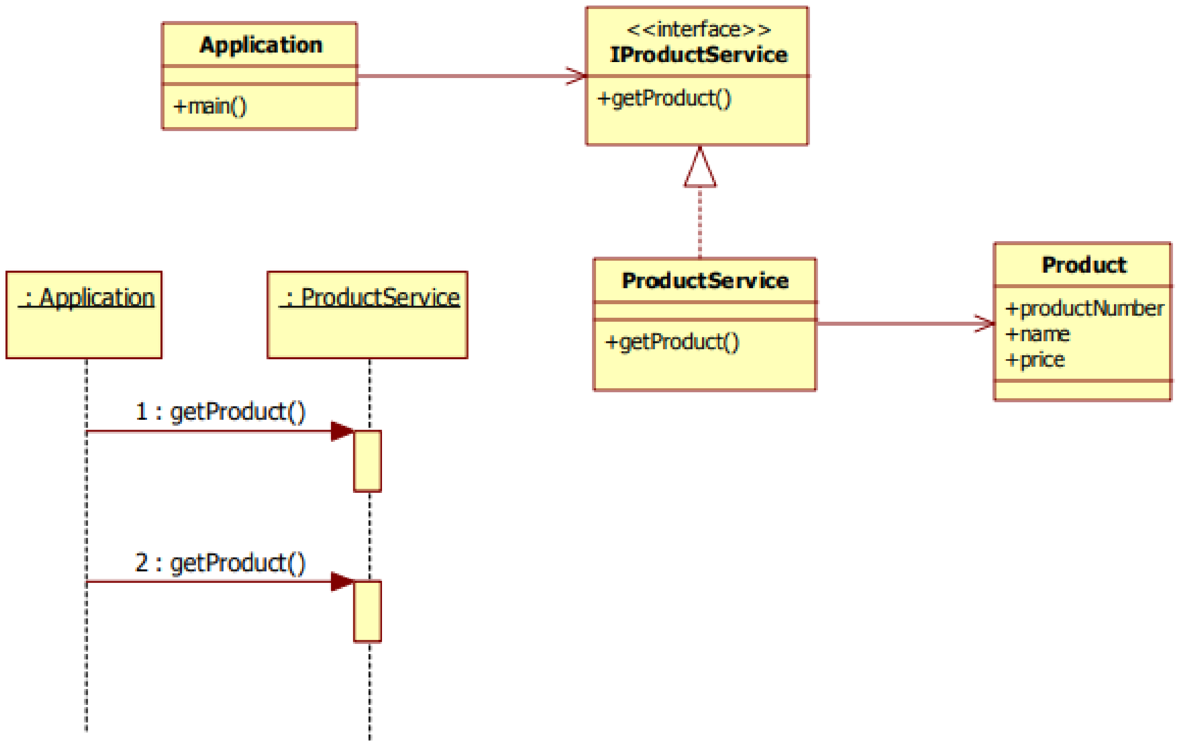
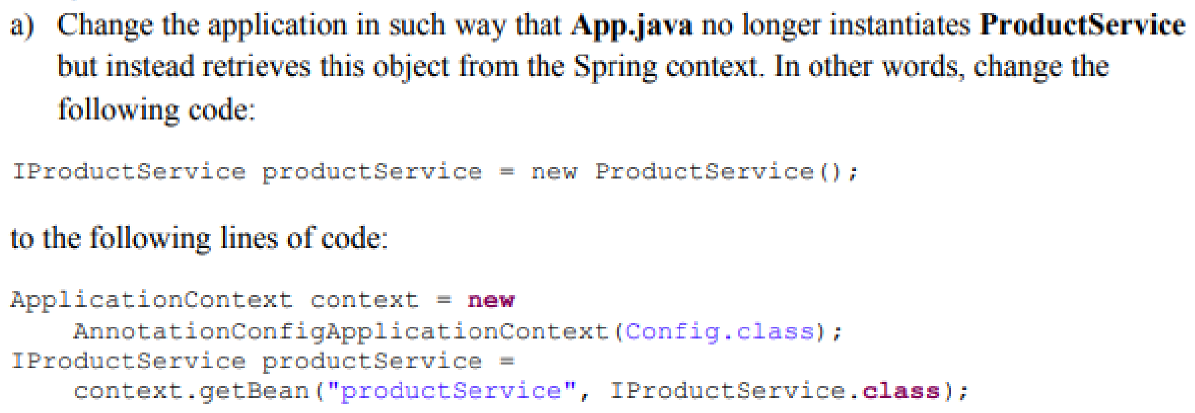
Contents

Exercise SDI.2 – Dependency Injection



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



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
    AnnotationConfigApplicationContext(Config.class);
IProductService productService =
    context.getBean("productService", IProductService.class);
```

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

1. pom.xml: Add spring dependency

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>cs544</groupId>
  <artifactId>W1D2-Dependency_Injection_2</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.target>1.8</maven.compiler.target>
    <maven.compiler.source>1.8</maven.compiler.source>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.0.8.RELEASE</version>
    </dependency>
  </dependencies>
</project>
```

2. Config.java

```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 {
}
```

3. ProductService.java

```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;
   }

}
```

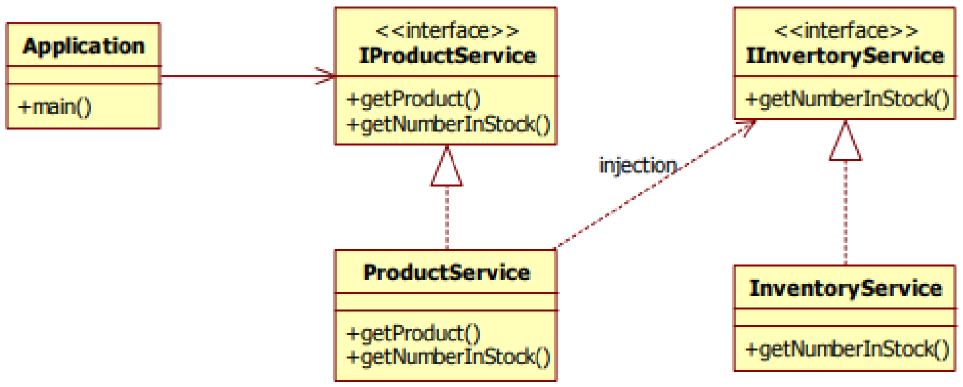
4. AppA.java: change the way of creating productService

```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

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

B. 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.

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

Solution

1. 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>
```

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

1. 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;
   }
}
```

2. 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:

```
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
```

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.

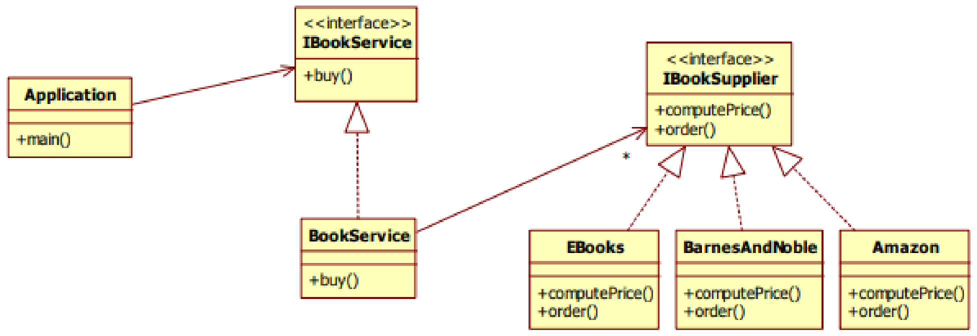
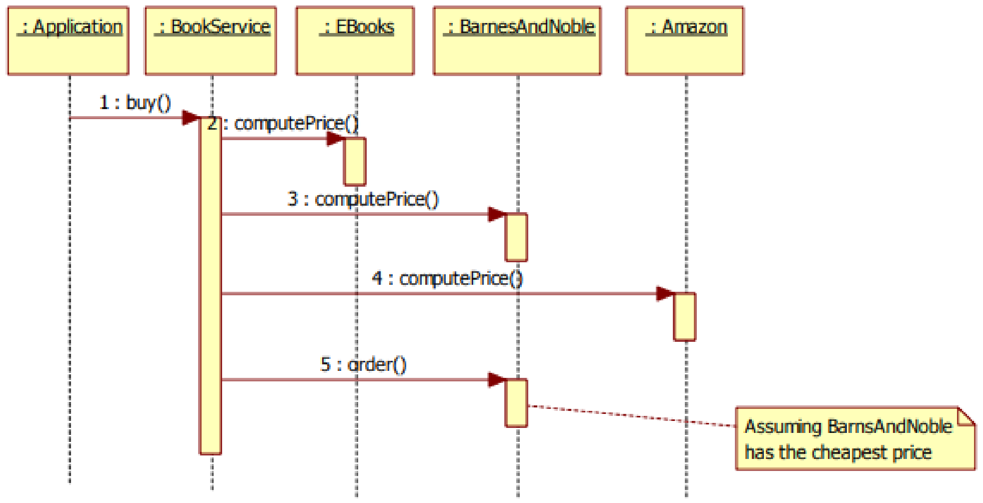


Figure 1Class diagram

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.



Problem with current solution

BookService.java hardcoded

The downside of this application is that the BookService is hardcoded with Amazon, EBooks and Barnes & Noble as IBookSuppliers. If we should want to add another book supplier, we would have to go in and change the code.

```
package edu.mum.cs544;
import java.util.*;
public class BookService implements IBookService {
   public List<IBookSupplier> suppliers = new ArrayList<IBookSupplier>();
   public BookService() {
      IBookSupplier amazon = new Amazon();
      IBookSupplier barnesandnoble = new BarnesAndNoble();
      IBookSupplier ebooks = new EBooks();

      suppliers.add(amazon);
      suppliers.add(barnesandnoble);
      suppliers.add(ebooks);
   }
   public void buy(Book book) {
      double lowestPrice = 0;
      IBookSupplier cheapestSupplier = null;
      // find the cheapest supplier
      for (IBookSupplier supplier : suppliers) {
         double price = supplier.computePrice(book.getIsbn());
         if (cheapestSupplier == null) {
            cheapestSupplier = supplier;
            lowestPrice = price;
         } else {
            if (price < lowestPrice) {
               cheapestSupplier = supplier;
               lowestPrice = price;
            }
         }
      }
      // buy with the cheapest supplier
      if (cheapestSupplier != null) {
         cheapestSupplier.order(book);
      }
```

BookService is hardcoded with Amazon, EBooks and Barnes & Noble as IBookSuppliers. If we should want to add another book

```
   }
}
```

App.java

```java
package edu.mum.cs544;

public class App
{
   public static void main(String[] args) {

      IBookService bookService = new BookService();
      bookService.buy(new Book("123433267", "Harry Potter and the Order of the Phoenix", "J.K. Rowling"));
      bookService.buy(new Book("888832678", "Harry Potter and the Sorcerer's Stone" , "J.K. Rowling"));
      bookService.buy(new Book("999923156", "Harry Potter and the Goblet of Fire" ,"J.K. Rowling"));

   }
}
```

A. Change the application in such a way that the List is injected into the **BookService** (already filled with IbookSuppliers). In other words, remove the code that instantiates them with new . You will also have to update App.java to retrieve the BookService from Spring.

Solution

Remove hardcoded ibookSuppliers and set property using bean property. So that need to define setter method for suppliers.

1. Define springsbean.xml
   a. Inject suppliers
   b. Set suppliers property in springbean.xml
2. Define setter for suppliers in BookService class.
3. Get bookService from spring bean in app.

1.pom.xml: Add spring dependency

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>cs544</groupId>
    <artifactId>W1D2-Dependency_Injection_3</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.target>1.8</maven.compiler.target>
        <maven.compiler.source>1.8</maven.compiler.source>
    </properties>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.0.8.RELEASE</version>
        </dependency>
    </dependencies>
</project>
```

1. Springbean.xml : Define springbean.xml and define bookService bean with suppliers property

```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="bookService" class="edu.mum.cs544.BookService">
        <property name="suppliers">
            <list>
                <ref bean="amazon" />
                <ref bean="barnesAndNoble" />
                <ref bean="ebooks" />
            </list>
        </property>
    </bean>
    <bean id="amazon" class="edu.mum.cs544.Amazon"></bean>
    <bean id="barnesAndNoble" class="edu.mum.cs544.BarnesAndNoble"></bean>
    <bean id="ebooks" class="edu.mum.cs544.EBooks"></bean>
</beans>
```

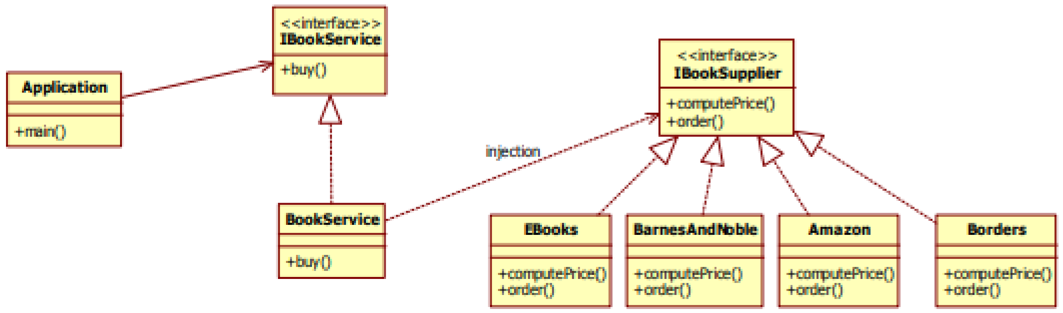
2. App.java : Get bookService from Application context bean in App class.

```java
package edu.mum.cs544;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App
{
   public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("springbean.xml");
      IBookService bookService = context.getBean("bookService", BookService.class);
      bookService.buy(new Book("123433267", "Harry Potter and the Order of the Phoenix", "J.K. Rowling"));
      bookService.buy(new Book("888832678", "Harry Potter and the Sorcerer's Stone" , "J.K. Rowling"));
      bookService.buy(new Book("999923156", "Harry Potter and the Goblet of Fire" ,"J.K. Rowling"));
   }
}
```

Output

```
Amazon charges $18.206081212694755 for book with isbn 123433267
Barnes&Noble charges $0.3024641264820671 for book with isbn 123433267
EBooks charges $12.665779402971845 for book with isbn 123433267
Book with isbn = 123433267 is ordered from Barnes&Noble
Amazon charges $23.516927795842683 for book with isbn 888832678
Barnes&Noble charges $42.39057781765773 for book with isbn 888832678
EBooks charges $2.3711777887830108 for book with isbn 888832678
Book with isbn = 888832678 is ordered from EBooks
Amazon charges $28.434244498546004 for book with isbn 999923156
Barnes&Noble charges $43.55396354587264 for book with isbn 999923156
EBooks charges $16.252521881731106 for book with isbn 999923156
Book with isbn = 999923156 is ordered from EBooks
```

B. Add the Borders BookSupplier to the list while only changing configuration (no business related Java code).



Solution

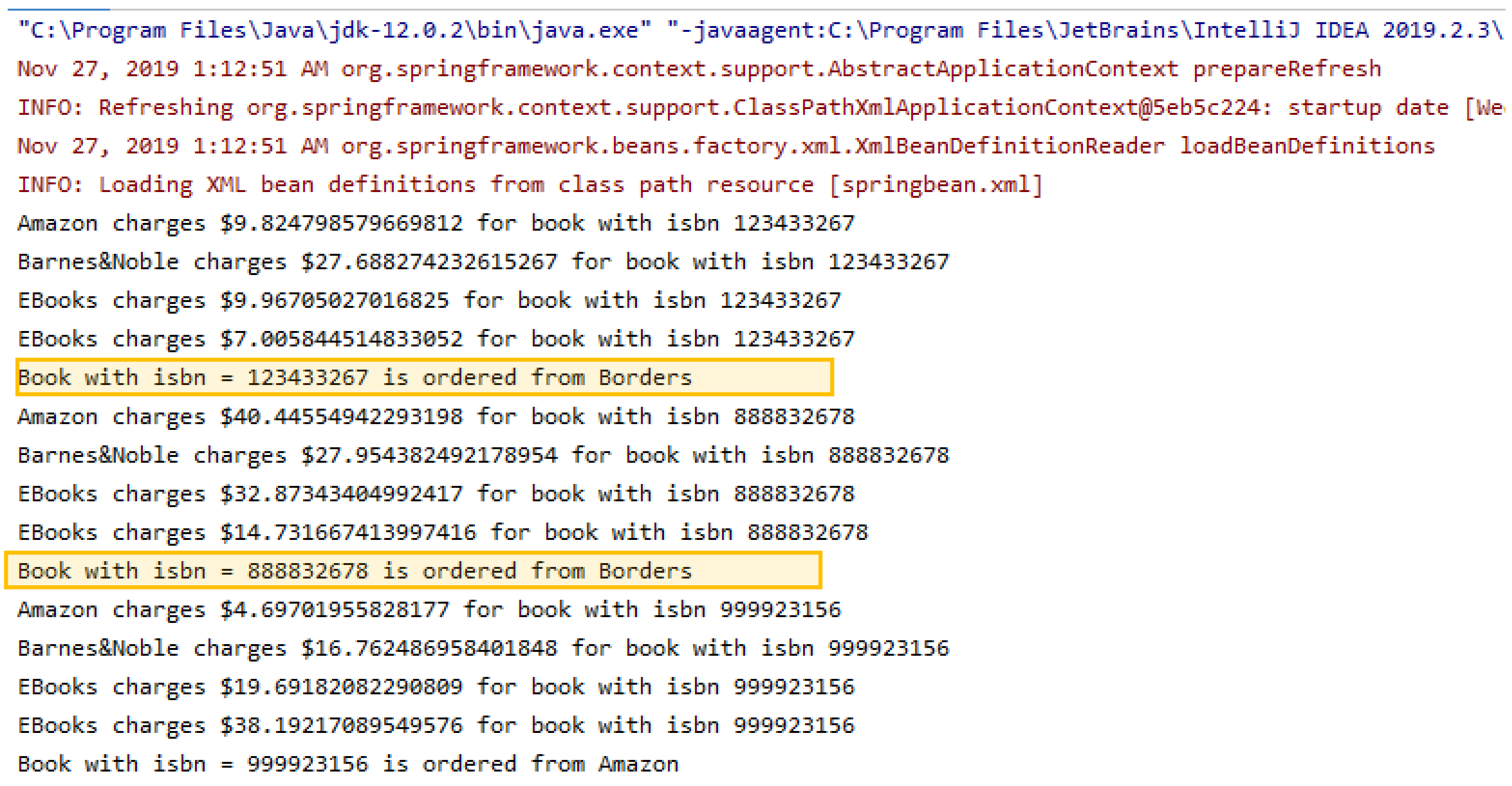
1. Borders.java: Add the class by implementing IbookSupplier

```java
package edu.mum.cs544;
public class Borders implements IBookSupplier {
    @Override
    public double computePrice(String isbn) {
        double price = Math.random() * 45;
        System.out.println("EBooks charges $" + price + " for book with isbn "
                + isbn);
        return price;
    }
    @Override
    public void order(Book book) {
        System.out.println("Book with isbn = " + book.getIsbn()
                + " is ordered from Borders");
    }
}
```

2. Springbean.xml: Add Borders to bean in configuration

```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="bookService" class="edu.mum.cs544.BookService">
        <property name="suppliers">
            <list>
                <ref bean="amazon" />
                <ref bean="barnesAndNoble" />
                <ref bean="ebooks" />
                <ref bean="borders" />
            </list>
        </property>
    </bean>
    <bean id="amazon" class="edu.mum.cs544.Amazon"></bean>
    <bean id="barnesAndNoble" class="edu.mum.cs544.BarnesAndNoble"></bean>
    <bean id="ebooks" class="edu.mum.cs544.EBooks"></bean>
    <bean id="borders" class="edu.mum.cs544.Borders"></bean>
</beans>
```

Output



```
"C:\Program Files\Java\jdk-12.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2019.2.3\
Nov 27, 2019 1:12:51 AM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@5eb5c224: startup date [We
Nov 27, 2019 1:12:51 AM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [springbean.xml]
Amazon charges $9.824798579669812 for book with isbn 123433267
Barnes&Noble charges $27.688274232615267 for book with isbn 123433267
EBooks charges $9.96705027016825 for book with isbn 123433267
EBooks charges $7.005844514833052 for book with isbn 123433267
Book with isbn = 123433267 is ordered from Borders
Amazon charges $40.44554942293198 for book with isbn 888832678
Barnes&Noble charges $27.954382492178954 for book with isbn 888832678
EBooks charges $32.87343404992417 for book with isbn 888832678
EBooks charges $14.731667413997416 for book with isbn 888832678
Book with isbn = 888832678 is ordered from Borders
Amazon charges $4.69701955828177 for book with isbn 999923156
Barnes&Noble charges $16.762486958401848 for book with isbn 999923156
EBooks charges $19.69182082290809 for book with isbn 999923156
EBooks charges $38.19217089549576 for book with isbn 999923156
Book with isbn = 999923156 is ordered from Amazon
```