# Beans and Dependency Injection

## Overview

In this lab you'll start to implement an "online retailer" Spring Boot application. The application will enable customers to view items on sale, and to add items to their shopping cart.

You'll define two Spring component classes in this lab:

- A **service** bean that lets a customer see what items are available in a catalog, and to add items to their shopping cart (via the repository bean, see below).
- A **repository** bean that provides persistence for the customer's shopping cart. Initially, you'll store items in memory for simplicity. Later in the course, you'll implement proper persistence by storing this info in a database.

## IntelliJ projects

Starter project: **`student\student-online-retailer`**

Solution project: **`solutions\solution-beans-dependencyinjection`**

## Roadmap

There are 4 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 repository bean
2. Defining a service bean
3. Writing client code
4. (If time permits) Additional suggestions

## Familiarization

Take a look in the **`student`** folder. We've provided an IntelliJ project to get you started (plus some other files for later in the course). Open the project in IntelliJ now:

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

Take a look at the following 3 classes, which are already complete:

- **`Item`** class – Represents an item in the shop catalog. Each item will have a unique id.
- **`Catalog`** class – Spring component class, holds a map of **`Item`** objects indexed by id.
- **`Application`** class – This is the usual Spring Boot application class.

Now take a look at the following 2 interfaces. You will define Spring Boot component classes to implement both of these interfaces during this lab:

- **`CartRepository`** interface – Specifies methods for the *repository* bean.
- **`CartService`** interface – Specifies methods for the *service* bean.

## Exercise 1: Defining a repository bean

Define a Spring component class named **`CartRepositoryImpl`**, which implements the **`CartRepository`** interface. The class will store info about the user's shopping cart (in memory initially, for simplicity).

We suggest defining an instance variable of type **`HashMap<Integer,Integer>`**. In this map, each entry will hold:

- A key, representing an item id.
- A value, indicating the quantity of that item in the user's cart.

Implement the **`add()`**, **`remove()`**, and **`getAll()`** methods accordingly, to add/remove/query items in the map. Note the following points:

- In the **`add()`** method, check to see if the user already has that item id in the cart. If so, just increment the quantity for that item (rather than having two separate map entries).
- In the **`remove()`** method, remove the item completely from the cart. For example, if the customer has 20,000 Swansea City shirts in the cart, and they remove that item, then the **`remove()`** method should remove the map entry completely (rather than decrementing the count to 19,999).

## Exercise 2: Defining a service bean

Define a Spring component class named **`CartServiceImpl`**, which implements the **`CartService`** interface. The class will allow the customer to choose items from a catalog and add them to his/her cart.

Autowire two beans into the **`CartServiceImpl`** class:

- A **`Catalog`** bean. This bean will be the source of catalog data for your service bean.
- A **`CartRepository`** bean, which provides persistence to store the customer's cart.

Now implement the following methods in the **`CartServiceImpl`** class:

- **`addToCart()`**
  The method takes an item id and a quantity as parameters. The method should check the item id is valid (i.e. it exists in the catalog); if so, add the specified item/quantity to the customer's cart (i.e. call the **`add()`** method on the repository bean).
- **`removeFromCart()`**
  You can just call the **`remove()`** method on the repository bean.
- **`getAllItemsInCart()`**
  You can just call the **`getAll()`** method on the repository bean.
- **`calculateCartCost()`**
  This method should get all items from the cart (via the repository's **`getAll()`** method), multiply the cost of each item by the quantity, and return the total cost.

## Exercise 3: Writing client code

In the **`Application`** class, write some client code to get the **`CartService`** bean and exercise its methods to add/remove items in the customer's cart, get all items in the cart, and calculate the total cost. Run the application and verify everything is hunky-dory.

## Exercise 4 (If time permits): Additional suggestions

Explore some of the techniques discussed during the chapter. For example:

- Defining beans with different scopes
- Defining multiple bean implementation classes and resolving autowiring ambiguities
- Autowiring collections