Exercise 11.2 – Dependency Injection

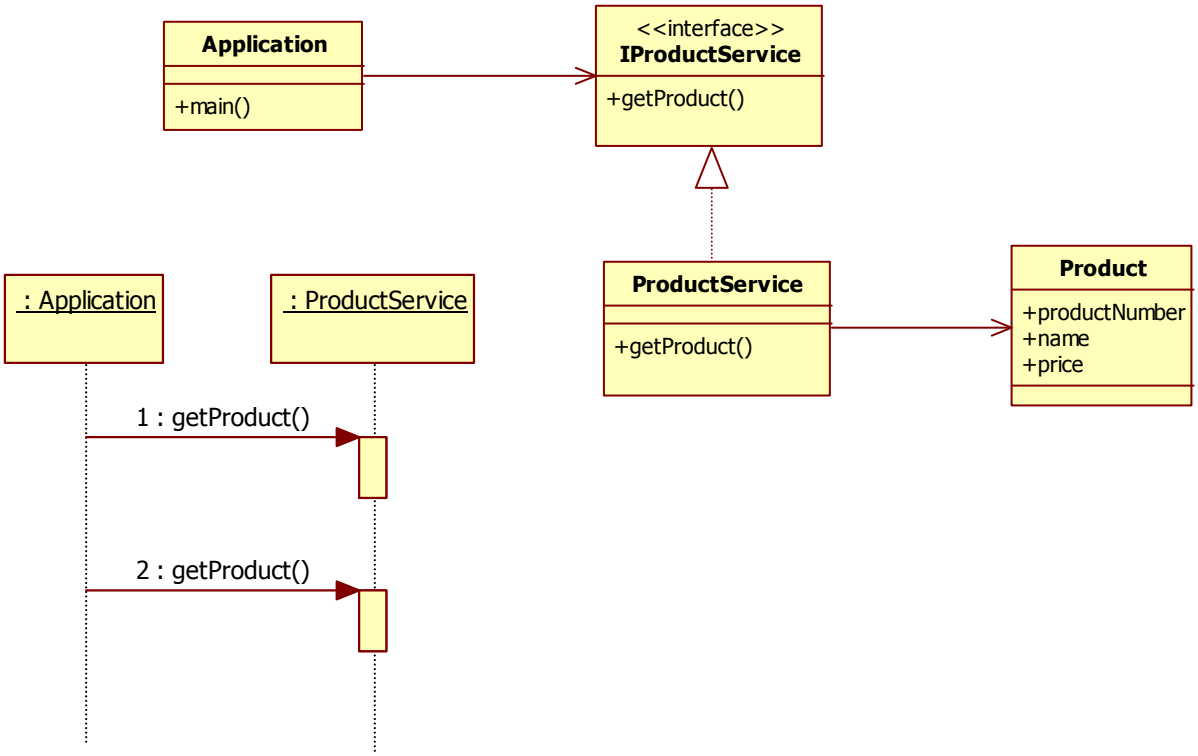
The Setup:

The main objective of this exercise is to work with the springconfig.xml file, configuring Spring beans and use dependency injection.

Start this project by opening **C:\CS544\exercises\exercise11.2**\. and updating its pom.xml file with the spring dependencies.

The Application:

The application provided is a very simple application that has a ProductService class, which implements the IProductService interface and contains a list of Products. Application.java calls the getProduct() method on ProductService.



You can run the file **App.java** by right-clicking on it and selecting **Run File** which should provide the following output:

```
productnumber=423  ,name=Plasma TV ,price=992.55
productnumber=239  ,name=DVD player ,price=315.0.
```

The Exercise:

a) Change the application in such way that **App.java** no longer instantiates **ProductService** but instead retrieves this object from the Spring context. In other words, change the following code:

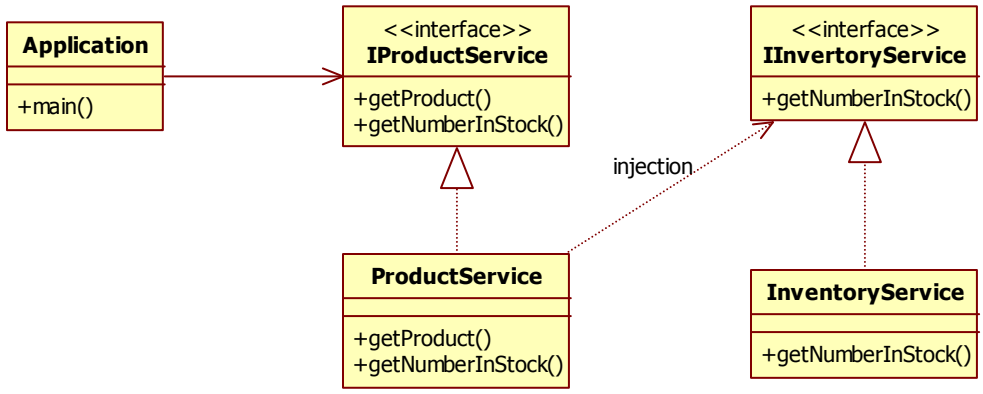
```
IProductService productService = new ProductService();
```

to the following lines of code:

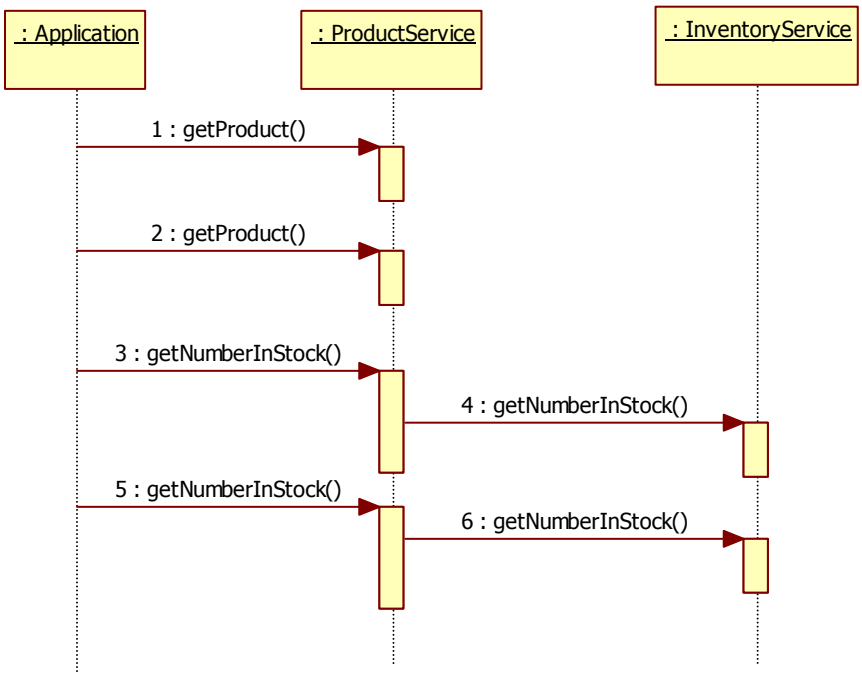
```
ApplicationContext context = new
    ClassPathXmlApplicationContext("springconfig.xml");
IProductService productService =
    context.getBean("productService", IProductService.class);
```

Of course, this will only work if we configure a bean with id=productService in **springconfig.xml**. Run the application and check that everything works correctly.

b) Now we want to add an **InventoryService** object, and inject this InventoryService into the ProductService using Spring dependency injection.



We will also want to add a **getNumberInStock()** method to the ProductService that calls the getNumberInStock() method of InventoryService.



First, create a new interface called IInventoryService and then a class called InventoryService using the code below:

```
public interface IInventoryService {
     public int getNumberInStock(int productNumber);
}
```

```
public class InventoryService implements IInventoryService{

   public int getNumberInStock(int productNumber) {
         return productNumber-200;
   }

}
```

Then, update IProductService and ProductService as shown in the code below.

```
public interface IProductService {
    public Product getProduct(int productNumber);
    public int getNumberInStock(int productNumber);
}
```

```
public class ProductService implements IProductService{
   private IInventoryService inventoryService;
   private Collection productList= new ArrayList();

   public ProductService(){
         productList.add(new Product(234,"LCD TV", 895.50));
         productList.add(new Product(239,"DVD player", 315.00));
         productList.add(new Product(423,"Plasma TV", 992.55));
   }
   public Product getProduct(int productNumber) {
         for (Product product : productList) {
               if (product.getProductNumber() == productNumber)
                     return product;
         }
         return null;
   }
   public int getNumberInStock(int productNumber) {
         return inventoryService.getNumberInStock(productNumber);
   }
   public void setInventoryService(IInventoryService inventoryService) {
         this.inventoryService = inventoryService;
   }

}
```

Make sure that the inventoryService object gets properly injected into productService by adding the needed XML configuration to **springconfig.xml**.

To test if this works, add the following two lines to **Application.java**:

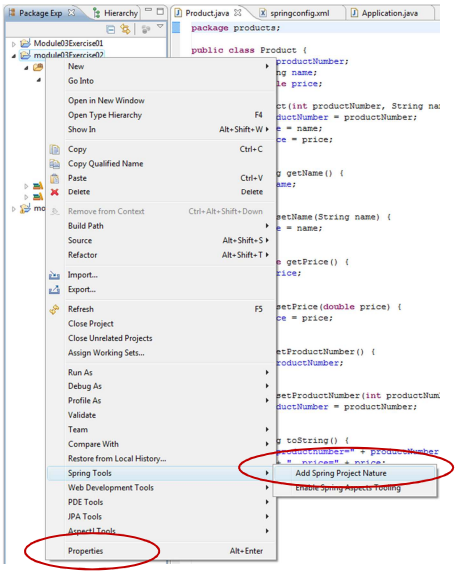
```
System.out.println("we have " + productService.getNumberInStock(423)
         + " product(s) with productNumber 423 in stock");
System.out.println("we have " + productService.getNumberInStock(239)
         + " product(s) with productNumber 239 in stock");
```

Then, run **Application.java** to make sure it gives the following output:

```
productnumber=423  ,name=Plasma TV ,price=992.55
productnumber=239  ,name=DVD player ,price=315.0
we have 223 product(s) with productNumber 423 in stock
we have 39 product(s) with productNumber 239 in stock
```

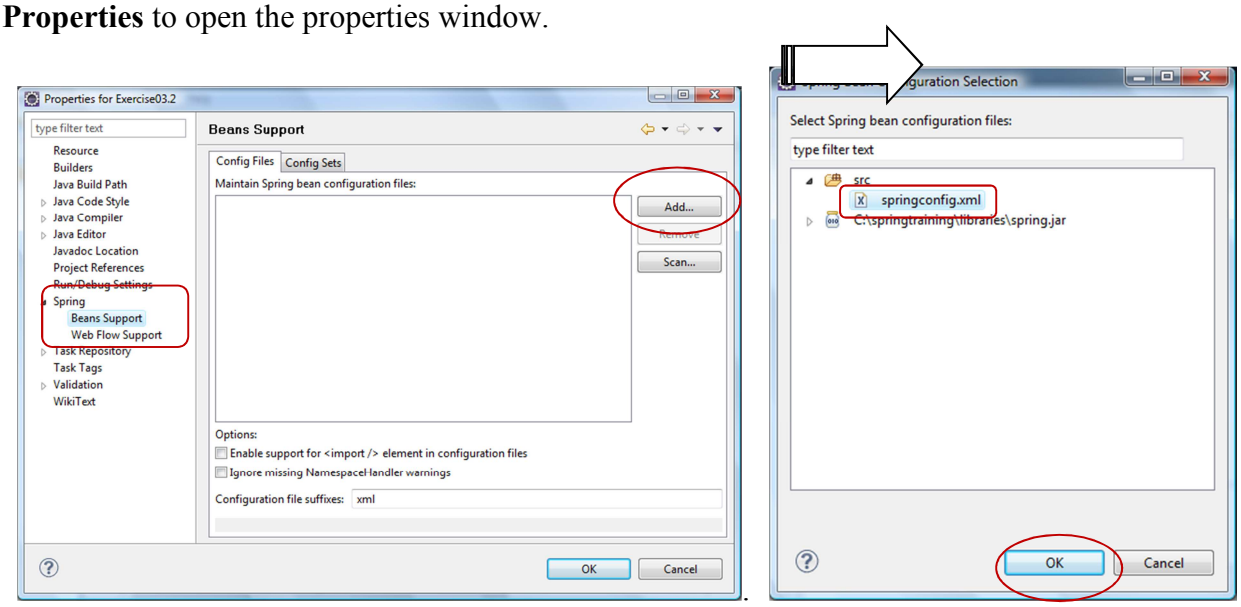
Spring Tool Suite (STS) Optional Exercise:

If you open your maven project in STS you can use it to show a graphical view of the configured Spring beans. Start by enabling and configuring the **Spring Project Nature** on your project:



Right-click the project **Exercise03.2** in the package explorer and select **Spring Tools → Add Spring Project Nature**.

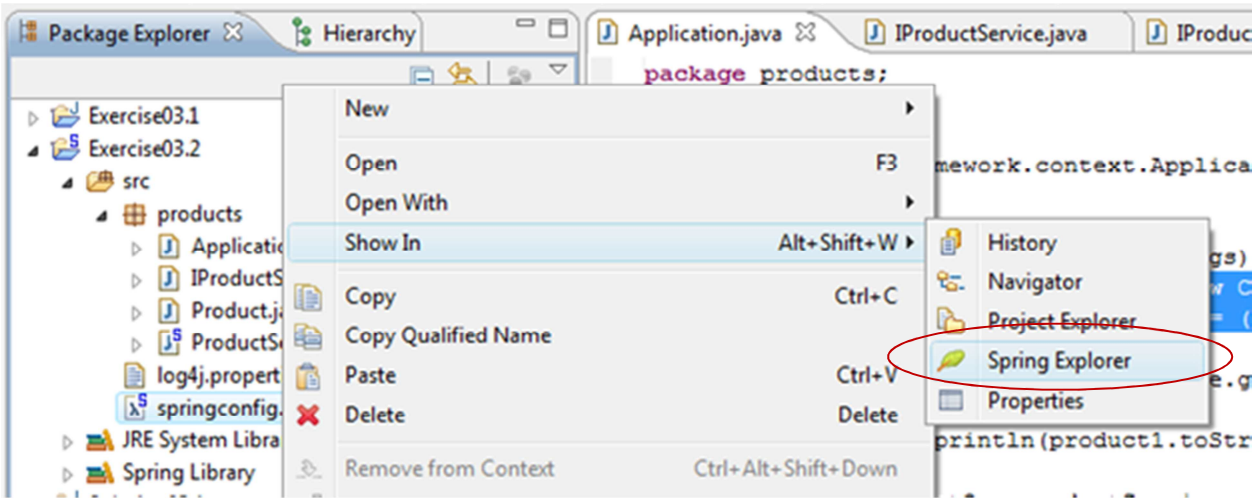
Once you have added the Spring Project Nature, right-click on the project again, and select **Properties** to open the properties window.



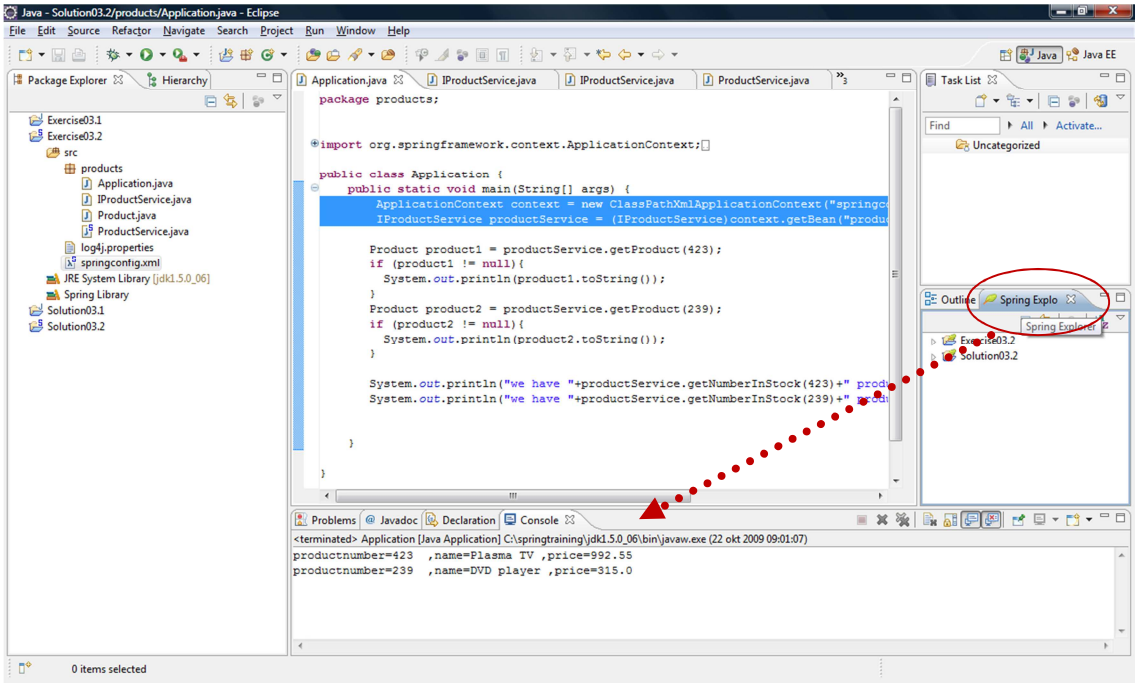
Select the **Spring → Beans Support** section on the left hand side to open the Beans Support page and click on the **Add** to open the Spring Bean Configuration Selection window.

Select your **springconfig.xml** file on the **Spring Bean Configuration Selection** window, (usually found under the **src** folder) and select **OK**. Once your springconfig.xml has been added, you can close the Properties window by clicking **OK** there as well.

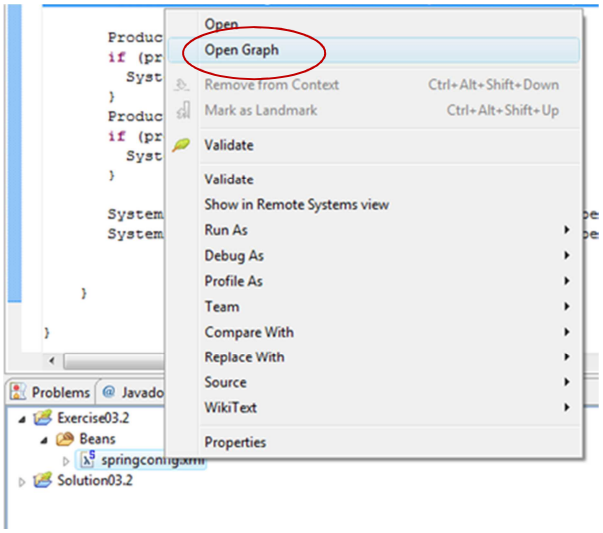
Eclipse can now generate a visual graph of your mapped beans.

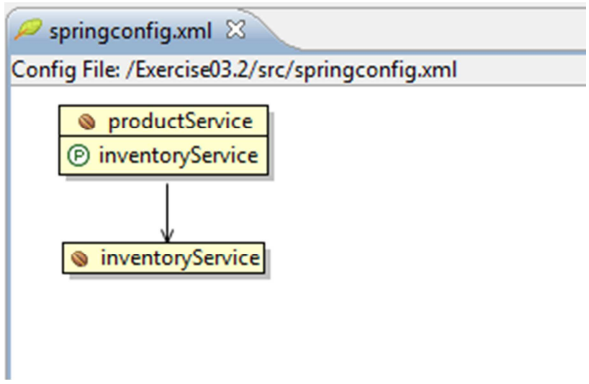


Right-click on your **springconfig.xml** file and select **Show In → Spring Explorer**.



The Spring Explorer tab sometimes opens on the right-hand side near the outline section. Feel free to drag the tab to another location, for example next to the console tab.





Then, in the Spring Explorer right-click on your springconfig.xml file and select **Open Graph**. The graph shows that the inventoryService bean is injected into the productService bean.