Aspect Oriented Programming (AOP)

Exercise AOP.1 – Basic Spring AOP

The Setup:

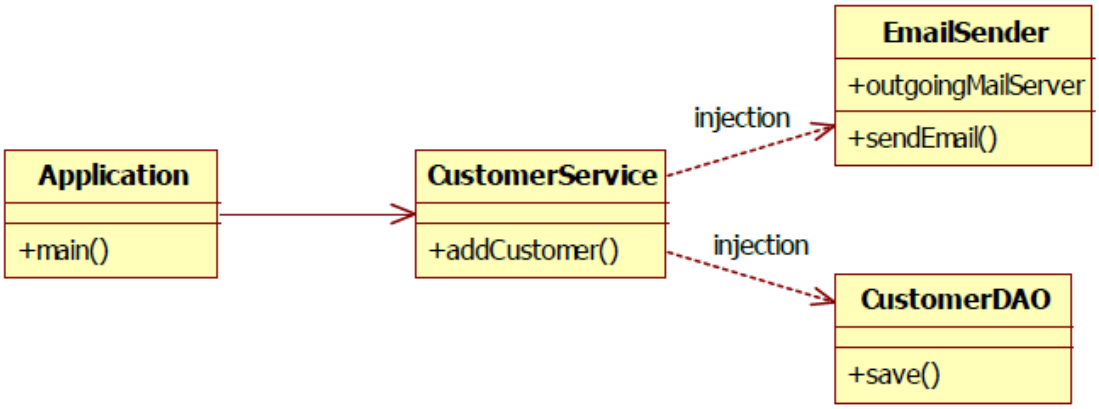
This exercise is a basic exercise to start using the Aspect Oriented Programming techniques available through the Spring Framework.

Start by downloading **Lab10-AOP-1** from Sakai and add the **Spring dependencies** to it. Then also add the following AspectJ dependencies:

```
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjrt</artifactId>
    <version>1.9.2</version>
</dependency>
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.9.2</version>
</dependency>
```

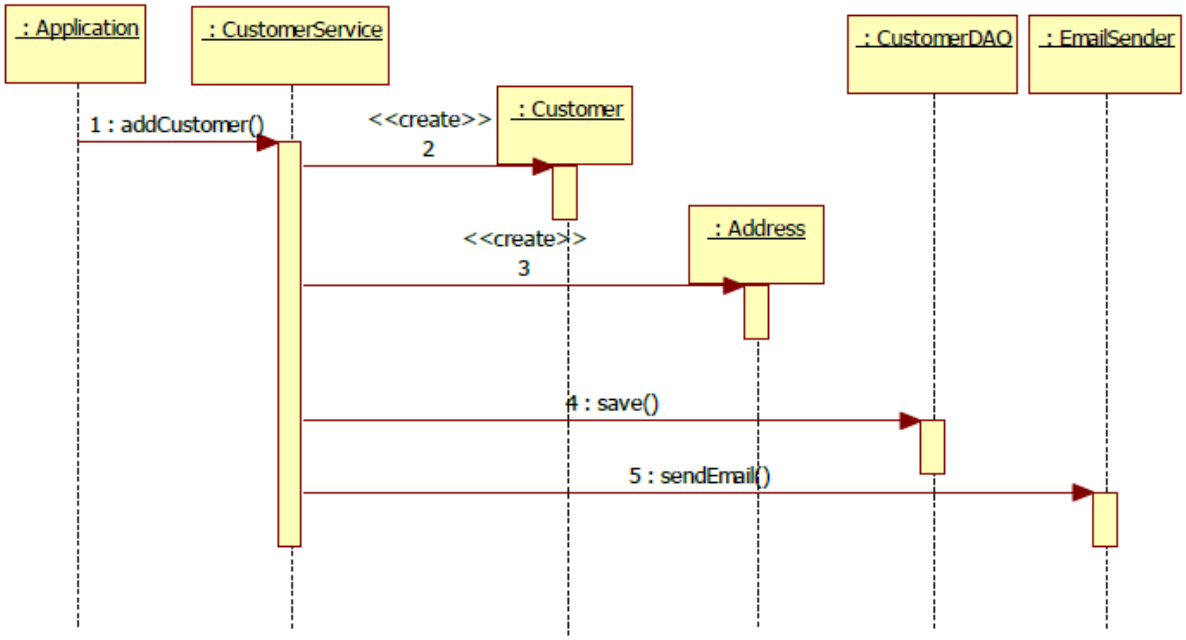
Be aware that if you use XML configuration your **springconfig.xml** file will require the **aop namespace** for this exercise.

The Application:



The provided application has a CustomerService class with an injected reference to the EMailSender class and an injected reference to the CustomerDAO class.

When addCustomer() is called on the CustomerService class it creates a Customer object and a corresponding Address object. The Customer is then saved to the database by the CustomerService by calling the save() method on the CustomerDAO, and an email is sent to the customer by calling the sendEmail() method on the EmailSender (see the sequence diagram on the next page)



Running the application should give the following output:

```
CustomerDAO: saving customer Frank Brown
EmailSender: sending 'Welcome Frank Brown as a new customer' to
fbrown@acme.com
```

The Exercise:

a) Reconfigure the application so that whenever the sendMail method on the EmailSender is called, a log message is created (using an after advice AOP annotation).

Remember to configure Spring to look for your aspect annotations!

Running your updated code should produce the following output:

```
CustomerDAO: saving customer Frank Brown
EmailSender: sending 'Welcome Frank Brown as a new customer' to
fbrown@acme.com
Fri Jun 05 14:09:47 GMT 2009 method= sendMail
```

b) Now change the log advice in such a way that the email address and the message are logged as well. You should be able to retrieve the email address and the message through the arguments of the **sendEmail()** method. This should produce the following output:

```
CustomerDAO: saving customer Frank Brown
EmailSender: sending 'Welcome Frank Brown as a new customer' to
fbrown@acme.com
Fri Jun 05 14:17:31 GMT 2009 method= sendEmail address=fbrown@acme.com
message= Welcome Frank Brown as a new customer
```

Please see the next page for part c) and d)

c) Change the log advice again, this time so that the outgoing mail server is logged as well. The **outgoingMailServer** is an attribute of the **EmailSender** object, which you can retrieve through the **joinpoint.getTarget()** method. This should produce the following output:

```
CustomerDAO: saving customer Frank Brown
EmailSender: sending 'Welcome Frank Brown as a new customer' to
fbrown@acme.com
Fri Jun 05 14:22:24 GMT 2009 method= sendEmail address=fbrown@acme.com
message= Welcome Frank Brown as a new customer
outgoing mail server = smtp.acme.com
```

d) Write a new advice that calculates the duration of the method calls to the DAO object and outputs the result to the console. Spring provides a stopwatch utility that can be used for this by using the following code:

```
import org.springframework.util.StopWatch;

public Object invoke(ProceedingJoinPoint call ) throws Throwable {
   StopWatch sw = new StopWatch();
   sw.start(call.getSignature().getName());
   Object retVal = call.proceed();
   sw.stop();

   long totaltime = sw.getLastTaskTimeMillis();
   // print the time to the console

   return retVal;
}
```

This should produce the following output:

```
CustomerDAO: saving customer Frank Brown
Time to execute save = 350 ms
EmailSender: sending 'Welcome Frank Brown as a new customer' to
fbrown@acme.com
Fri Jun 05 14:30:07 GMT 2009 method= sendEmail address=fbrown@acme.com
message= Welcome Frank Brown as a new customer
outgoing mail server = smtp.acme.com
```

Once you're done with the exercise delete the target directory and create a zip of your project. If you want you can already submit it on Sakai (you have unlimited resubmits).

Do this by writing the first part of your report in the textbox and attaching your zip.

Spring Applications

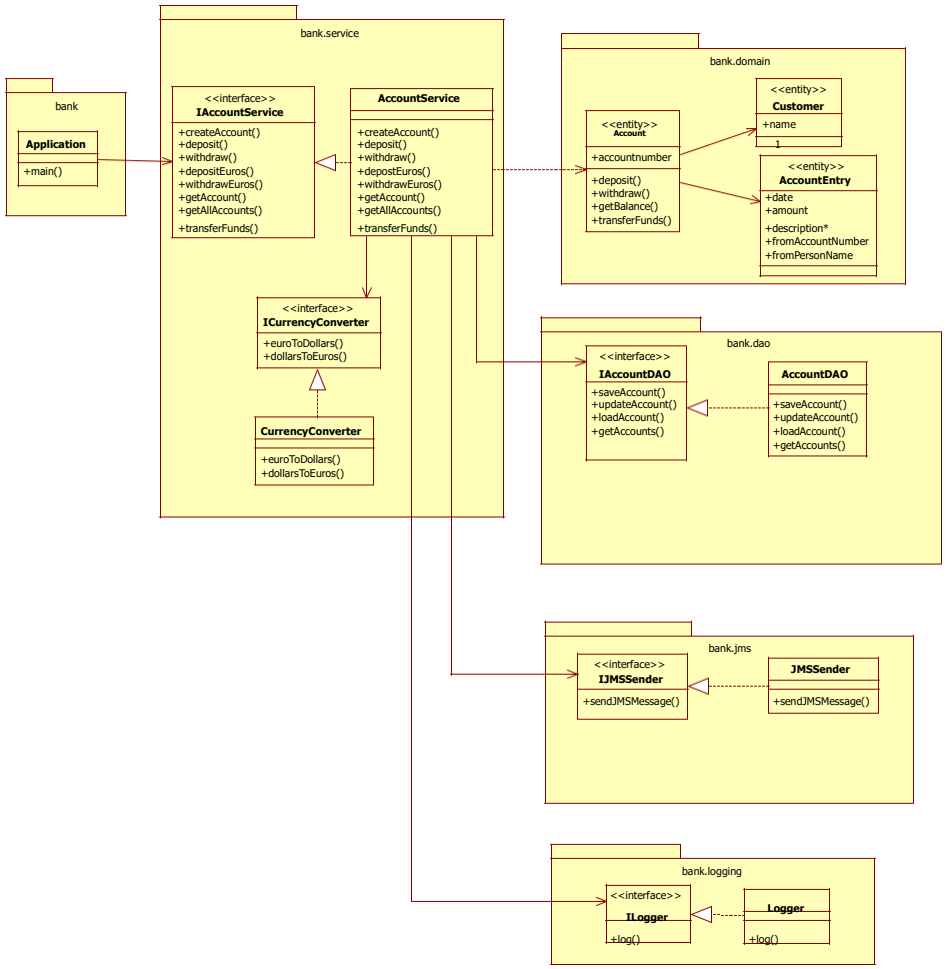
Exercise SSL.1 – Bank Application Dependency Injection

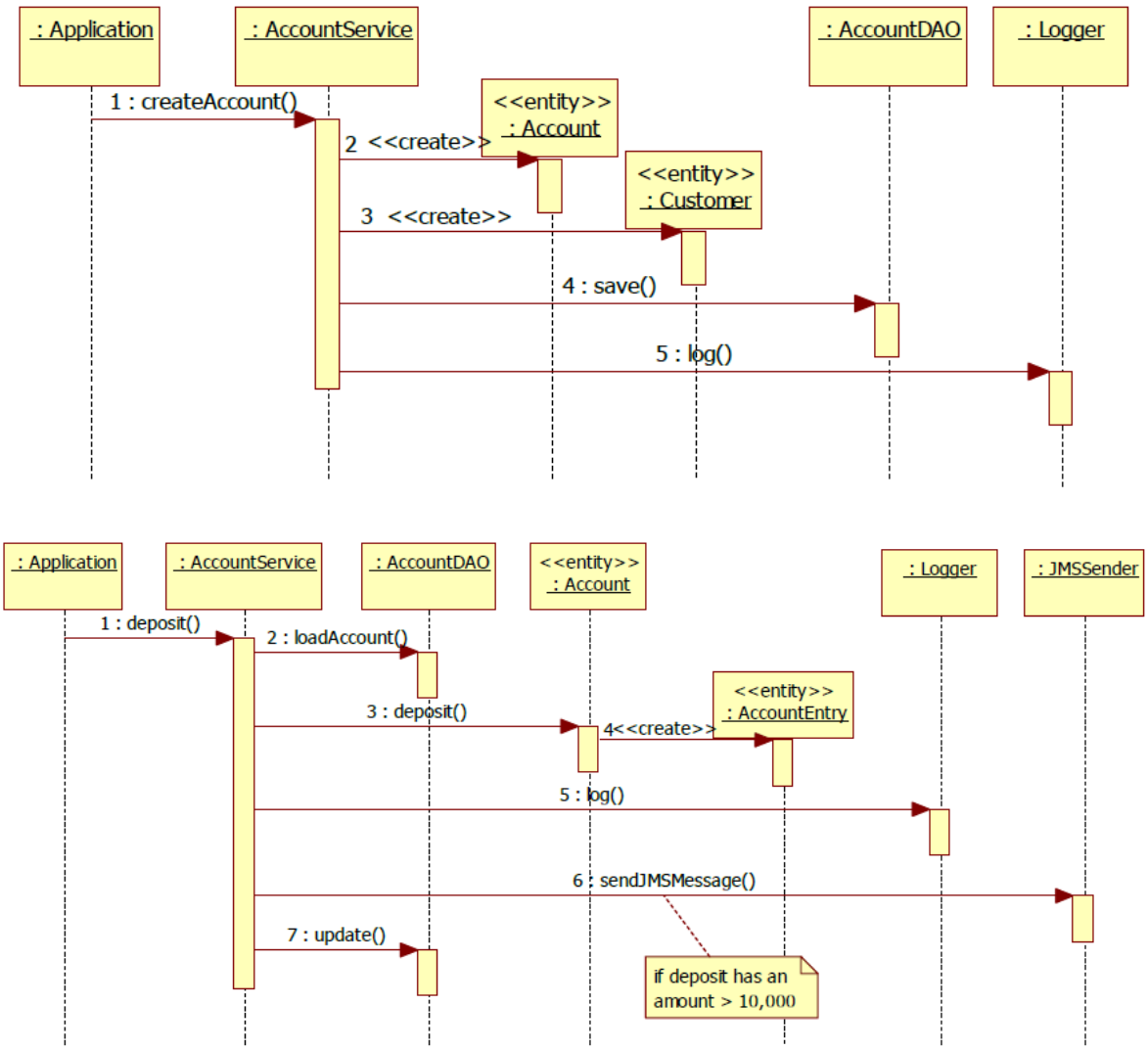
The Setup:

This exercise introduces the bank application. The bank application is a small application that embodies most of the architectural needs of a more real world enterprise application. Although the application that we start with in this exercise does not use any of Spring's features (yet), many areas in this application could benefit from them.

In this exercise, we will start by adding dependency injection to the application. In subsequent exercises we will continue to build on this, adding new features as they are covered.
Start by downloading **Lab10-Bank_Application** from Sakai and add the **Spring dependencies**.

The Application:





The bank application uses an **AccountService** object to perform the various bank-related services such as creating accounts, depositing money (Euros or Dollars), withdrawing money (Euros or Dollars), and transferring funds between accounts.

The AccountService object manipulates the domain objects Account, Customer and AccountEntry through the methods mentioned above. An Account object and a Customer object are created when CreateAccount() is called, and an AccountEntry is created for every deposit or withdrawal. All changes are then saved to the database through the AccountDAO.

Since the Accounts only work internally with dollar amounts, all euro deposits and withdrawals are first converted to dollars using a CurrencyConverter object.

All AccountService methods are logged through the Logger object, and whenever an amount greater than 10,000 dollars is deposited or transferred into an account, an additional JMS message is sent to the taxation services department.
Running **App.java** in the **edu.mum.cs544.bank** package should output:

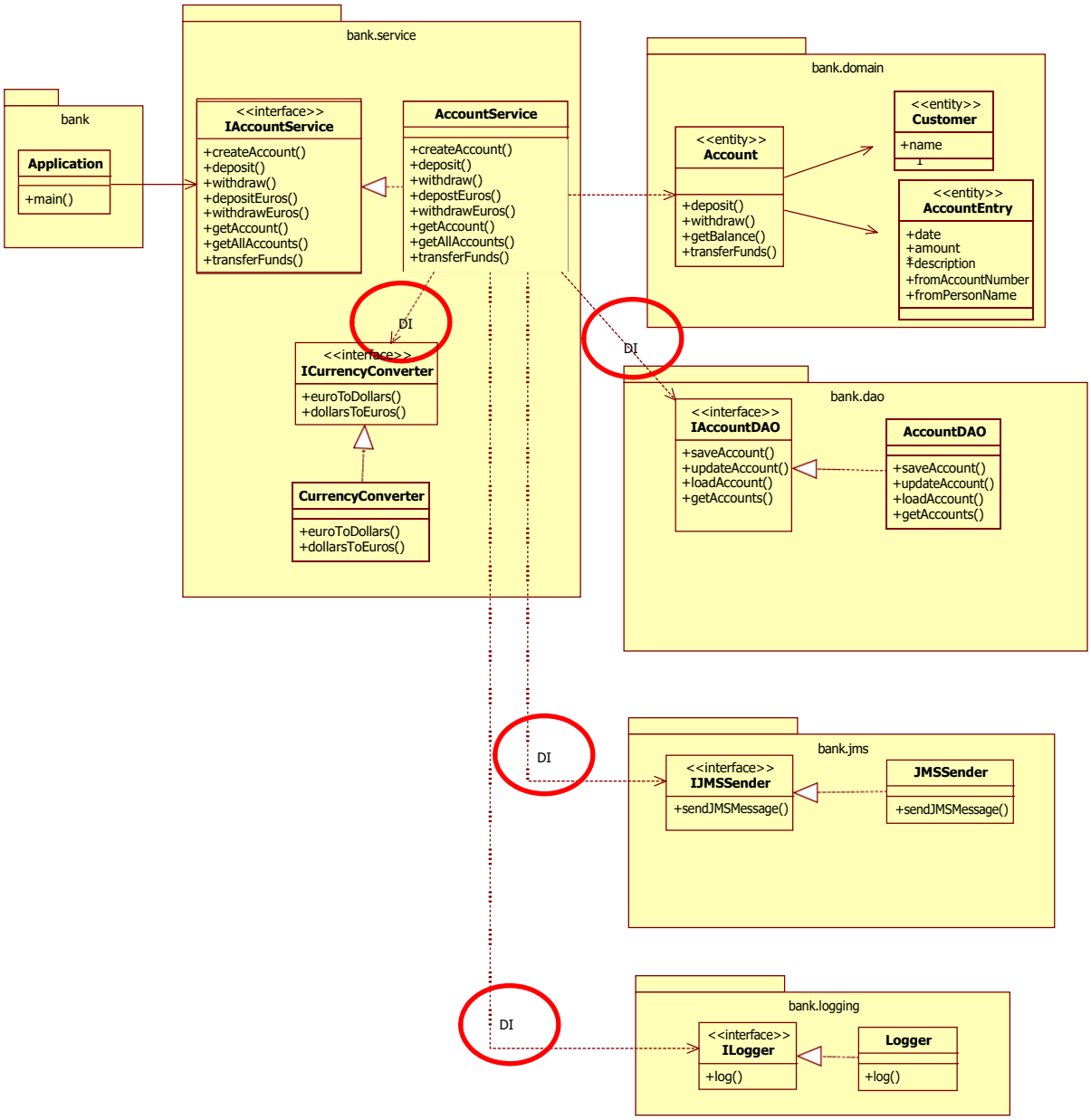
```
Jun 12, 2009 11:45:24 PM bank.logging.Logger log
INFO: createAccount with parameters accountNumber= 1263862 , customerName= Frank Brown
Jun 12, 2009 11:45:24 PM bank.logging.Logger log
INFO: createAccount with parameters accountNumber= 4253892 , customerName= John Doe
Jun 12, 2009 11:45:24 PM bank.logging.Logger log
INFO: deposit with parameters accountNumber= 1263862 , amount= 240.0
Jun 12, 2009 11:45:24 PM bank.logging.Logger log
INFO: deposit with parameters accountNumber= 1263862 , amount= 529.0
CurrencyConverter: converting 230.0 dollars to euros
Jun 12, 2009 11:45:24 PM bank.logging.Logger log
INFO: withdrawEuros with parameters accountNumber= 1263862 , amount= 230.0
Jun 12, 2009 11:45:24 PM bank.logging.Logger log
INFO: deposit with parameters accountNumber= 4253892 , amount= 12450.0
JMSSender: sending JMS message =Deposit of $ 12450.0 to account with accountNumber=
4253892
CurrencyConverter: converting 200.0 dollars to euros
Jun 12, 2009 11:45:24 PM bank.logging.Logger log
INFO: depositEuros with parameters accountNumber= 4253892 , amount= 200.0
Jun 12, 2009 11:45:24 PM bank.logging.Logger log
INFO: transferFunds with parameters fromAccountNumber= 4253892 , toAccountNumber=
1263862 , amount= 100.0 , description= payment of invoice 10232
Statement For Account: 4253892
Account Holder: John Doe
-Date--------------------------Description-------------------Amount-------------
  Fri Jun 12 23:45:24 GMT 2009                       deposit            12450.00
  Fri Jun 12 23:45:24 GMT 2009                       deposit              314.00
  Fri Jun 12 23:45:24 GMT 2009      payment of invoice 10232              -100.00
--------------------------------------------------------------------------------
                                        Current Balance:            12664.00

Statement For Account: 1263862
Account Holder: Frank Brown
-Date--------------------------Description-------------------Amount-------------
  Fri Jun 12 23:45:24 GMT 2009                       deposit              240.00
  Fri Jun 12 23:45:24 GMT 2009                       deposit              529.00
  Fri Jun 12 23:45:24 GMT 2009                      withdraw             -361.10
  Fri Jun 12 23:45:24 GMT 2009      payment of invoice 10232              100.00
--------------------------------------------------------------------------------
                                        Current Balance:              507.90
```

The Exercise:

Change the bank application in such a way that the Logger, CurrencyConverter, AccountDAO and JMSSender are injected into the AccountService, rather than being instantiated with ***new***. In other word, AccountService should no longer contain these lines:

```
accountDAO = new AccountDAO();
currencyConverter = new CurrencyConverter();
jmsSender =  new JMSSender();
logger = new Logger();
```

Also update **App.java** so that it retrieves the AccountService from the Spring context.



Once you are done with the exercise, delete the target directory and create a zip file. Submit this zip file on Sakai along with your report. The report should be written in the textbox on Sakai, not attached as a file.

If you want you can even already submit, as you have unlimited resubmits.

Exercise AOP.2 – Bank Application AOP

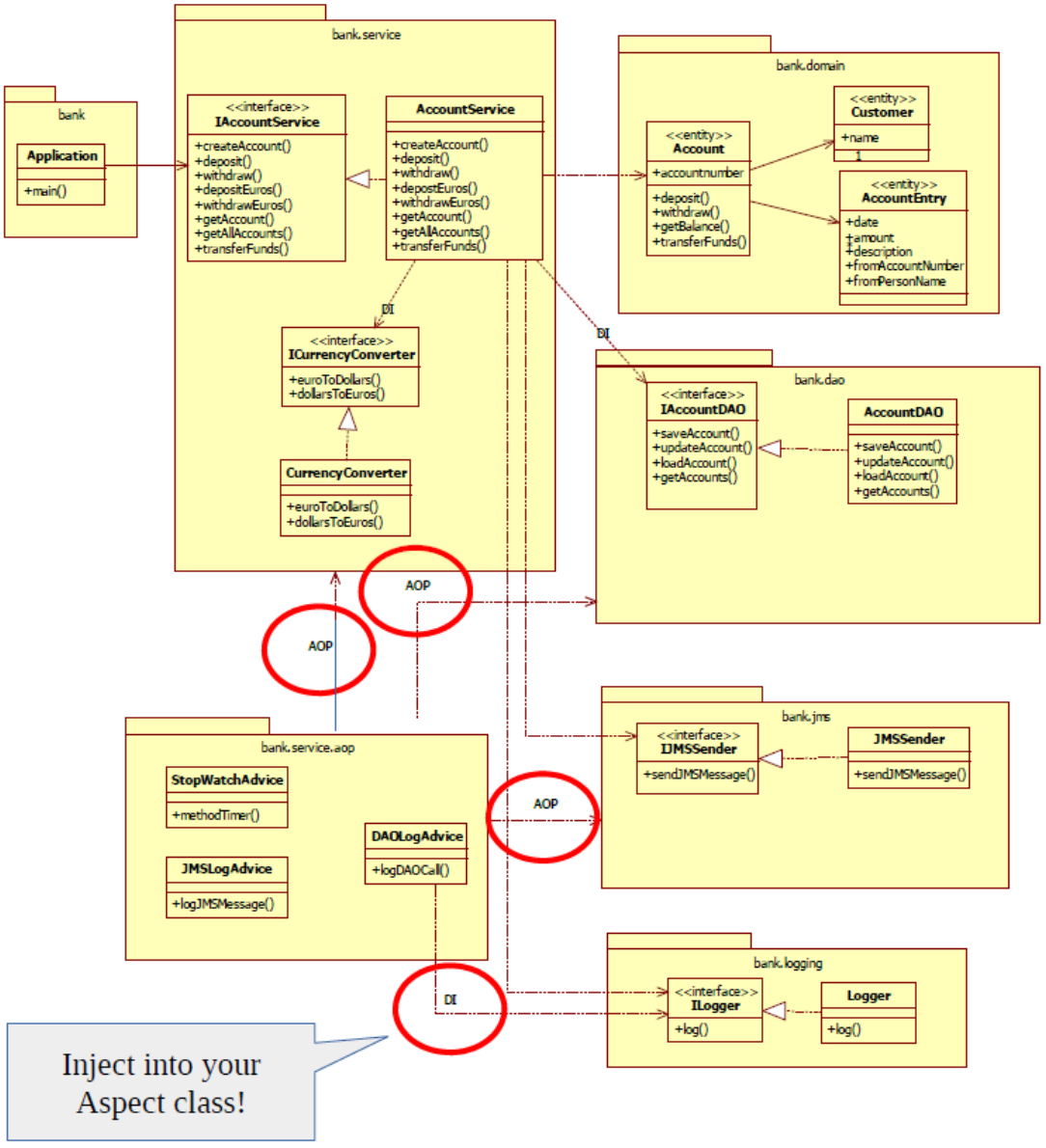
The Setup:

In this exercise, we will be extending the bank application to use AOP. Create a copy of **Lab10-Bank_Application** and call it **Lab10-AOP-2**. Update the pom.xml to have the new name as well, and also add the AOP dependencies (as shown in the first AOP exercise).

The Exercise:

Use AOP to:

- Log every call to any method in the bank.dao package (using the Logger).
- Use the Spring StopWatch functionality to measure the duration of all service level methods (any method in the bank.service package) and output the results to the console.
- Log every JMS message that is sent (using the Logger)
- In AccountService you can remove all the calls to the logger so that it is easier to see whether your advice is running or not.
- Be sure to inject the logger into the advice class as shown below.



To submit delete the target directory and create a zip of your project. Then attach it on Sakai along with your report (in the textbox on Sakai).

Please let me know how things went, even if you were not able to finish any of the exercises.