

## MODULE 4

# INTRODUCTION TO SPRING

# TABLE OF CONTENT

- Java Bean
- Spring
- Project Structure
- Maven
- Spring Boot
- Yaml Property File
- Application Class
- Configuration Class

# JavaBean

- ▶ **Plain Old Java Object (POJO)** is an ordinary Java Object
  - A POJO class does not have
    - extends
    - implements
    - annotations
- ▶ **JavaBean** is a POJO that
  - Has a no-argument constructor
  - Allows access to properties using getter and setter methods
  - Is serializable

# Spring

- ▶ Open source application framework
- ▶ To simplify development of enterprise application
  - POJO-oriented development
- ▶ Dependency injection (DI) and aspect-oriented programming
  - Lightweight development with POJOs
  - Loose coupling through DI and interface orientation
- ▶ **Inversion of control** container for the Java platform

# Spring – *Inversion of Control (IoC)*

## ▶ **Inversion of control container**

- To manage Java object lifecycles
  - Creating objects
  - Calling initialization methods
  - Configuring objects by wiring them together
- Done mainly via **dependency injection**

## ▶ The container can be configured by providing the information required to create the beans

- Through XML files
- Through Java **annotations** in classes ←

# Spring – *Dependency Injection*

- ▶ Dependency Injection
  - The ability to inject components into an application in a typesafe way
  - The ability to choose at deployment time which implementation of a particular interface to inject
- ▶ The programmer does no longer create **objects**
  - But describes how they should be created
- ▶ The programmer does no longer call **services** and **components**
  - But tells which services and components must be called
- ▶ Benefit
  - Code easier to maintain
  - Code easier to test

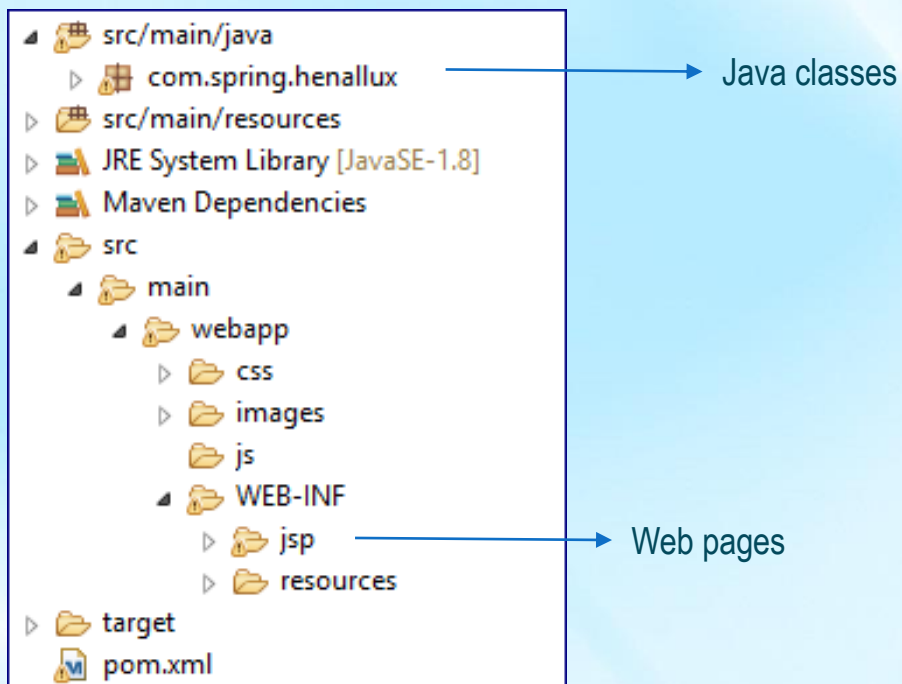
# Spring – *Dependency Injection*

- ▶ DI involves four elements
  - The implementation of a service object
  - The client object depending on the service
  - The interface the client uses to communicate with the service
  - The injector object responsible for injecting the service into the client
    - Also referred to as an assembler, provider, container, factory, or spring



# Project Structure

## ► In Package Explorer View





# Maven

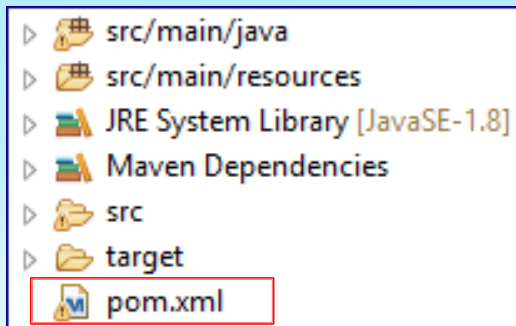
- ▶ Software project management and comprehension tool
  - Describes how software is **built**
  - Describes its **dependencies**
- ▶ Maven can manage
  - Project's build
  - Reporting
  - Documentation

# Maven

- ▶ To simplify the build processes
  - Making the build process easy
  - Providing a uniform build system
  - Providing project information
  - Providing guidelines for best practices development
  - Allowing transparent migration to new features

# Maven – *pom.xml*

- ▶ Maven is based on the concept of a Project Object Model (POM)
- ▶ *pom.xml* used to build the project
  - Contains information about the project and configuration details
  - Default values for most projects



# Spring Boot

- ▶ To create easily Spring based Application
  - Needs very little Spring configuration
- ▶ Lets the developer focus on the application's development
  - Removes the need to be concerned with other aspects of application lifecycle
    - Like deployment and management

# Spring Boot

## ► Features

- Create stand-alone Spring applications
- Embed Tomcat
- Provide 'starter' POMs to simplify Maven configuration
- Automatically configure Spring whenever possible
- Provide production-ready features
- No code generation and no requirement for XML configuration

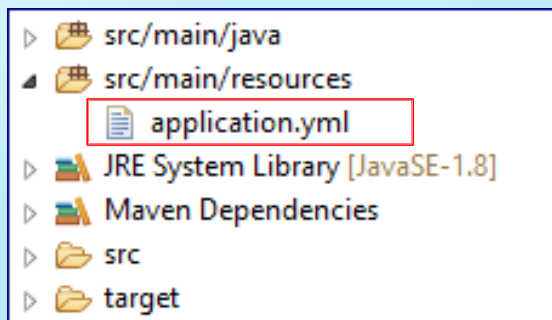
# Spring Boot

- ▶ Add a dependency in pom.xml

```
<dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-web</artifactId>  
</dependency>
```

# Yaml Property File

- ▶ Use yaml file for external properties



- ▶ YAML is a superset of JSON
  - Convenient syntax for storing external properties in a hierarchical format
  - E.g,

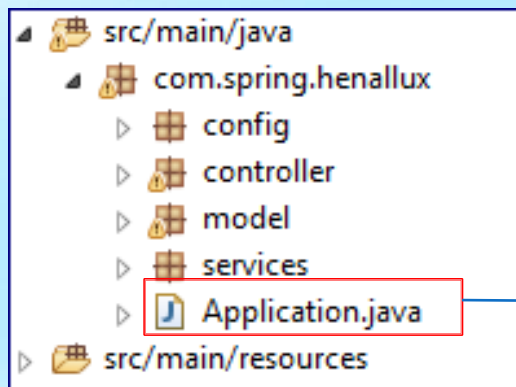
```
1 # Local server
2 server:
3   # port is used by spring-boot-admin
4   port: 8080
5   contextPath: /first
```

→ Root of the project



# Application Class

- ▶ In the root package *above all other java classes*



Run to launch the application

# Application Class

- ▶ Class Annotations
- ▶ **@Configuration**
  - Tags the class as a source of bean definitions for the application context
- ▶ **@EnableAutoConfiguration**
  - Tells Spring Boot to start adding beans
    - Based on classpath settings, other beans, and various property settings
- ▶ **@ComponentScan**
  - Tells Spring to look for other components, configurations, and services in the package

# Application Class

## ► Main method

- Uses Spring Boot's ***SpringApplication.run()*** method to launch the application

```
@Configuration
@EnableAutoConfiguration
@ComponentScan
public class Application {

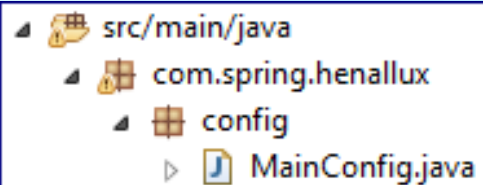
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

# Configuration Class

- ▶ Spring Boot favors Java-based configuration
- ▶ One or more Configuration classes
  - Contains bean definitions
- ▶ Class annotation: **@Configuration**
  - Indicates that the class can be used by the Spring IoC container as a source of bean definitions
- ▶ Bean definition
  - Method Annotation: **@Bean**
    - ⇒ The method will return an object that should be registered as a bean in the Spring application context

# Configuration Class

- ▶ E.g, MainConfig



```
src/main/java
├── com.spring.henallux
│   └── config
│       └── MainConfig.java
```

# Configuration Class

```
@Configuration
public class MainConfig extends WebMvcConfigurerAdapter {

    @Bean
    public ViewResolver viewResolver ()
    {
        InternalResourceViewResolver resolver = new InternalResourceViewResolver();
        resolver.setPrefix("/WEB-INF/jsp/");
        resolver.setSuffix(".jsp");

        return resolver;
    }
}
```