IMPORTANTE: Ver los “relative paths” en aplicaciones web Java.

**CURSO DE MIGUEL ÁNGEL**

SPRING:

**Anotaciones vistas hasta ahora:**

- @Component (“beans” de Spring)

@Controller

@ControllerAdvice

@Configuration

- @Mapping

@RequestMapping

@GetMapping

@PostMapping

- @RequestParam

- @PathVariable

- @ExceptionHandler

- @Autowired

- @Qualifier

**Interfaces vistas hasta ahora:**

- org.springframework.ui.Model

- org.springframework.web.servlet.HandlerInterceptor

- org.springframework.web.servlet.config.annotation.WebMvcConfigurer

**Clases vistas hasta ahora:**

- org.springframework.web.servlet.ModelAndView

**Clases abstractas vistas hasta ahora:**

**-** org.springframework.web.servlet.handler.HandlerInterceptorAdapter (ver la interfaz HandlerInterceptor)

- org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter (ver interfaz WebMvcConfigurer)

SPRING BOOT:

Por defecto, SpringBoot buscará las páginas de error en “templates/error”.

THYMELEAF:

Cuando Thymeleaf analiza un .html, no sabe resolver las rutas en la propiedad “src”. Hay que indicarlas con *th:src=”@{/imgs/2.png}”* (Spring busca por defecto los recursos estáticos en la carpeta “static”, de manera que es la ruta base sobre la que trabaja al poner “@{}”).

**Separación de funcionalidades**

Conviene programar usando interfaces, de manera que sus implementaciones pueden variar sin modificar el programa que las esté utilizando. Para instanciar estos objetos se utiliza el patrón “Factory”, con lo que se consigue desacoplar el código.

Mediante los interfaces “provider” se desacoplan las implementaciones de los distintos módulos del programa, logrando mayor flexibilidad. (pág. 38 del libro Pro Spring...)

ApplicationContext es una interfaz de Spring que permite instanciar de manera transparente los “beans” que hayan sido declarados, bien en el xml de configuración (app-context.xml), bien mediante anotaciones.s

**Dependency pull:** por ejemplo, JNDI, utilizado para buscar componentes EJB en EJB 2.1 o anteriores.

El propio Spring utiliza ese mecanismo cuando hace un ctx.getBean(<nombre>, <clase>);

**Contextualized dependency lookup:** es similar al modelo “dependency pull”, pero en lugar de acceder a un registro central, la búsqueda se hace directamente sobre el contenedor que gestiona el objeto.

**Inyección de dependencias por constructor:** obvio

**Inyección por métodos “setter”:** es el mecanismo más utilizado de IoC, y de los más sencillos.

**Comparación entre los “Injection” y “Lookup”:**

En general, es mucho mejor el mecanismo de inyección, ya que, entre otras cosas, tiene un impacto cero en los componentes que lo utilizan.

El componente central del contenedor de inyección de dependencias en Spring es la interfaz BeanFactory. En Spring, un “bean” es cualquier componente que sea gestionado por el contenedor.

La configuración de la BeanFactory se lleva a cabo, internamente, mediante las clases que implementan la interfaz BeanDefinition (una implementación es, por ejemplo, DefaultListableBeanFactory). Externamente esa configuración se hace con algún fichero .xml (ahora ya con anotaciones).

**ApplicationContext**: interfaz que hereda de BeanFactory. Tiene en sus implementaciones, además de los servicios de DI, servicios de AOP, internacionalización, manejo de eventos y otros.

Configuración básica:

Fichero app-context-annotation.xml

@Autowired: tiene un equivalente (no del todo. Soporta el parámetro “name) en el estándar de Java que es @Resource(<id>). Otro equivalente (éste sí) de JEE es @Inject, también soportado por Spring.