**Lab 4**

**Context and dependency injection**

In the preceding labs you ‘hard-wired’ the dependencies that exist in the shopping and product service. In this lab you will make changes to the dependencies, so that they are no longer ‘hard wired’.

**Refactoring the ProductService**

* Open your product service from lab 3, if it is closed.
* Open the file ProductRepistoryImpl.java and annotate the class as follows:

@Repository

Public class ProductRepositoryImpl {

[leave the code as it is]

}

* Extract an interface from the ProductRepositoryImpl.java class and name it ProductRepository. This can be done for you by tooling available in your IDE. In IntelliJ right click the ProductRepositoryImpl -> Refactor -> Extract -> Interface ->

Fill in the name for the interface and select the 4 methods.

package nl.groothandel.service.domain;

import java.util.Map

public interface ProductRepository {

void setProducts(Map<String,Product> products);

Map<String,Product> getProducts();

boolean addProduct(String id, Product product);

boolean updateProduct(String id, Product product);

}

* In the ProductsController.java file replace the hard wired ProductRepositoryImpl for just a property with the interface type:

~~ProductRepositoryImpl productRepository = new ProductRepositoryImpl();~~

ProductRepository productRepository;

Now, you can probably will see a message like the following “Could not autowire. No beans of ‘ProductRepository’ type found. You will fix this in the next bullet.

* In the ProductserviceApplication.java file replace the line with the @SpringBootApplication annotation with the following:

@SpringBootApplication(scanBasePackages = {“nl.groothandel.service”})

The scanBasePackages attribute is read by the Spring container, so it knows where to look/scan for beans (@Component, @Repository, @Service annotated Java classes) and to inject them. Sub folders from this base package are scanned as well, you do not need to specify them separately.

* Restart the product service and perform some tests to see thing are still working.

**Refactoring the ShoppingService**

* Open your shopping service from lab 3, if it is closed.
* In the file ShoppingController.java there are 4 lines for instantiating a rest template:

RestTemplate restTemplate = new RestTemplate();

These are hard wires, which you are going to replace.

* At the class level add a property referencing a RestTemplate instance, which will be autowired.

@Autowired

private RestTemplate restTemplate;

* Comment out the 4 lines mentioned above, that create a hard wired RestTemplate.

Do not remove them, as you will be using these below.

* If the method retrieveDrinks is defined as static, remove the static specifier.

Static methods cannot be autowired.

* In the ShoppingserviceApplication add a bean which exposes a RestTemplate:

@Bean

RestTemplate restTemplate() {

return new RestTemplateBuilder().build();

}

* In the ShoppingserviceApplication add ‘(scanBasePackages= {“nl.sjop.service”})’ to the @SpringBootApplication annotation (without the single quotes).
* Restart the shopping service and test if things still work.
* Another way to go, is to replace the 4 lines with the following:

RestTemplate restTemplate = restTemplateBuilder().build;

and replace the autowired RestTemplate with a RestTemplateBuilder :

@Autowired

private RestTemplateBuilder restTemplateBuilder;

. Also remove the bean that exposes the RestTemplate from the ShoppingserviceApplication.

* Restart and test the shopping service again.