
  - how all the data got extracted from it.
- It simplifies the methods and write cleaner, simpler methods, by using request parameters in the request mapping to extract that data and pass it into the method

```
@Controller
public class ContactController {

@RequestMapping("/search/{str}")
   public Contacts searchContacts(
                              Search s) {
          //retrieve contacts
          Contacts c=…
           …
           return c;
         }
```

Path variable provides a nicer way of parsing the request parameters rather than ?<key>=value

https://spring.io/guides/gs/serving-web-content/

```
@Controller
public class ContactController {

@RequestMapping(“/search/”)
  public Contacts searchContacts(
                 Search s) {
      //retrieve contacts
      Contacts c=…
            …
            return c;
         }
```

```
public class Search  {

  private string fname;
  private string lname;

  public String getFname()
   {..}
  public setFname(String name)
   {..}

…}
```

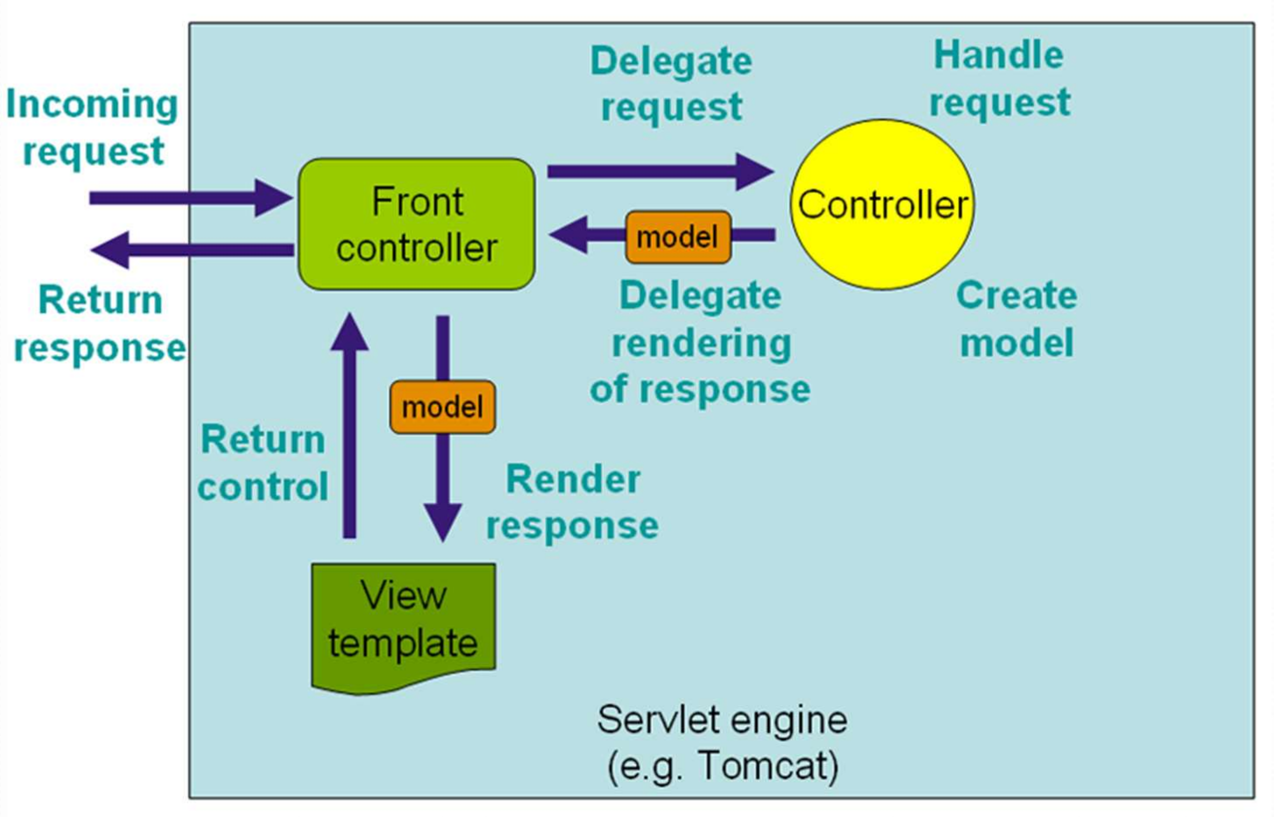
Automatic data marshalling through HTTP message converters

Response

```
@RequestMapping(value = "/prod", produces = {"application/JSON"})
  @ResponseBody StringResponse getProduces(){
  StringResponse s= new StringResponse();
  s.setResponse("Attribute!");
    return s;
  }
```

```
{"response":"Attribute!"}
```

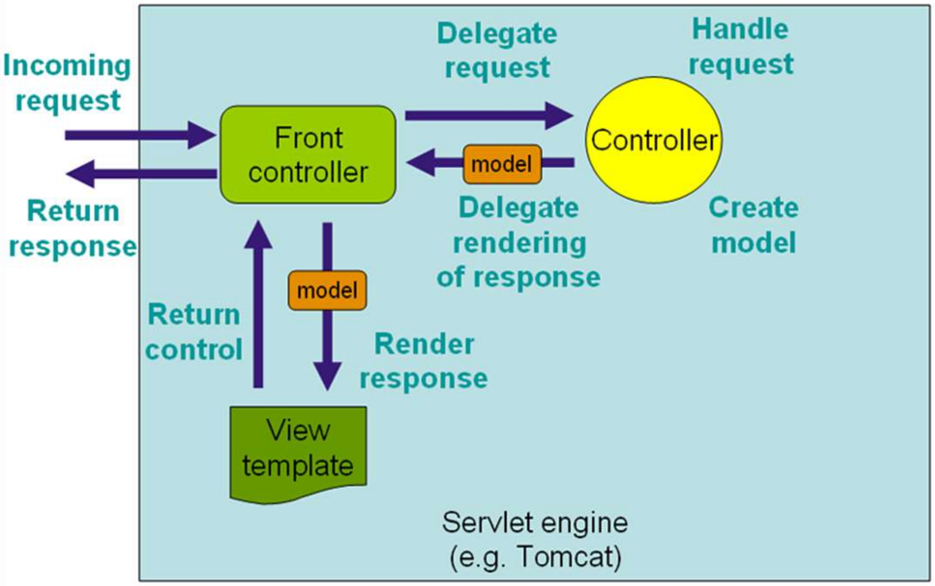
MVC Workflow



DispatcherServlet

- It gets its name from the fact that it dispatches the request to many different components, each an abstraction of the processing pipeline

1. Discover the request's Locale; expose for later usage.
2. Locate which request handler is responsible for this request (e.g., a Controller).
3. Locate any request interceptors for this request. Interceptors are like filters, but customized for Spring MVC.
4. Invoke the Controller.
5. Call postHandle() methods on any interceptors.
6. If there is any exception, handle it with a HandlerExceptionResolver.
7. If no exceptions were thrown, and the Controller returned a ModelAndView, then render the view. When rendering the view, first resolve the view name to a View instance.


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)