Managing Beans and Dependency Injection

Spring Beans

- A Spring bean is a Java object that is managed by the Spring Container
- Spring is responsible for creating, configuring and managing its lifecycle
- When a bean is created it is typically stored in a bean container

Spring Beans - Dependency Injection

- One of the main benefits of using Spring Beans is the support for DI
- Spring can automatically wire up dependencies between objects with DI
- This allows more modular and reusable code since objects can be swapped out or reused in different contexts

Spring Beans - Configuration

- Spring Beans are typically defined in a configuration file or a Java configuration class
- This allows developers to configure and customize the properties and behavior of the bean

Spring Beans - Scoping

- Spring Beans can be scoped to control their lifecycle and visibility
- A bean can be defined as a singleton - only one instance is created and shared across the application
- A bean can be defined as a prototype - a new instance is created each time the bean is requested
- Other scopes, such as request and session can be used in web apps

Configuration classes

- Configuration classes are used to define and configure Spring Beans
- Can be used instead of XML configuration that was used in the past
- Are annotated with @Configuration which tells Spring their purpose

Configuration classes - features

- Bean Definition - each @Bean annotated method defines a Spring bean and its dependencies.
- Takes advantage of the dependency injection capabilities in Spring to wire up dependencies between beans
- Dependencies can be specified as constructor parameters or using @Autowired annotations

Demo 1 - Dependency Injection

- Using the @Autowired annotation
- Using Constructor parameters

Value Injection

- Value Injection is the ability to inject values into a bean at runtime, rather than specifying them at configuration time
- The @Value annotation can be used to inject values into a bean's property
- The Environment object can be used to access property values
- Using the @ConfigurationProperties annotation

Demo 2 - Value Injection

- Using the @Value annotation
- Using the Environment object

Spring Expression Language (SpEL)

- Spring Expression Language is used to evaluate expressions at runtime
- It can be used to reference bean properties, invoke methods, perform arithmetic and logical operations, and more. For example:
- The value #{3 + 4} can be used to calculate a value
- The value #{systemProperties[’user.home']} can be used to inject the value of the user.home system property
- th:text=”${user.firstName}" can be used in a Thymeleaf template to display the firstName variable in the user attribute

Demo 3 - SpEl

- SpEL for injecting a calculation
- SpEL for injecting a system property

Exercise 1 - Dependency Injection

- Reuse the solution from Creating a Spring Boot Application
- In one of the methods that takes two int values as input and returns the sum, refactor the solution so that you now have a Calculate class with a sum function that sums up two int values, and use an object of this new class to do the calculation from the Controller method
- Use an @Autowired annotation in the Controller to inject the dependency to the Calculator class into an instance variable in the Controller class
- Just remember to put a @Service annotation on the Calculator class!

Exercise 2 - Dependency Injection

- Continue to reuse the same solution from Creating a Spring Boot Application

- In the other of the methods that takes two int values as input and returns the sum, refactor the solution the same way as in Exercise 1 but this time use a Constructor parameter in the Controller to inject the dependency of the Calculator into another instance variable in the Controller class

- You will now have two different instance variables in the Controller class of the same type (a Calculator object), this might look strange but this is just an exercise and not a real world project!