Configuration Classes

Overview

In this lab you'll enhances your "online retailer" application from the previous lab. You will define a configuration class to create some beans, and then autowire the beans into components.

IntelliJ starter project

If you're happy to continue where you left off in the previous lab, use the following project:

- **`student\student-online-retailer`**

If you'd prefer a fresh start, use the solution project from the previous lab instead:

- **`solutions\solution-injection-techniques`**

IntelliJ solution project

The solution project for this lab is located here:

- **`solutions\solution-configuration-classes`**

Roadmap

There are 3 exercises in this lab, of which the last exercise is "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Defining a configuration class and creating a bean
2. Autowiring the bean into a component
3. (If time permits) Additional suggestions

Familiarization

Take a look in the **`student`** folder. Notice we've defined a class named **`Transcript`**. Add this class into your IntelliJ project now.

The **`Transcript`** class is already complete. It holds a linked list of messages, where new messages are added at the end. If the list has already reached its maximum size, the first element if dropped to make space. In this way, the list always holds the most recent messages.

Note that the constructor takes 2 parameters:

- **`maxSize`** specifies the maximum allowable size of the list
- **`verbose`** specifies whether messages should be in verbose format (i.e., with date/time)

Exercise 1: Defining a configuration class and creating a bean

Define a configuration class named **`Config`**. In the configuration class, define a **`@Bean`** method to create a **`Transcript`** bean. Note the following points:

- The bean should have **`"prototype"`** scope. This means any component that autowires the bean will get its own individual instance, with its own individual list of messages.
- Give the bean method a name such as **`verboseTranscript()`**.
- In the method, create and return a **`Transcript`** object that holds up to 5 messages in verbose format. Return the object from the method.

Exercise 2: Autowiring the bean into a component

Autowire a **`Transcript`** bean into the **`CartServiceImpl`** component class, so that the **`CartServiceImpl`** component can keep a transcript of shopping activity (i.e., it can keep track of when the user adds/removes items to the cart).

Enhance the **`CartServiceImpl`** class as follows:

- In the **`addItemToCart()`** method, add a message to the transcript indicating the item id and the quantity that has been added to the cart.
- In the **`removeItemFromCart()`** method, add a message to the transcript indicating the item id that has been removed from the cart.
- Define a new method named **`displayRecentActivity()`**, to display the messages in the transcript.

Now go to the **`Application`** class and add some code to add/remove lots of items to the shopping cart. Then call **`displayRecentActivity()`** and verify that the 5 most recent messages in the transcript are displayed.

Exercise 3 (If time permits): Additional suggestions

In the **`Config`** class, define another **`@Bean`** method named **`briefTranscript()`**. In the method, create and return a **`Transcript`** object that holds up to 5 messages in *brief* format (i.e., not *verbose* format). Return the object from the method.

Try to run the application again. You'll get an exception when Spring Boot tries to create the **`CartServiceImpl`** bean, because it doesn't know which **`Transcript`** bean to autowire (there are 2 Transcript beans available – **`verboseTranscript`** and **`briefTranscript`**).

Fix this problem, either via a **`@Qualifier`** annotation or a **`@Primary`** annotation. Run the application again and verify the **`CartServiceImpl`** transcript messages are displayed correctly (i.e., either in verbose format or brief format, as appropriate).