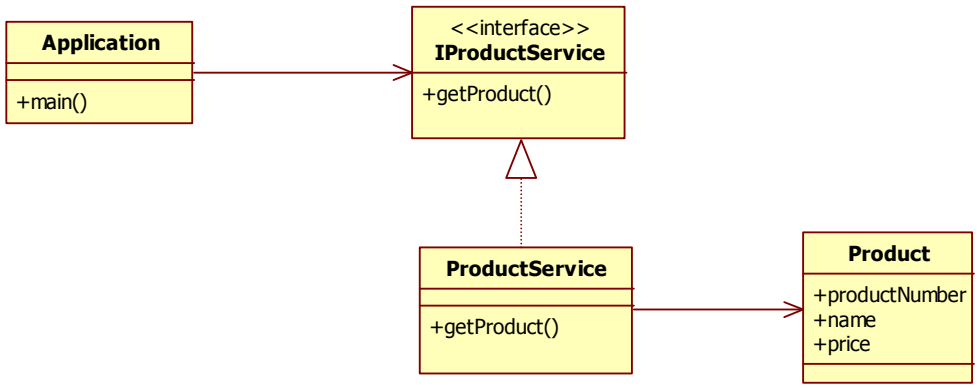
Spring DI lab

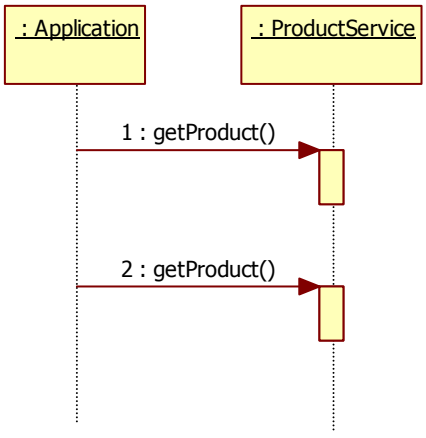
Import the given GreetingApplication (in the directory given applications) into Eclipse.
Then add all JAR files in the directory **libraries** to the classpath of this application.
Then you should be able to run the file HelloApp.java and it should write "Hello World" to the console.

Study the application, and notice that the String "Hello World" is injected.
Now modify the application such that the String "Hello CS525" gets injected and printed to the console.

Now import the given ProductApplication and add again all JAR files in the directory **libraries** to the classpath of this 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 **Application.java** by right-clicking on it and selecting **Run As → Java Application** to provide the following output:

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

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

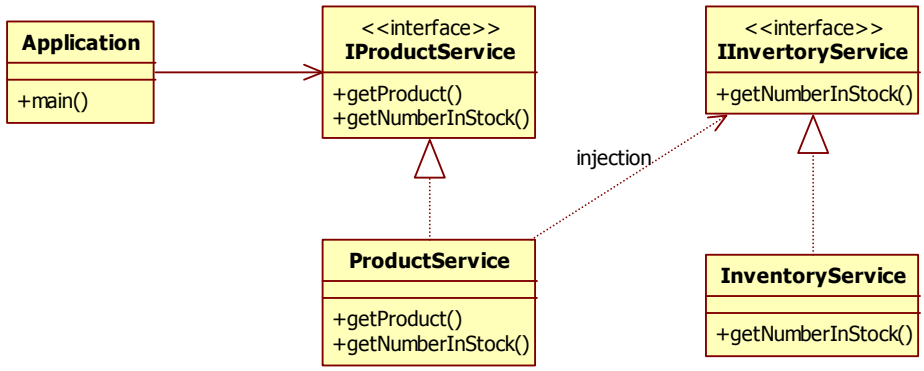
```
IProductService productService = new ProductService();
```

to the following lines of code:

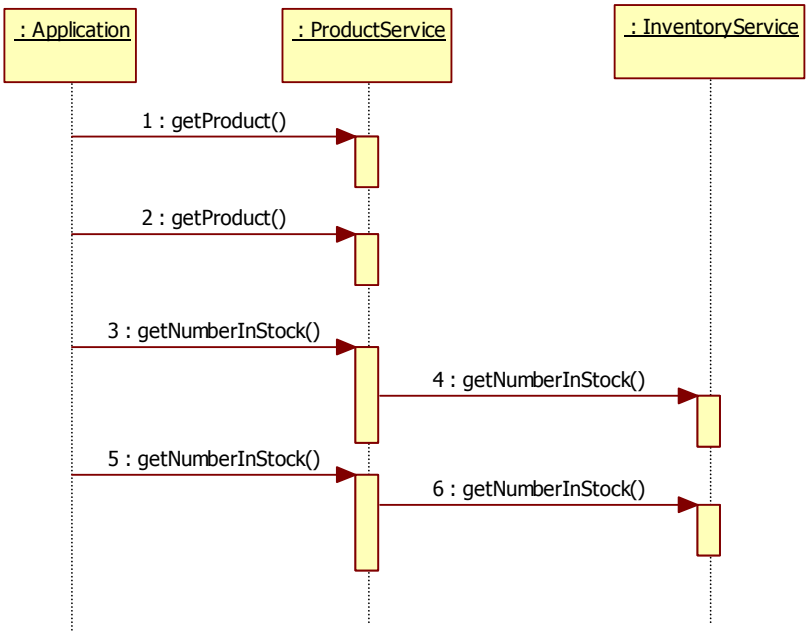
```
ApplicationContext context = new
    ClassPathXmlApplicationContext("springconfig.xml");
IProductService productService =
    context.getBean("productService", IProductService.class);
```

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.

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
```

The last part of this lab is using DI in our Bank application.

Start with the original bank application project that we used for the pattern labs (command pattern lab, strategy pattern lab, etc).

Modify the application such that it becomes a Spring application. Use dependency Injection between the AccountService and the AccountDAO.

Then write a new TestAccountDAO class that actually does the same as the original AccountDAO, only every method in this class also writes to the console the name of the method.

Now change the application as such that it uses the TestAccountDAO instead of the AccountDAO, without changing code.