More container concepts

Objectives

In this workshop we are going to look at how we can inject the whole object.

Then we will look at the constructor injection.

To show this, we create another interface and its implementation.

The implementation needs two other classes to call their methods in one of its methods. We will do it through dependency injection and the container through introducing the as the properties for this class in the application.xml file.

Then we will use the alternative way which is passing the two properties as the parameters of the construction of the class and then configure them in the application.xml file.

The AccountService interface and its implementation

We continue working with the BookStore project.
In our BookService system, we have a *BookService* interface.

Now we want another interface which is called *AccountsService* which will manage all the financial accounts of the system.
This interface will have a method *raiseInvoice* which will raise an invoice for the customer.
You can download the *AccountsService* and *AccountsServiceMockImpl*.

The scenario

- We want to have a method called *buyBook* which:
    - will call the *bookService* to find the book that the customer wants,
    - and then it will call the *accountService* to create an invoice for the customer.

- The client code should be able to call this integrated *buyBook* method.
- To do this, we create a third interface: *PurchasingService*
    - This interface has a method called *buyBook* which the client will call it. It is much easier for the client to call only one method.

Create PurchasingService interface

```
package se.yrgo.spring.services;

public interface PurchasingService {
     public void buyBook(String isbn);
}
```

The *PurchasingService* will do all the work by

- first calling BookService to find the book
- and then calling the *AccountService* to create an invoice for the customer.

The implementation of purchasingService

We create a class which is called *PurchasingServiceImpl* and this class implements the *PurchasingService* interface.

First of all, we follow the principle of *programming to interface*. So in the *buyBook* method we will not hardcode the *AccountsService* and *BooksService* implementations directly, because if we want to change the implementation later, we have to change them here and recompile the code.

Right now we want to use the mock implementations for these interfaces. But we need the flexibility to be able to switch to the production implementation which has a connection to a real database later.

So this will be the code in our *PurchasingServiceImp*l class.

```
private AccountsService accounts;
private BookService books;
```

And then we can have the set methods for these two properties:

```
public void setAccountsService (AccountsService accounts) {
      this.accounts = accounts;
}

public void setBookService (BookService books) {
      this.books= books;
}
```

If you look at the above code, you see that we do not have any code indicating the implementation of the interfaces. This is something that we will do in our xml file.

This is the PurchasingServiceImpl class:

```
package se.yrgo.spring.services;

import se.yrgo.spring.domain.Book;

public class PurchasingServiceImpl implements PurchasingService {
	private AccountsService accounts;
	private BookService books;

	@Override
	public void buyBook(String isbn) {
		//To find the book
		Book book = books.getBookByIsbn(isbn);
		//Raise invoice
		accounts.raiseInvoice(book);

	}

	public void setAccountsService (AccountsService accounts) {
		this.accounts = accounts;
	}

	public void setBookService (BookService books) {
		this.books= books;
	}
}
```

In Client class

This is the way that we use it in our client code:

```
package se.yrgo.spring.client;

import java.util.List;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import se.yrgo.spring.domain.Book;
```

```
import se.yrgo.spring.services.BookService;
import se.yrgo.spring.services.PurchasingService;

public class Client {
      public static void main(String[] args) {
            System.out.println("Testing buying a book ....");
            String isbn = "ISBN1"; // we know that this isbn exists in the mock

            ClassPathXmlApplicationContext container = new
ClassPathXmlApplicationContext("application.xml");

            PurchasingService purchasing =
container.getBean(PurchasingService.class);

            purchasing.buyBook(isbn);
            container.close();
      }
}
```

In application.xml

Now we should set up our wiring in the application.xml file.

```
<bean id="accountsService"
class="se.yrgo.spring.services.AccountsServiceMockImpl"/>

<bean id="bookService"
class="se.yrgo.spring.services.BookServiceMockImpl"/>

<bean id="purchasingService" class=
"se.yrgo.spring.services.PurchasingServiceImpl"/>
```

We run the code, but we get a null pointer exception which is when we call *getBookByIsbn* on the book object and the reference is null.

This is because we have not told the xml file to inject the *bookservice* and *accountsservice* into the purchasing service as a dependency.

So we modify the xml file as following:

```
<bean id="purchasingService" class=

"se.yrgo.spring.services.PurchasingServiceImpl">
      <property name = "bookService" ref="bookService" />
      <property name= "accountsService" ref="accountsService" />
</bean>
```

The name of properties should be the type of the properties in the class.

One property in the class *PurchasingServiceImpl* is BookService so the name of the property becomes *bookService* and the other property is AccountService so the name of the property becomes accountService.

This is the complete xml file:

```
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
            "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
  <bean id="bookService" class="se.yrgo.spring.services.BookServiceMockImpl">
  </bean>

   <bean id="accountsService"
            class="se.yrgo.spring.services.AccountsServiceMockImpl"/>

  <bean id="purchasingService" class= "se.yrgo.spring.services.PurchasingServiceImpl">
             <property name = "bookService" ref="bookService" />
             <property name= "accountsService" ref="accountsService" />
   </bean>
</beans>
```

Now if we run the code, it will work. We will get the message that “Raised the invoice for …”

Constructor Injection

An alternative to use the set methods for the dependencies, is to pass the dependent objects into the constructor.

We remove the set methods in this class and we add a constructor with these two properties as arguments to this constructor:

In *PurchasingServiceImpl*

```
public PurchasingServiceImpl(AccountsService accounts, BookService books) {
            this.accounts = accounts;
```

```
            this.books = books;
}
```

And in the xml file, we change as follows:

```
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
            "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
  <bean id="bookService"
class="se.yrgo.spring.services.BookServiceMockImpl">
  </bean>

  <bean id="accountsService" class="se.yrgo.spring.services.AccountsServiceMockImpl"/>

  <bean id="purchasingService" class= "se.yrgo.spring.services.PurchasingServiceImpl">
                  <constructor-arg ref = "bookService"/>
                <constructor-arg ref = "accountsService" />
  </bean>
</beans>
```

Run the code again. It should work.