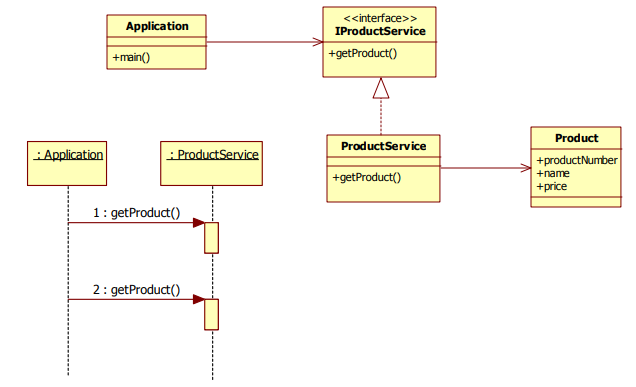
## Exercise SDI.2 – Dependency Injection



### Change the application in such way that App.java no longer instantiates ProductService but instead retrieves this object from the Spring context.

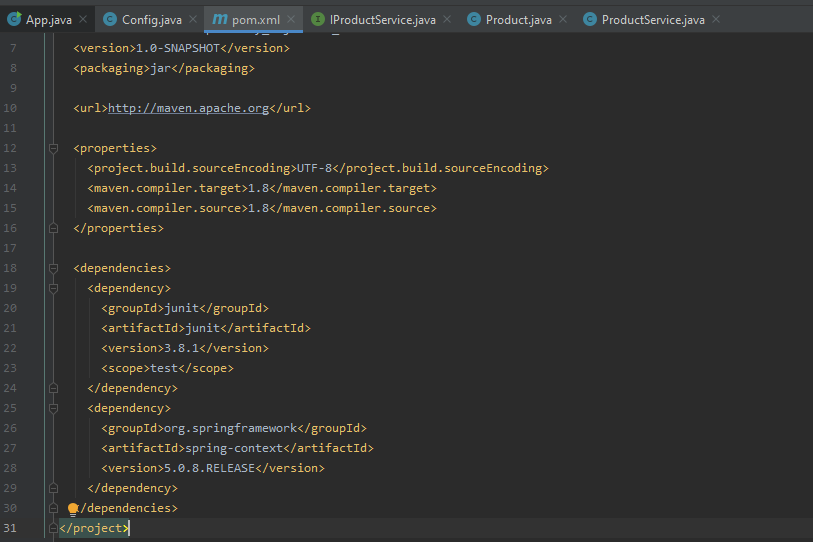
#### Solution

For that we need

1. Spring-core dependencies, which means to add dependency in pom.xml
2. Config.class configuration file using notation
3. Use annotations for ProductService class

##### pom.xml

<**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-context</**artifactId**>  
 <**version**>5.0.8.RELEASE</**version**>  
</**dependency**>



##### Config.java

**package** edu.mum.cs544;  
  
**import** org.springframework.context.annotation.ComponentScan;  
**import** org.springframework.context.annotation.Configuration;  
@Configuration  
@ComponentScan(**"edu.mum.cs544"**)  
**public class** Config {  
}

##### ProductService.java

**package** edu.mum.cs544;  
  
**import** org.springframework.stereotype.Component;  
**import** java.util.\*;  
  
@Component  
**public class** ProductService **implements** IProductService {  
 **private** Collection<Product> **productList** = **new** ArrayList<Product>();  
  
 **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**;  
 }  
  
}

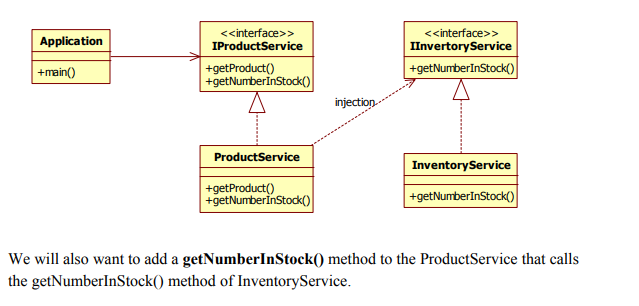
##### AppA.java

**package** edu.mum.cs544;  
  
**import** org.springframework.context.ApplicationContext;  
**import** org.springframework.context.annotation.AnnotationConfigApplicationContext;  
  
**public class** AppA {  
 **public static void** main(String[] args) {  
 *//IProductService productService = new ProductService();* ApplicationContext context = **new** AnnotationConfigApplicationContext(Config.**class**);  
 IProductService productService = context.getBean(**"productService"**, IProductService.**class**);  
  
 Product product1 = productService.getProduct(423);  
 **if** (product1 != **null**) {  
 System.***out***.println(product1.toString());  
 }  
 Product product2 = productService.getProduct(239);  
 **if** (product2 != **null**) {  
 System.***out***.println(product2.toString());  
 }  
 }  
}

#### Output



### We want to add an InventoryService object, and inject this InventoryService into the ProductService using Spring dependency injection.



Configure the Inventory Service to be a spring bean, and make sure that it is properly injected into the ProductService.

#### Solution

##### Dependency injection is done through springbean.xml

*<?***xml version="1.0" encoding="UTF-8"** *?>*<**beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans  
http://www.springframework.org/schema/beans/spring-beans.xsd"**>  
 <**bean id="productService" class="edu.mum.cs544.ProductService"**>  
 <**property name="inventoryService" ref="inventoryService"**/>  
 </**bean**>  
 <**bean id="inventoryService" class="edu.mum.cs544.InventoryService"**/>  
</**beans**>

1. Here inventoryService is injected into ProductService as a property. For that inventoryService must have setter method in ProductService class.

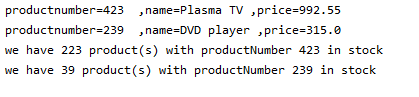
##### ProductService.java

**package** edu.mum.cs544;  
  
**import** org.springframework.stereotype.Component;  
**import** java.util.\*;  
  
@Component  
**public class** ProductService **implements** IProductService {  
 **private** Collection<Product> **productList** = **new** ArrayList<Product>();  
 **private** IInventoryService **inventoryService**;  
 **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**;  
 }  
 @Override  
 **public int** getNumberInStock(**int** productNumber) {  
 **return inventoryService**.getNumberInStock(productNumber);  
 }  
 **public void** setInventoryService(IInventoryService inventoryService) {  
 **this**.**inventoryService** = inventoryService;  
 }  
}

##### AppB.java

**public class** AppB {  
 **public static void** main(String[] args) {  
 ApplicationContext context = **new** ClassPathXmlApplicationContext(**"springbean.xml"**);  
 IProductService productService = context.getBean(**"productService"**, ProductService.**class**);  
  
 Product product1 = productService.getProduct(423);  
 **if** (product1 != **null**) {  
 System.***out***.println(product1.toString());  
 }  
 Product product2 = productService.getProduct(239);  
 **if** (product2 != **null**) {  
 System.***out***.println(product2.toString());  
 }  
 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"**);  
  
 }  
}

#### Output:



## Exercise SDI.3 – Dependency Injection using Lists

The purpose of this exercise is for you to use the more advanced list configuration feature of dependency injection.

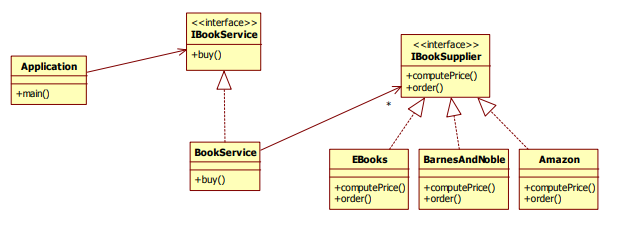


Figure 1Class diagram

The application buys 3 books through the **IBookService** implemented by **BookService**. In the **buy** method, the **BookService** checks each of its **IBookSuppliers**, finding the cheapest one and ordering the book from there.

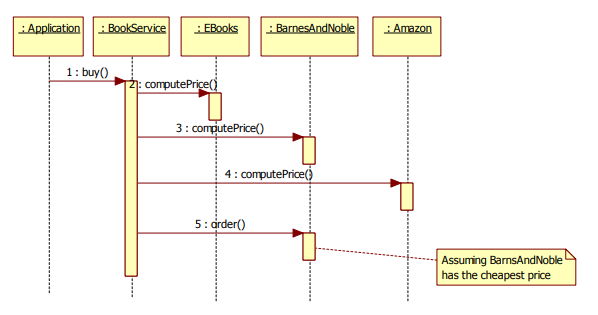


Figure 2Sequence diagram