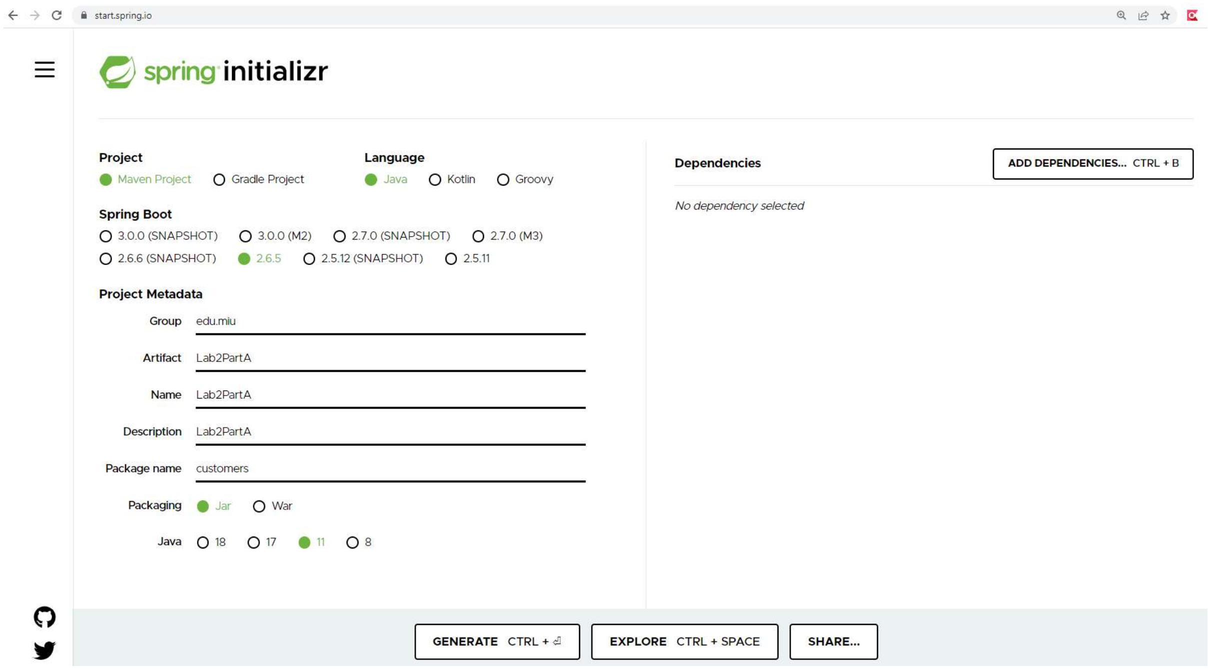
Lab 2

Part A: Spring Boot

In the browser go to https://start.spring.io/



Project: [✓ Maven Project] Gradle Project

Language: [✓ Java] Kotlin Groovy

Spring Boot: 3.0.0 (SNAPSHOT) 3.0.0 (M2) 2.7.0 (SNAPSHOT) 2.7.0 (M3) 2.6.6 (SNAPSHOT) [✓ 2.6.5] 2.5.12 (SNAPSHOT) 2.5.11

Project Metadata

Group	edu.miu
Artifact	Lab2PartA
Name	Lab2PartA
Description	Lab2PartA
Package name	customers
Packaging	[✓ Jar] War
Java	18 17 [✓ 11] 8

Dependencies

No dependency selected

Fill in the fields as shown above (select the latest stable Spring Boot version).

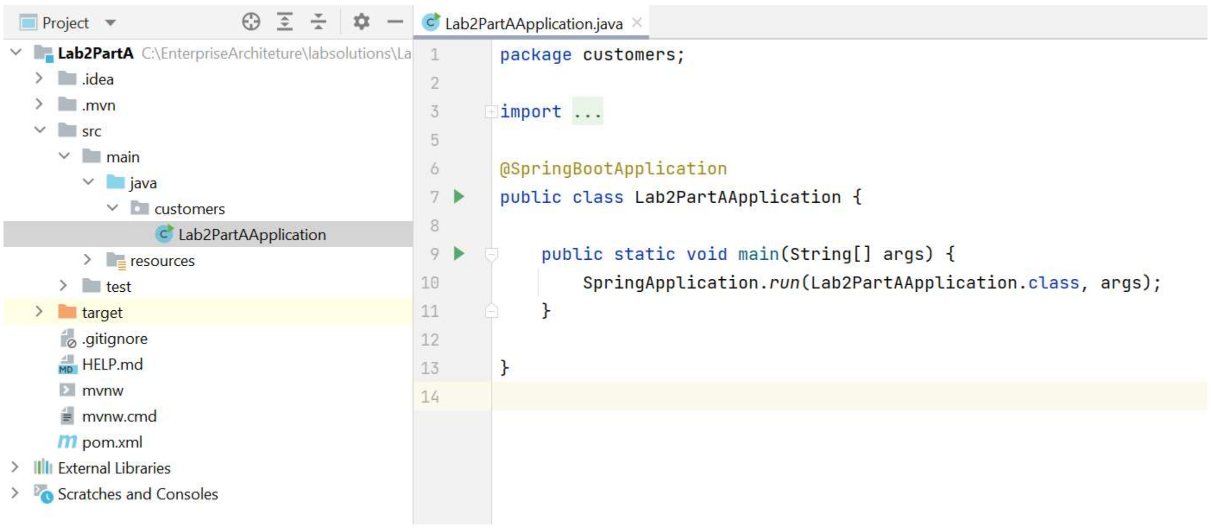
Click the **Generate** button.

Lab2PartA.zip is now downloaded to your computer.

Unzip the content of this file

In IntelliJ, open the just created project

You see now a basic Spring boot application.



```java
package customers;

import ...

@SpringBootApplication
public class Lab2PartAApplication {

    public static void main(String[] args) {
        SpringApplication.run(Lab2PartAApplication.class, args);
    }

}
```

a. Copy and paste the Java files from the Customer application of lab 1 into this project, and make it work using Spring Boot.
b. Inject the outgoingMailServer attribute in the EmailSender using the **application.properties** file in src/main/resources
c. Write a CustomerDAOMock class that we want to use in our test environment. In our production environment we want to use the existing CustomerDAO class. Use **profiles** so that we can configure in **application.properties** which CustomerDAO we want to inject.

Part B: AOP

Copy and paste the **Lab2PartA** to **Lab2PartB**.

If we run the code of **Lab2PartA**, we get the following output in the console:

```
CustomerDAO: saving customer Frank Brown
Logging 2018-03-20T11:23:44.588 Customer is saved in the DB: Frank
Brown
EmailSender: sending 'Welcome Frank Brown as a new customer' to
fbrown@acme.com
Logging 2018-03-20T11:23:44.588 Email is sent: message= Welcome
Frank Brown as a new customer , emailaddress =fbrown@acme.com
```

a. Modify the application so that whenever the sendEmail method on the EmailSender is called, a log message is created (using an after advice AOP annotation). This should produce the following output:

```
CustomerDAO: saving customer Frank Brown
Logging 2018-03-20T11:23:44.588 Customer is saved in the DB: Frank
Brown
EmailSender: sending 'Welcome Frank Brown as a new customer' to
fbrown@acme.com
Logging 2018-03-20T11:23:44.588 Email is sent: message= Welcome
Frank Brown as a new customer , emailaddress =fbrown@acme.com
2018-03-20T11:23:44.789 method=sentEmail
```

In order to use AOP in a Spring Boot project, we have to add the following dependency in the POM file:

```
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.8.12</version>
    <scope>compile</scope>
</dependency>
```

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
Logging 2018-03-20T11:23:44.588 Customer is saved in the DB: Frank
Brown
EmailSender: sending 'Welcome Frank Brown as a new customer' to
fbrown@acme.com
Logging 2018-03-20T11:23:44.588 Email is sent: message= Welcome
Frank Brown as a new customer , emailaddress =fbrown@acme.com
2018-03-20T11:23:44.789 method=sentEmail address=fbrown@acme.com
message= Welcome Frank Brown as a new customer
```

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
Logging 2018-03-20T11:23:44.588 Customer is saved in the DB: Frank
Brown
EmailSender: sending 'Welcome Frank Brown as a new customer' to
fbrown@acme.com
Logging 2018-03-20T11:23:44.588 Email is sent: message= Welcome
Frank Brown as a new customer , emailaddress =fbrown@acme.com
2018-03-20T11:23:44.789 method=sentEmail address=fbrown@acme.com
message= Welcome Frank Brown as a new customer outgoing mail server
= smtp.mydomain.com
```

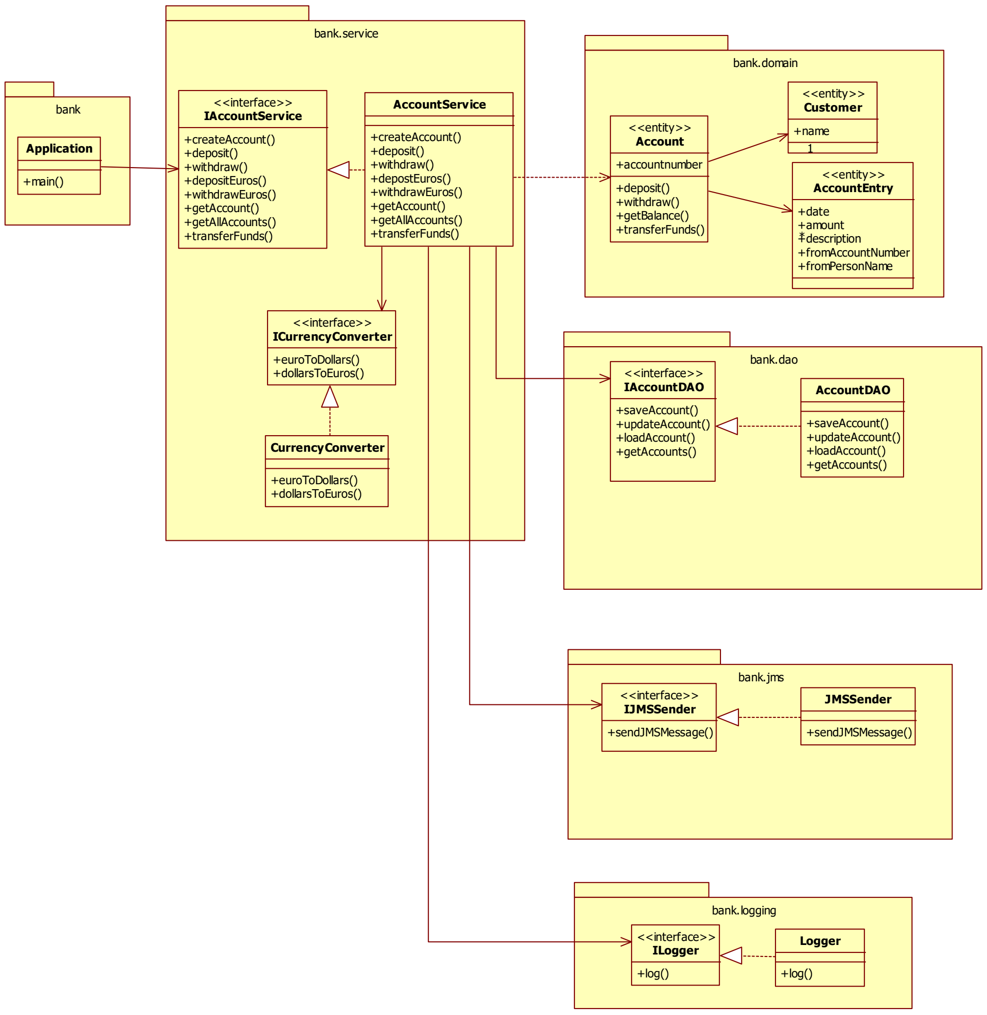
d. Write a new advice that calculates the duration of the method calls to the DAO object and outputs the result to the console. Spring provides a stopwatch utility that can be used for this by using the following code:

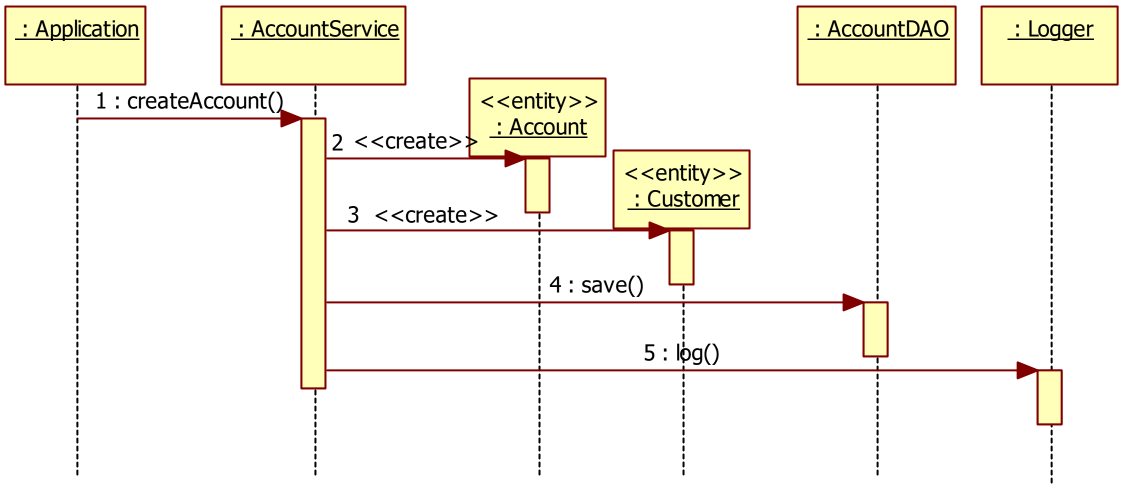
```java
import org.springframework.util.StopWatch;

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
```

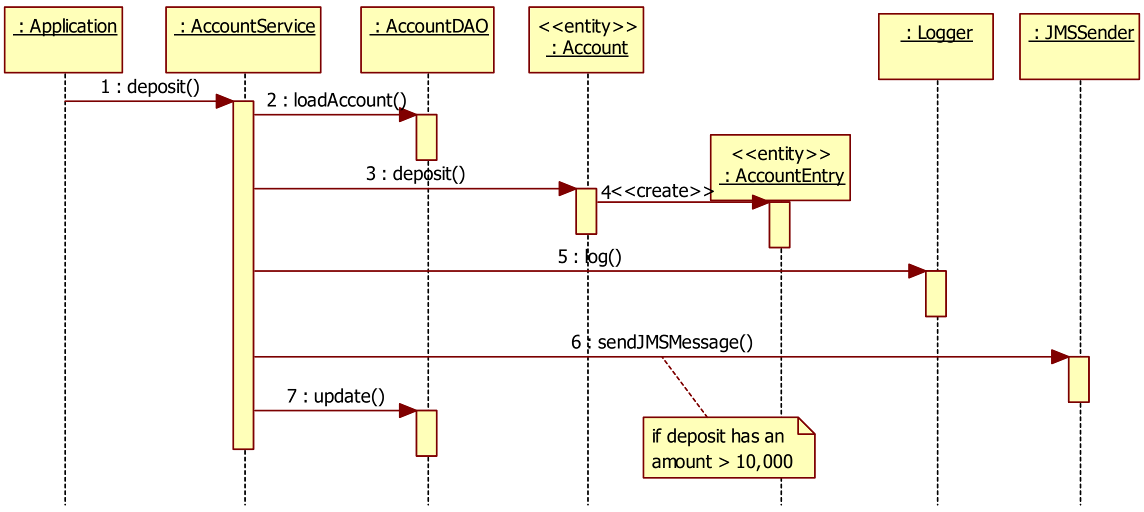
Part 3: Bank Application with DI

Open the given project **BankApplication.** This application contains the code of a simple bank account application. The given code is plain java code.

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 **Application.java** in the **bank** package should produce the following output:

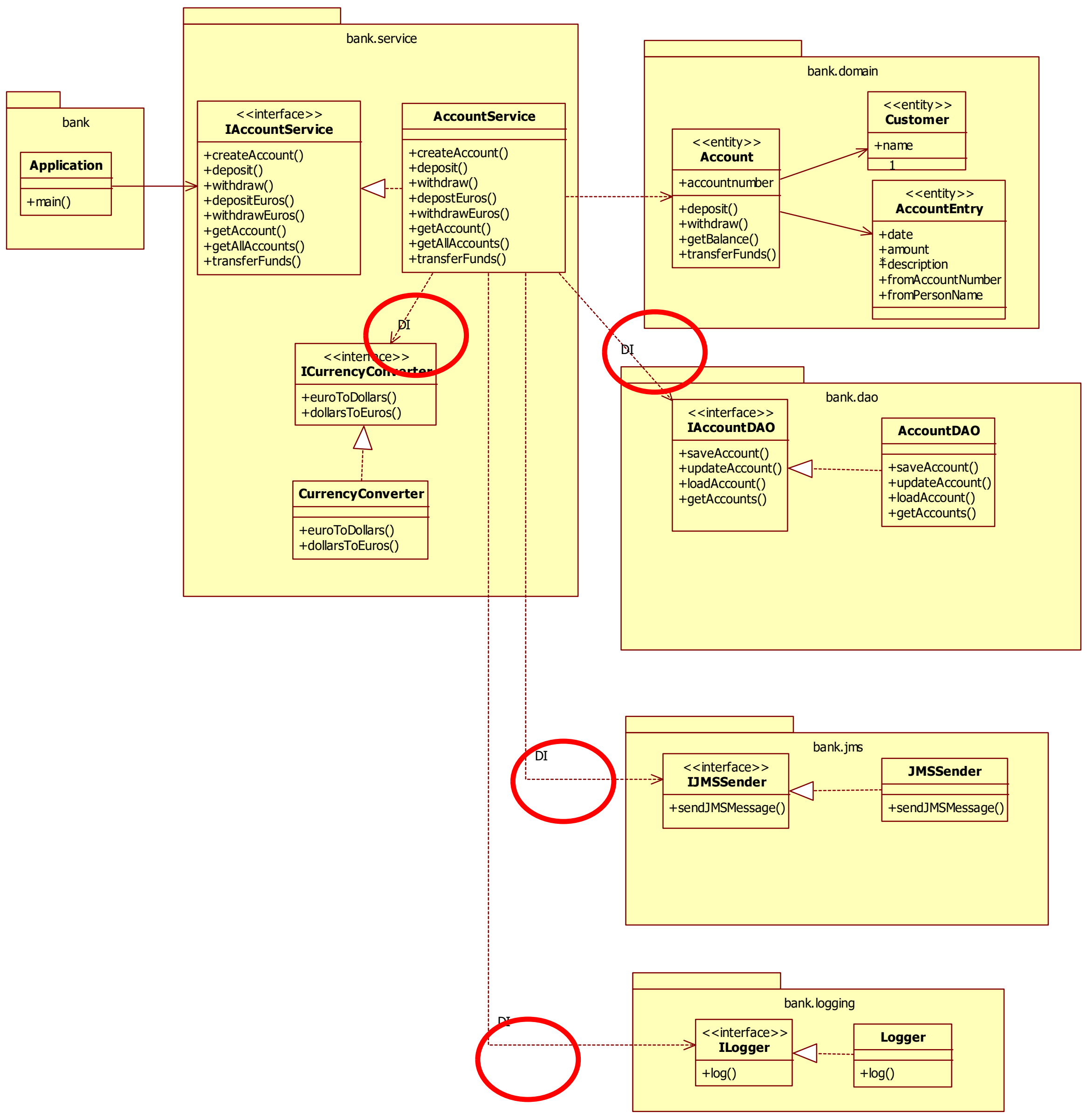
```
Jun 12, 2009 11:45:24 PM bank.logging.Logger log
INFO: createAccount with parameters accountNumber= 1263862 , customerName= Frank
Brown
Jun 12, 2009 11:45:24 PM bank.logging.Logger log
INFO: createAccount with parameters accountNumber= 4253892 , customerName= John Doe
Jun 12, 2009 11:45:24 PM bank.logging.Logger 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
```

a. 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***.

Therefore, AccountService should no longer contain lines that create these objects with new:

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

Instead, these objects should be injected.



Part 4: Bank Application with AOP

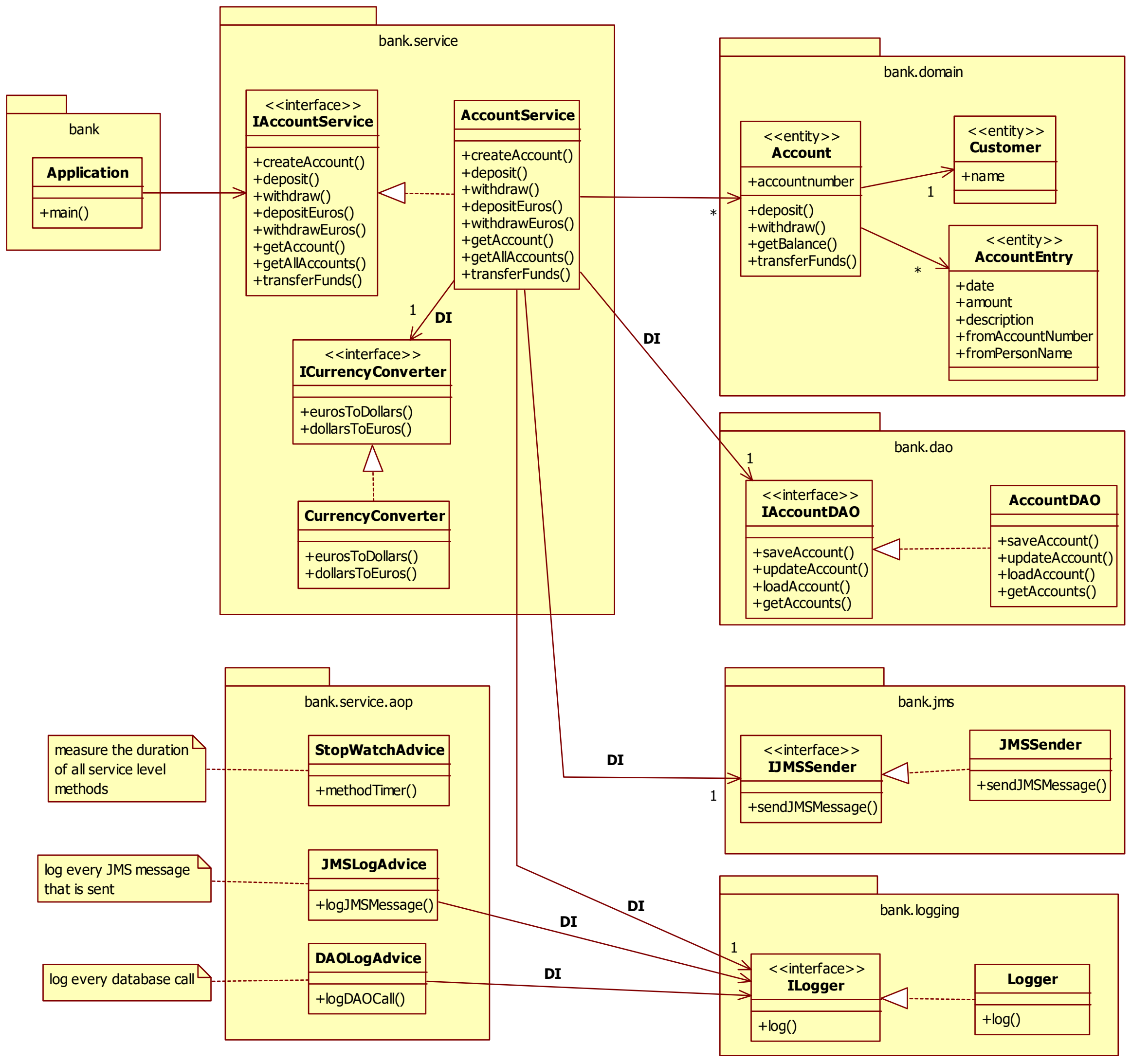
b. Now we want to extend the bank application to use AOP.

Use AOP to

a) Log every call to any method in the bank.dao package (using the Logger).

b) 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.

c) Log every JMS message that is sent (using the Logger)



What to hand in:

1. A separate zip file with the solution of part 1
2. A separate zip file with the solution of part 2
3. A separate zip file with the solution of part 3
4. A separate zip file with the solution of part 4