ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA
Universidad de Córdoba

UNIVERSIDAD D CÓRDOBA

Programación web

Prácticas

Semana 2: Modelo-Vista-Controlador y primeros pasos en Spring

Aurora Ramírez Quesada (aramirez@uco.es)

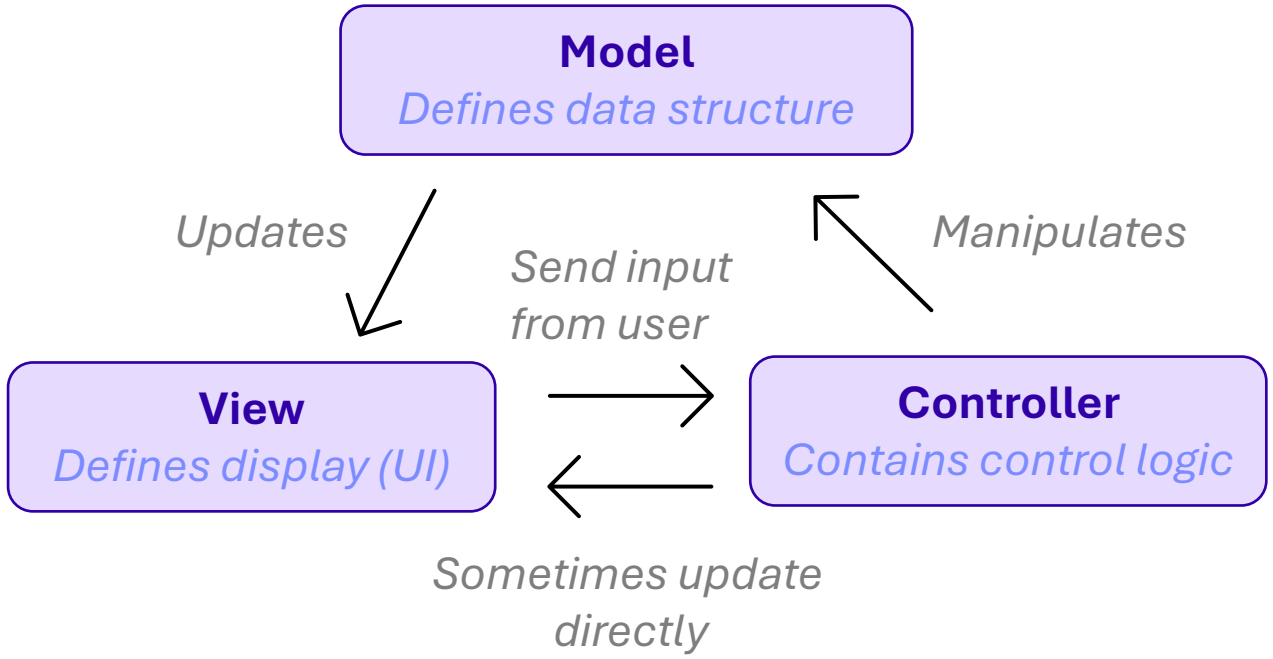
Departamento de Ciencia de la Computación e Inteligencia Artificial

Universidad de Córdoba

Índice de contenido

Patrón MVC
Definición

- **MVC** (Modelo-Vista-Controlador) es un patrón de diseño software que permite la separación entre los datos, la lógica de control y la visualización de los datos.
  - **Modelo**. Se centra en definir los datos y la lógica de negocio que los acompaña
  - **Vista**. Establece cómo se representan los datos recibidos, sirviendo de interfaz con el usuario
  - **Controlador**. Son los encargados de gestionar la lógica de actualización del modelo a respuesta del usuario, así como enrutar el flujo de vistas en base a los cambios del modelo



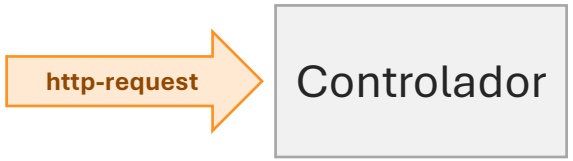
- Fuente: https://developer.mozilla.org/es/docs/Glossary/MVC

Patrón MVC
MVC para aplicación web

- Patrón muy habitual en desarrollo web, donde se utilizan tecnologías específicas:
  - **Modelo**. Base de datos, habitualmente relacional (MySQL, Oracle, PostreSQL, etc.)
  - **Control**. Lenguajes de lado cliente (Javascript) o servidor (Java)
  - **Vista**. HTML5/CSS3/JS, incorporando lenguajes para contenido dinámico (JSP, Thymeleaf, etc.)
- Implementado en *frameworks* de desarrollo web como Spring o Angular JS
  - En ocasiones puede adaptar el modelo, imponiendo o relajando restricciones sobre la estructura del patrón MVC "puro"
  - Toman el control de la lógica de la aplicación, el programador solo implementa código para el dominio de aplicación concreto
  - Algunos permiten la generación automática de algunos componentes, reduciendo el coste de configuración y haciendo transparente la implementación del patrón al desarrollador

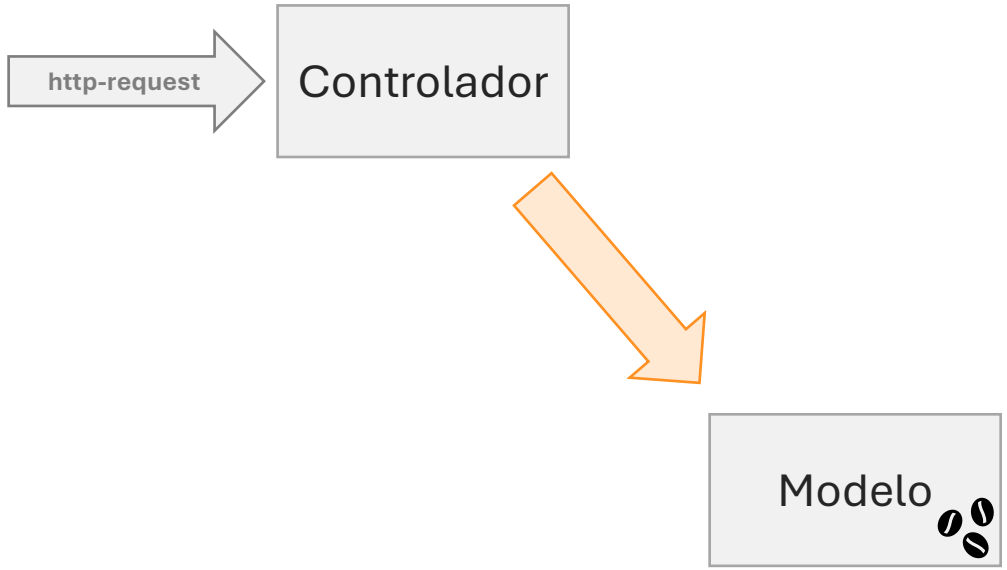
Patrón MVC
Esquema general

- La solicitud ***http-request*** siempre llega primero al **controlador**
- Se recuperan los parámetros de la solicitud
- Se recupera la información de sesión
- Se realiza el control de privilegios y de seguridad (p.ej., mediante ACL)
- Se comprueban las restricciones y se ejecuta la lógica de negocio requerida
- El controlador no implementa ningún tipo de código de agente de usuario
- Se accede a fuentes de datos para recuperar toda la información necesaria



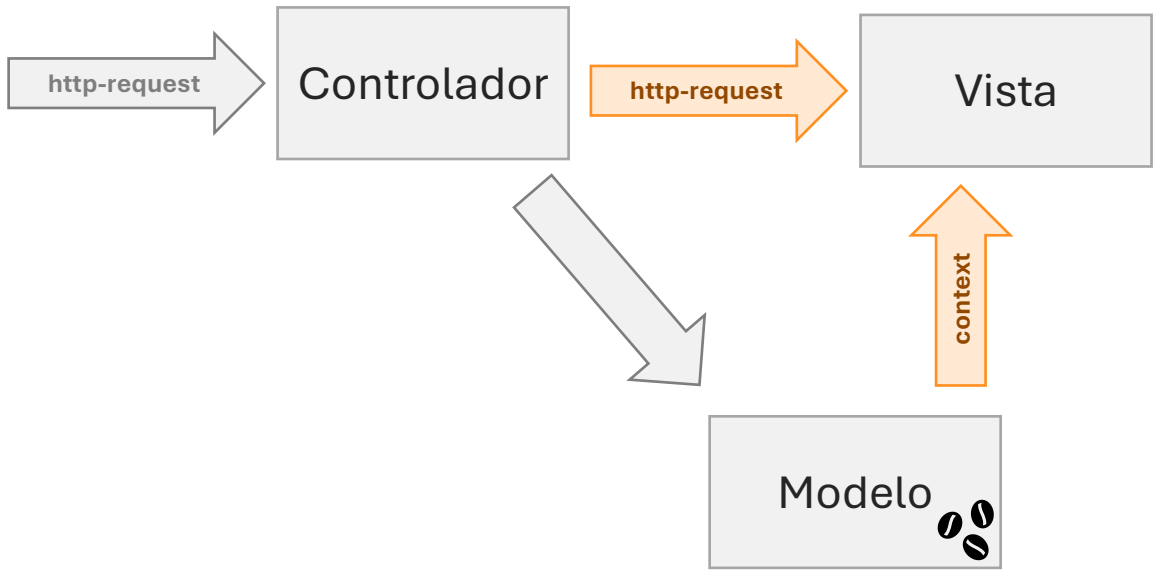
Patrón MVC
Esquema general

- La información se transfiere al modelo, usualmente mediante objetos especializados:
  - *Data Access Object (DAO)*
  - *Data Transfer Object (DTO)*
- Los modelos pueden definir un alcance ("*scope*") que delimita para quién están accesibles:
  - Durante el procesamiento de la "*request*"
  - Mientras dure la sesión de navegación
- A los modelos se los conoce como "***beans***" (En Java, *JavaBean*)
  - Comparten una serie de características que los identifican dentro del entorno
  - Según el *framework*, pueden ser creados y actualizados automáticamente



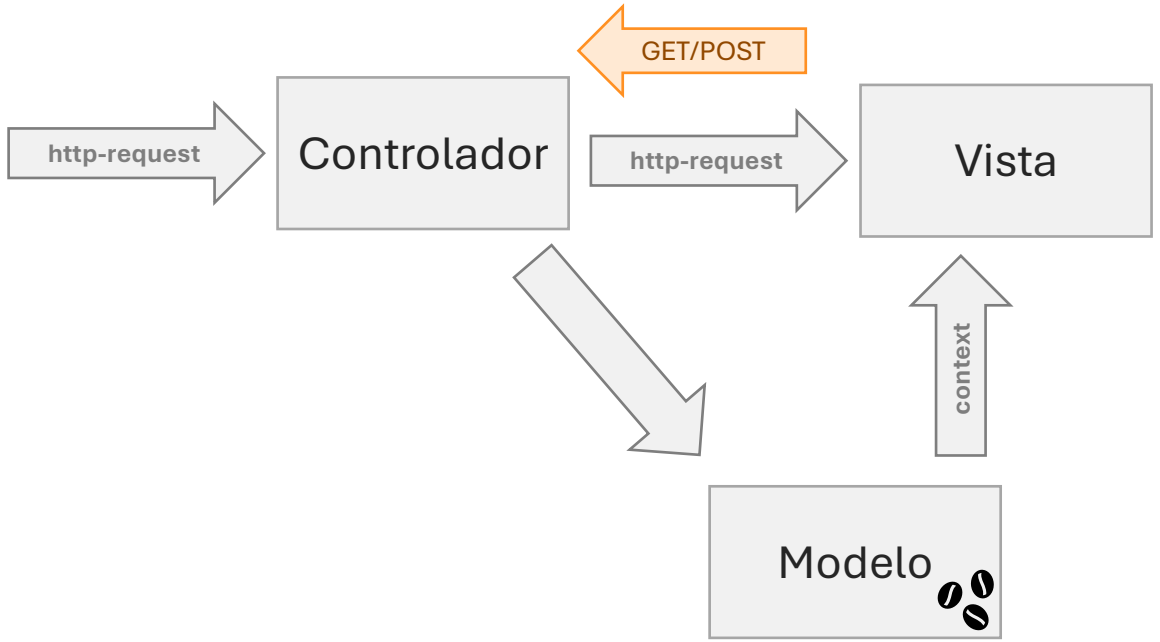
Patrón MVC
Esquema general

- El flujo de navegación se redirige a la vista correspondiente (*forward*):
- La "*http-request*" puede incluir datos sencillos a modo de parámetros, que serán recuperables en la vista o en otros controladores
- Para visualizar datos más complejos, o que estén contenidos en modelos, la vista tendrá acceso a ellos
  - A través de JavaBeans puestos a disposición en la "**sesión**" (contexto de la aplicación)
- Es importante realizar el control de privilegios y de seguridad también en las vistas (ACL)



Patrón MVC
Esquema general

- La vista muestra toda la información necesaria a partir de parámetros y modelos
- La interfaz de usuario se construye conforme a lo implementado en HTML5, CSS3 y JS
- Las interacciones del usuario generan eventos cuya lógica de respuesta puede estar implementada en el cliente (vista)
- El usuario podrá continuar la navegación desde la vista de dos formas habituales:
  - Mediante enlace que lleve a otro controlador
  - Invocando a un controlador desde un formulario
- Los formularios de HTML son los mecanismos más habituales para la devolución de datos del usuario al controlador



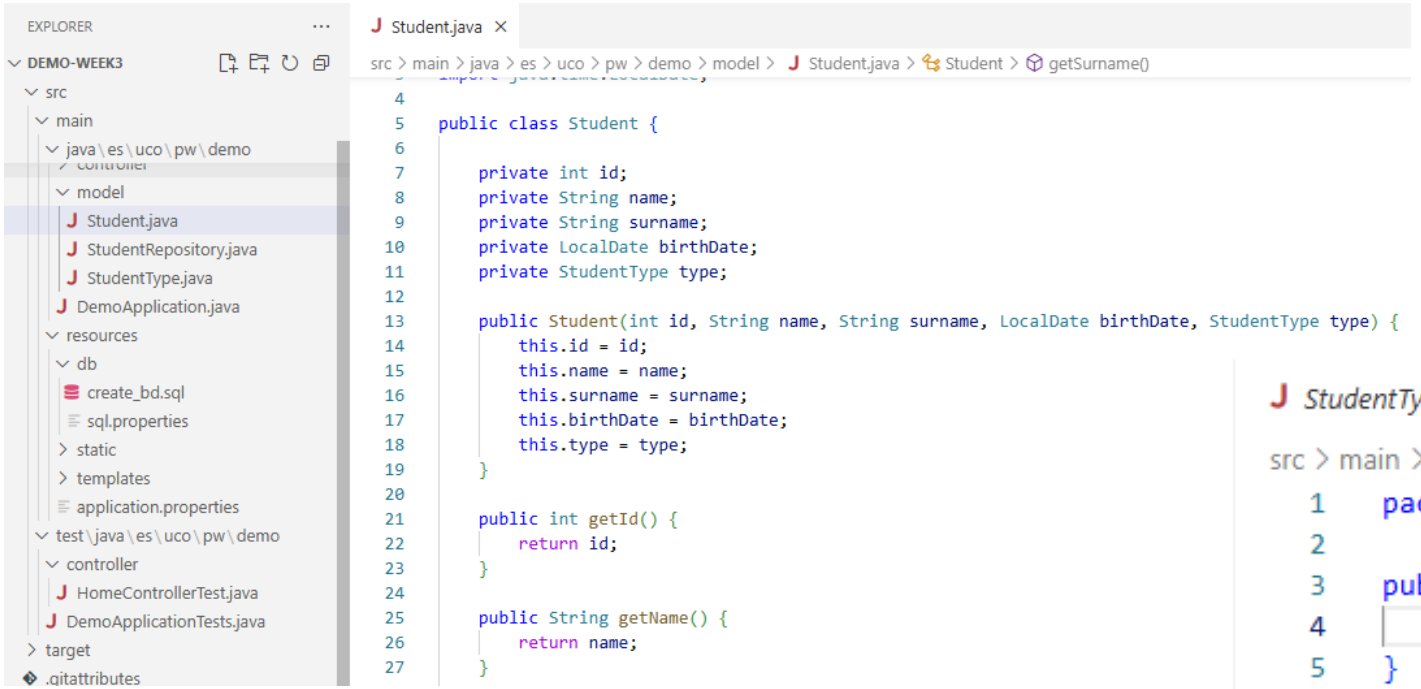
Definición de clases de dominio

- Dentro del MVC, comenzamos definiendo el componente **Modelo**:
  - Se necesitarán clases Java para representar los conceptos del dominio de aplicación
  - Tendrán equivalencia con las tablas de la base de datos relacional
  - Actúan como DTO (*Data Transfer Object*) para intercambiar información con controladores y vistas, aislando de la tecnología de base de datos concreta que se esté utilizando

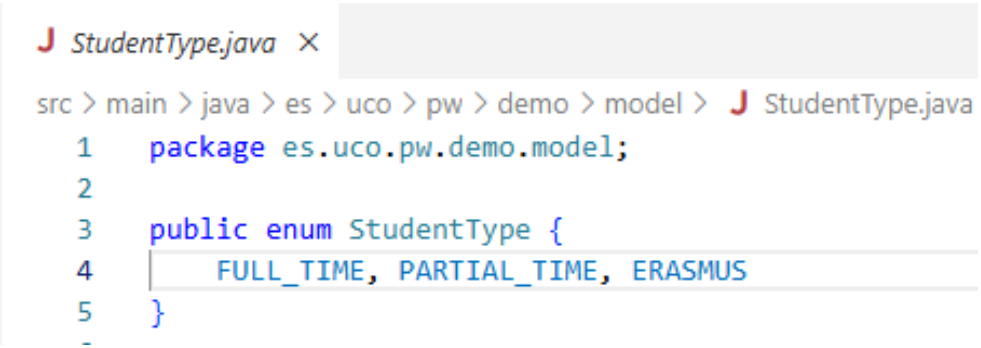
- Partiendo del modelo relacional de base de datos:
  - Definir una clase por cada entidad identificada, incluyendo sus propiedades con los tipos de datos adecuados
  - Definir un constructor con todos los parámetros: Source Action > Generate Constructors...
  - Definir los métodos de acceso (`get/set`) a cada propiedad: Source Action > Generate Getters / Setters...

Definición de clases de dominio
Clases Java

- Definición de clase de dominio: **Student**
  - Es conveniente que las clases tengan un identificador (id), que luego se utilizará como clave primaria
  - Definimos el resto de las propiedades, con los tipos de datos adecuados (p.ej. LocalDate para fechas)



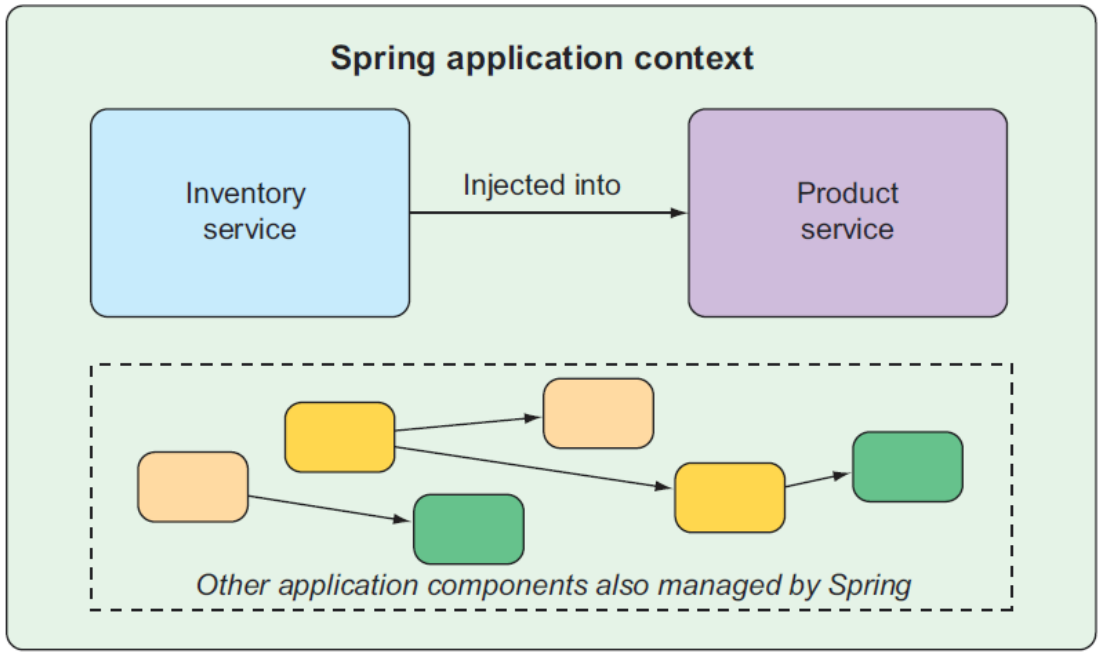
Las enumeraciones (**enum**) permiten definir una lista de valores que puede tomar una variable



Primeros pasos en Spring
Visión general

- Spring ofrece un "contenedor" de objetos (***Spring application context***) que crea y gestiona los componentes (*"beans"*) y recursos de la aplicación

- Spring se encarga de "conectar" estos componentes, a veces descubriendo por sí mismo cuál es el componente adecuado a una tarea (p. ej. Acceso a tablas de base de datos)
- A este procedimiento se le conoce como **inyección de dependencias:**
  - La parte específica de una aplicación (aquello que el desarrollador implementa) se "inyecta" dentro de los *beans* que necesitan conocerlos
  - Esta inyección se suele hacer mediante argumentos en los constructores o métodos de acceso a propiedades (`get`/`set`)



Contexto de la aplicación en Spring.
Fuente: Fig. 1.1 - "Spring in Action" (6th ed.).

Primeros pasos en Spring
Visión general

- Ofrece un componente especializado para MVC: **Spring Web MVC**
  - Se construye sobre la API de Servlets (base de la definición de controladores web en Java)
  - Permite la integración de varios lenguajes de marcado para la creación de vistas con contenido dinámico: Thymeleaf, FreeMarker, Groovy, Java Server Pages (JSP), etc.
  - Fácil configuración con ficheros de propiedades sencillos y ocultando detalles más complejos
  - Proporciona un estilo de programación basado en **anotaciones** (en lugar de extender clases)
    - Permiten identificar ciertos tipos de elementos predefinidos (*controller*, *request*, etc.)
    - El programador se concentra en definir elementos, y Spring hace el "*mapeo*" para ponerlos disponibles en el sitio adecuado

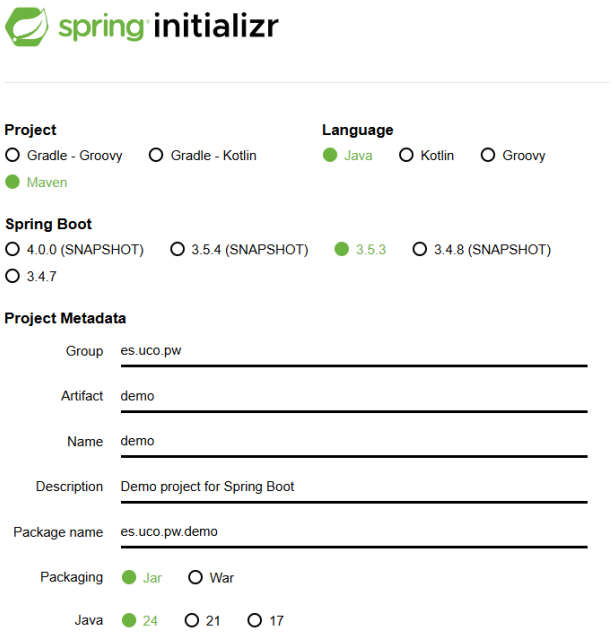
```java
@Controller
public class HelloController {

    @GetMapping("/hello")
    public String handle(Model model) {
        model.addAttribute("message", "Hello World!");
        return "index";
    }
}
```

https://docs.spring.io/spring-framework/reference/we/webmvc.html

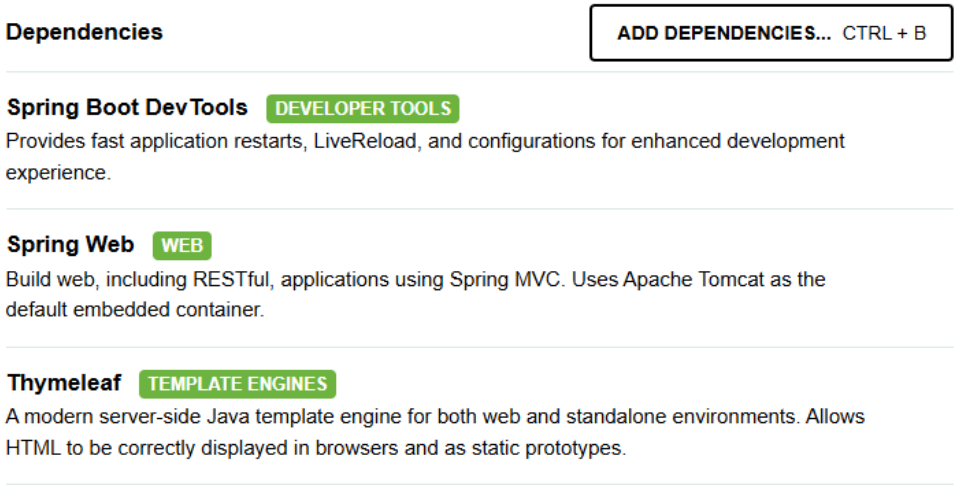
Primeros pasos en Spring
Creación del proyecto

- Dos formas de crear un proyecto Spring
  - Dentro de VSC, si se ha instalado la "Spring Tool Suite": File > New > Spring Starter Project
  - [Recomendado] Mediante Spring initialzr: https://start.spring.io/
- Elementos básicos que hay que definir:
  - Tipo de proyecto: Maven
  - Versión de Spring: 3.5.3
  - Lenguaje y versión: Java 17
  - Nombre de la aplicación: Determinará la jerarquía de paquetes (es.uco.pw) y ruta de acceso (/)
  - Tipo de empaquetado: Jar



Primeros pasos en Spring
Creación del proyecto

- Añadir dependencias
  - **Spring Boot Dev Tools**: Herramientas de gestión y configuración de la aplicación
  - **Spring Web**: Soporte al desarrollo con MVC y API REST. **Apache Tomcat** como servidor de aplicaciones embebido
  - **Thymeleaf**: Lenguaje "plantilla" para incorporar información dinámica en HTML




```
32  <dependencies>
33      <dependency>
34          <groupId>org.springframework.boot</groupId>
35          <artifactId>spring-boot-starter-thymeleaf</artifactId>
36      </dependency>
37      <dependency>
38          <groupId>org.springframework.boot</groupId>
39          <artifactId>spring-boot-starter-web</artifactId>
40      </dependency>
41
42      <dependency>
43          <groupId>org.springframework.boot</groupId>
44          <artifactId>spring-boot-devtools</artifactId>
45          <scope>runtime</scope>
46          <optional>true</optional>
47      </dependency>
48      <dependency>
49          <groupId>org.springframework.boot</groupId>
50          <artifactId>spring-boot-starter-test</artifactId>
51          <scope>test</scope>
52      </dependency>
53  </dependencies>
```

- Una vez generado, el proyecto se descarga en formato ZIP. Descomprime e impórtalo directamente en Visual Studio Code
- Conforme avance el desarrollo, se irán incluyendo otras dependencias en el fichero *pom.xml* generado por Maven

Primeros pasos en Spring
Estructura del proyecto

- El proyecto importado consta de varios directorios, ficheros y recursos:
  - Directorio `src`: Contiene el código fuente de la aplicación
    - Directorio `main`: Contiene el código funcional de la aplicación y sus recursos
      - Directorio `java`: Inicio de la ruta de paquetes Java en las que se incluirán las clases para implementar la aplicación siguiendo el patrón MVC. Se proporciona una clase con `Main`
      - Directorio `resources`: Elementos no Java necesarios en la aplicación
        - Directorio `static`: contenido estático para el navegador (imágenes, CSS3, JS)
        - Directorio `templates`: plantillas para renderizar contenido en el navegador (Thymeleaf)
        - Fichero *`application.properties`*: Permite configurar aspectos de la aplicación como el nombre, configuración de la base de datos, etc.
    - Directorio `test`: Contiene el código de pruebas
  - Directorio `target`: Contiene el código compilado
  - Fichero *`pom.xml`*: Fichero de configuración de Maven con la especificación de la aplicación
  - Ficheros *mvnw* y *mvnw.cmd*: Se encargan de construir la aplicación

Primeros pasos en Spring

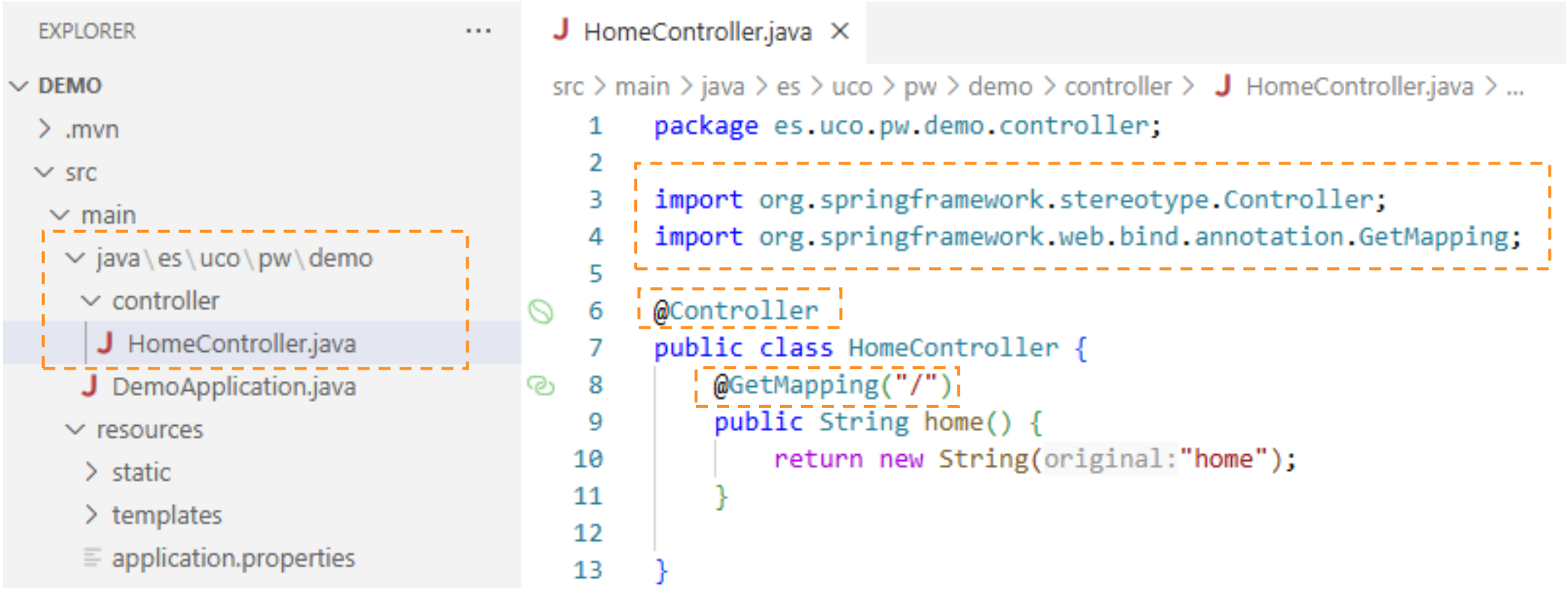
Estructura del proyecto

Primeros pasos en Spring
Ejemplo MVC

- Creación de la página principal (home) y un controlador asociado
  - HomeController.java: Se encargará de manejar las peticiones (*requests*) a la página principal
  - home.html: Plantilla HTML + Thymeleaf para implementar la vista con información dinámica
- Ubicación de los archivos:
  - HomeController.java debe ir dentro de `es.uco.pw.demo.controller`
  - home.html debe ir dentro de *src/main/resources/templates*
- Pasos:
  1. Definir la clase `HomeController` con la anotación `@Controller`
  2. Incluir un método para responder a las peticiones "GET" de la URL deseada: `@GetMapping("/")`
  3. Definir la vista home.html, que importará una imagen desde el directorio *static/images*
  4. Al lanzar la aplicación y acceder a /home.html, Spring lanzará el controlador `HomeController`

Primeros pasos en Spring
Ejemplo MVC

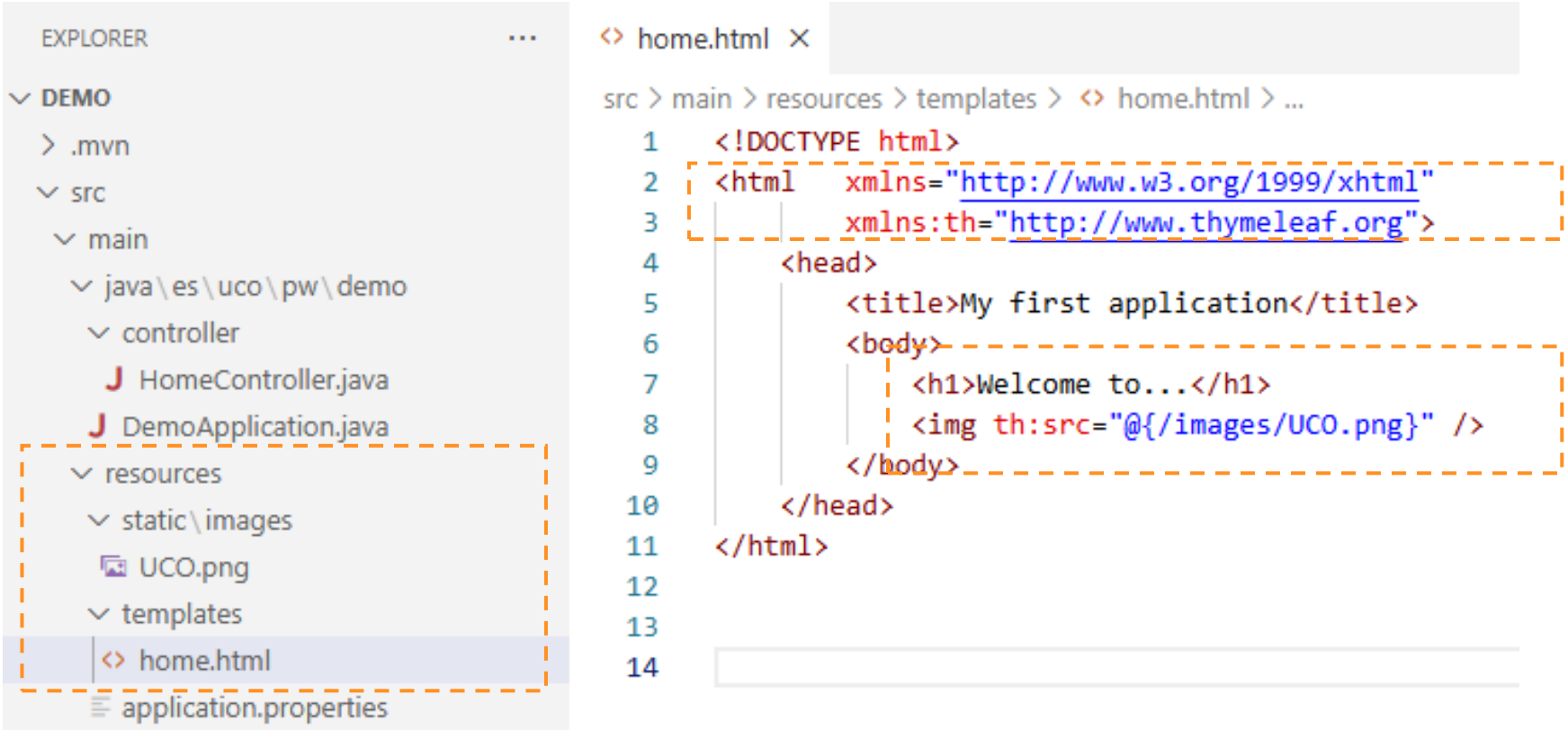
- Definición del controlador **`HomeController.java`**



Primeros pasos en Spring
Ejemplo MVC

Descarga el logotipo de la UCO y ubícalo en el directorio *resources/static/images*

- Definición de la vista ***home.html***



Nota: No es necesario comprender HTML/Thymeleaf por el momento, se explicará más adelante

Primeros pasos en Spring
Ejemplo MVC

- Ejecución
  - La ejecución de la aplicación se lanza desde la clase **`DemoApplication.java`**, que contiene el método **`main`**
  - Si la compilación es correcta, Spring inicializará el servidor **Tomcat** en el puerto 8080
  - La aplicación estará accesible en la URL: https//localhost:8080

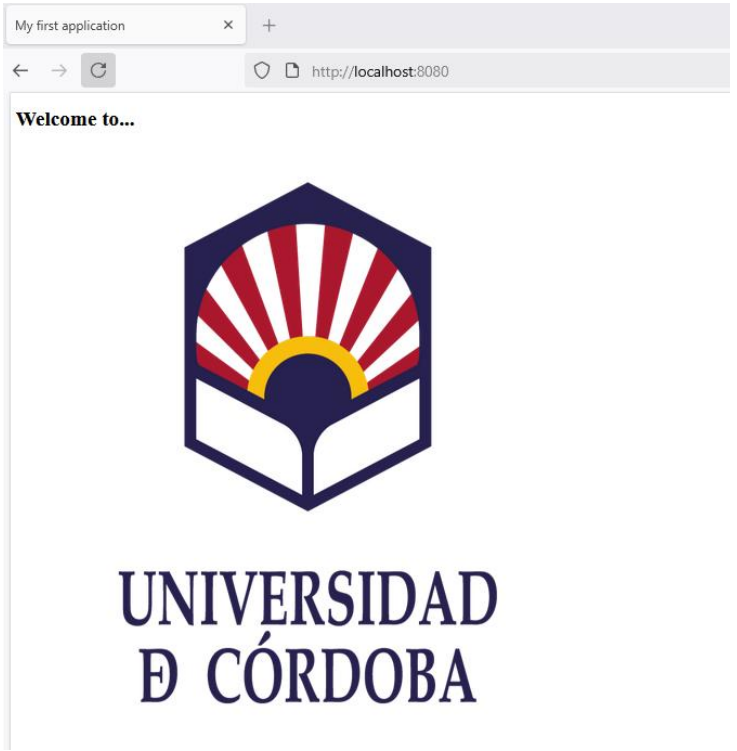
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                                    Run: DemoApplication

PS C:\Users\auror\Documents\Repositorios\VSC-PW2526\github\demo-week1>  & 'C:\Program Files\Java\jdk-24\bin\java.exe' '@C:\Users\auror\AppData\Local\Temp\cp_eckxubvo8pxdp24avsuml03td.argfile' 'es.uco.pw.demo.DemoApplication'

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::                (v3.5.3)

2025-07-15T18:50:17.447+02:00  INFO 20472 --- [demo] [  restartedMain] es.uco.pw.demo.DemoApplication          : Starting DemoApplication using Java 24.0.2 with PID 20472 (C:\Users\auror\Documents\Repositorios\VSC-PW2526\gi
thub\demo-week1\target\classes started by Aurora in C:\Users\auror\Documents\Repositorios\VSC-PW2526\github\demo-week1)
2025-07-15T18:50:17.453+02:00  INFO 20472 --- [demo] [  restartedMain] es.uco.pw.demo.DemoApplication          : No active profile set, falling back to 1 default profile: "default"
2025-07-15T18:50:17.519+02:00  INFO 20472 --- [demo] [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-properties' to 'false' to disable
2025-07-15T18:50:17.519+02:00  INFO 20472 --- [demo] [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level.web' property to 'DEBUG'
2025-07-15T18:50:18.591+02:00  INFO 20472 --- [demo] [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2025-07-15T18:50:18.611+02:00  INFO 20472 --- [demo] [  restartedMain] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2025-07-15T18:50:18.612+02:00  INFO 20472 --- [demo] [  restartedMain] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.42]
2025-07-15T18:50:18.653+02:00  INFO 20472 --- [demo] [  restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2025-07-15T18:50:18.654+02:00  INFO 20472 --- [demo] [  restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1133 ms
2025-07-15T18:50:19.104+02:00  INFO 20472 --- [demo] [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
2025-07-15T18:50:19.143+02:00  INFO 20472 --- [demo] [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path '/'
2025-07-15T18:50:19.152+02:00  INFO 20472 --- [demo] [  restartedMain] es.uco.pw.demo.DemoApplication          : Started DemoApplication in 2.124 seconds (process running for 2.474)
```



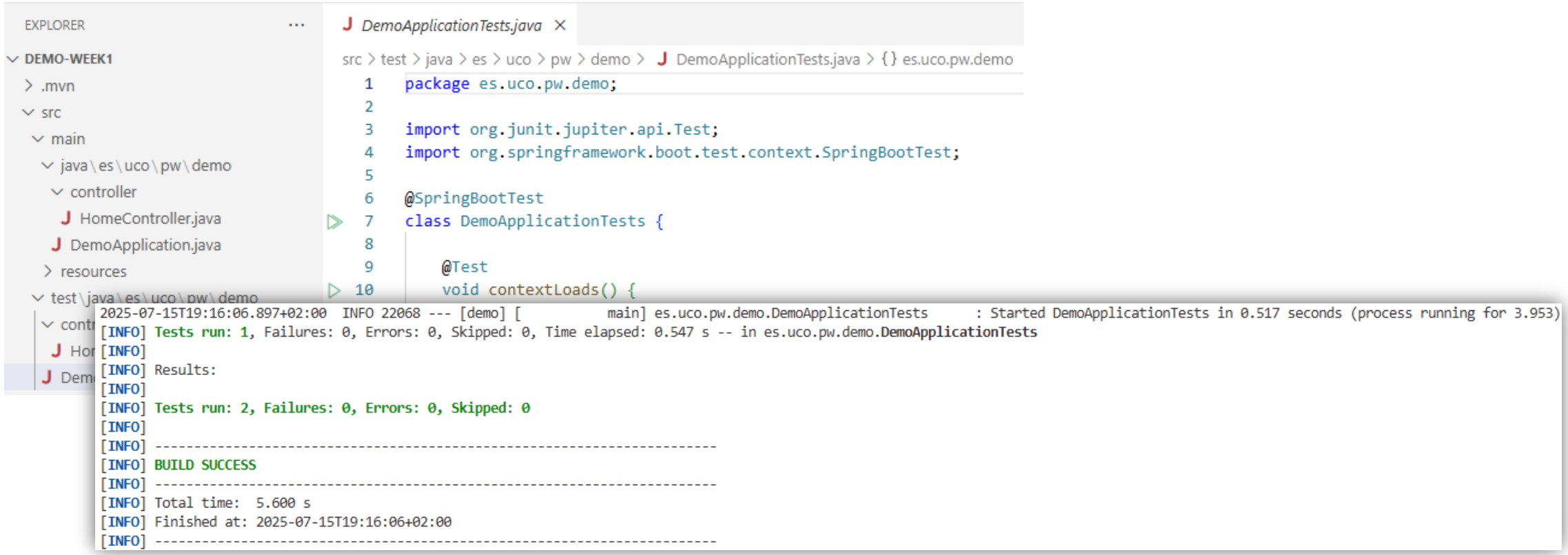
! Si tienes el puerto 8080 ocupado, puedes indicar que se utilice otro en el fichero *application.properties*, por ejemplo: *"server.port=8090"*

Primeros pasos en Spring
Ejemplo MVC

- Al estar basado en Java, podemos usar el *framework* de pruebas **JUnit**
- Con la anotación @WebMvcTest se indica la clase a probar
- El caso de prueba simula una petición **GET** y comprueba que:
  - La respuesta recibida es correcta: HTTP 200 (OK)
  - La vista accedida tiene el nombre que hemos indicado: “home”
  - La vista contiene cierto contenido: “Welcome to...”

Primeros pasos en Spring
Ejemplo MVC

- La clase `DemoApplicationTests`, encargada de ejecutar los casos de prueba (`@Test`), viene implementada por defecto
- Para lanzar las pruebas: `./mvn test` (Linux) o `./mvnw test` (Windows)



Objetivos de la semana

1. Crea un proyecto con las dependencias MVC/Web necesarias (diap. 13 y 14)
2. Replica el ejemplo explicado en clase
   - Versión completa disponible en el repositorio Github: https://github.com/aramirez-uco/pw-examples
3. Realiza el diseño relacional de la base de datos en base al enunciado disponible en Moodle
4. Inicia la implementación de las clases de dominio, definiendo sus propiedades y métodos de acceso (`get/set`)
5. Consulta la bibliografía recomendada y extendida:
   - Capítulo 1: "*Getting started with Spring*" del libro "*Spring in Action*" (6th ed.)
   - Spring MVC: https://spring.io/guides/gs/serving-web-content
   - Pruebas: https://spring.io/guides/gs/testing-web