JAVA TECHNOLOGY

(503111)

LAB 6

EXERCISE 1

An exercise on Bean concept, ApplicationContext, ClassPathXmlApplicationContext and spring project configuration in an xml file.

Create a maven project and configure the spring library in an xml file (xml-based configuration). In the xml file, you need to declare 3 different beans of the same class (eg Product.java). Then set an arbitrary value to any value for these beans, but make sure:

- The first bean should be set using the <property> tag
- The second bean should be set up using <constructor-arg> tags
- The third bean can be set up using any of the above two tags, but it must be a singleton.

In the main method, initialize the spring context as a ClassPathXmlApplicationContext object and then load the beans from the context then print their information to the console. You need to write test code in main method to make sure that first two beans are prototype object and third bean is singleton object.

Hints:

- Create a maven project with archtype: maven-archetype-quickstart
- Add a dependency named spring-context version 5.3.20 to the project (pom.xml)
- Create a resources folder for the maven project if it doesn't exist.
- Create class Product.java containing information: id, name, price and description.

- In the resources folder, create an xml file (eg appConfig.xml) to configure beans for the spring project in this file.
- In the main method, initialize a context as ClassPathXmlApplicationContext, load the declared bean in xml and then print this bean information to the console.

EXERCISE 2

An exercise on configuring a spring project with annotations and java code, using annotations like @Configuration and @Bean.

Repeat the previous exercise (in a separate project) with but configure the project using java code and annotations instead of xml file, do not create xml files as in the previous exercise.

Hints:

- Create a maven project and add the same spring-context library as in the previous exercise.
- Create any java class to be the place to configure the beans, this class must be marked with the @Configuration annotation. This class contains methods marked with the @Bean annotation, these methods should return instances of beans in your program.
- Use the @Scope("prototype") or @Scope("singleton") annotation to set scope for beans. If scope is not set, singleton will be used by default.
- In the main method, initialize the context as an instance of the AnnotationConfigApplicationContext class.

EXERCISE 3

An exercise on learning the concept of dependency injection through the @Autowired annotation and setting the bean identifier through the @Qualifier annotation.

Create a spring core project configured using java-based and create classes as described in the class diagram in Figure 1 below.

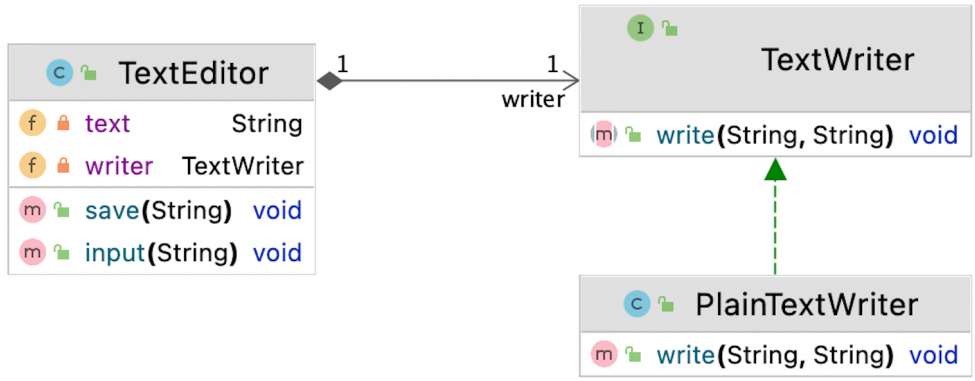

Figure 1

In the figure above, TextWriter is an interface, PlainTextWriter is a class that implements TextWriter and TextEditor is a class that uses TextWriter as a property. PlainTextWriter has a write(fileName, text) method that writes text content to a txt file named fileName. TextWriter plays the role of a text editor with two functions: input() - receiving input text and save() - writing received text content to a file. Inside the save() method there is a call to the TextWriter's write() function to write the content. Looking at this class diagram, we can see that TextEditor is dependent on TextWriter, in other words, TextWriter is a dependency of TextEditor.

Requirements:

- Register beans for two classes: TextWriter and PlainTextWriter
- Set up dependency injection for TextWriter using the @Autowired annotation.
- Initialize context in main method, load TextEditor bean from the context and call the input() and the save() methods respectively for testing.
- Continue adding the PdfTextWriter class that implements TextWriter as described in the class diagram in Figure 2 below.
- After adding PdfTextWriter, the program will crash because now there is an ambiguity between the two subclasses of TextWriter as a dependency of TextEditor. Use the @Qualifier annotation to set the identifier for each of these beans to solve the above problem. Then choose any bean to inject into the TextEditor.

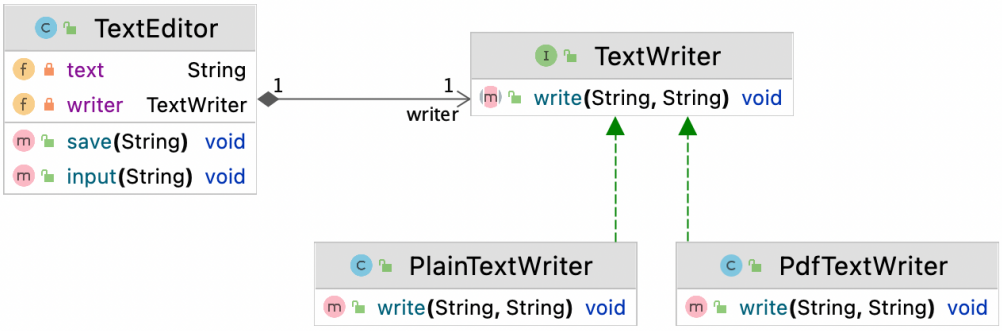

Figure 2

EXERCISE 4

Continue working on the results of exercise 3, using two annotations @Component and @ComponentScan to set up automatic bean 'scanning' mode without having to specifically set up java code in configuration.

EXERCISE 5

Create a spring core project that uses the @PropertySources, @PropertySource and @Value annotations to set the values for the beans from the *.properties file located in the resources directory of the maven project.