Contents

Source code:

Exercise AOP.1 - **Lab2-AOP-1**
Exercise SSL.1 – Bank Application Dependency Injection Exercise 1: **Lab2-Bank_Application**
Exercise SSL.1 – Bank Application Dependency Injection Exercise 2: **Lab2-AOP-2**

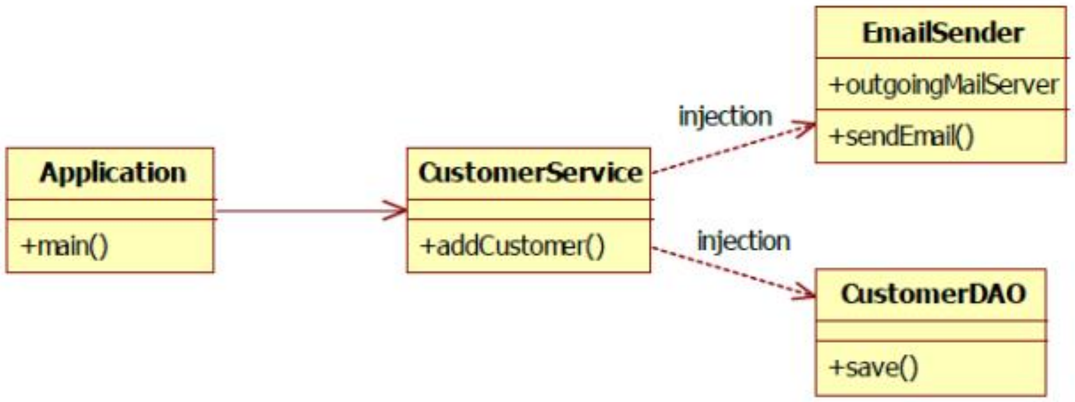
Exercise AOP.1 – Basic Spring AOP

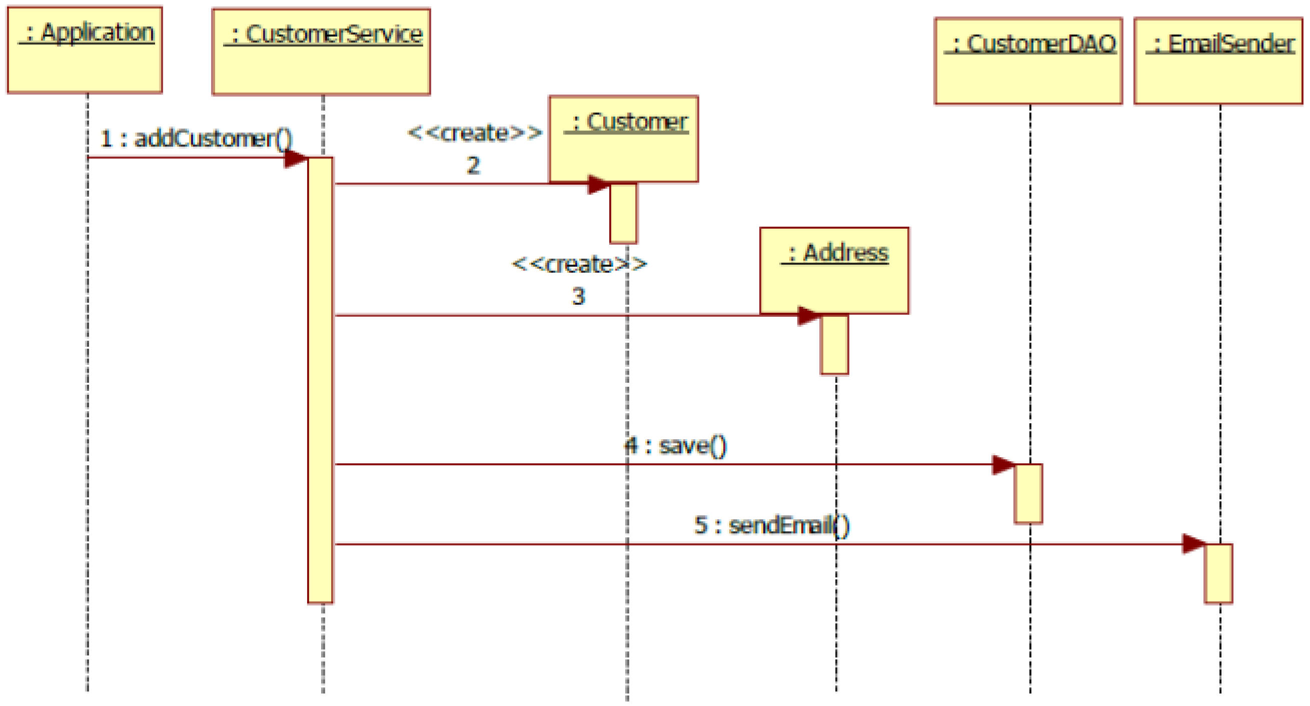
The Setup:

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:

- org.aspectj aspectjrt 1.9.2
- org.aspectj aspectjweaver 1.9.2

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





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 1

Source code: Lab2-AOP-1

DI

Setter Based DI

Define bean in springconfig.xml & define setter methods.

Springconfig.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"
      xmlns:aop="http://www.springframework.org/schema/aop"
      xmlns:context="http://www.springframework.org/schema/context"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
     http://www.springframework.org/schema/beans/spring-beans.xsd
     http://www.springframework.org/schema/aop
     http://www.springframework.org/schema/aop/spring-aop.xsd http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

   <bean id="customerService" class="edu.mum.cs544.CustomerService">
      <property name="customerDAO" ref="customerDAO" />
      <property name="emailSender" ref="emailSender" />
   </bean>
   <bean id="customerDAO"  class="edu.mum.cs544.CustomerDAO" />
   <bean id="emailSender"  class="edu.mum.cs544.EmailSender" />
</beans>
```

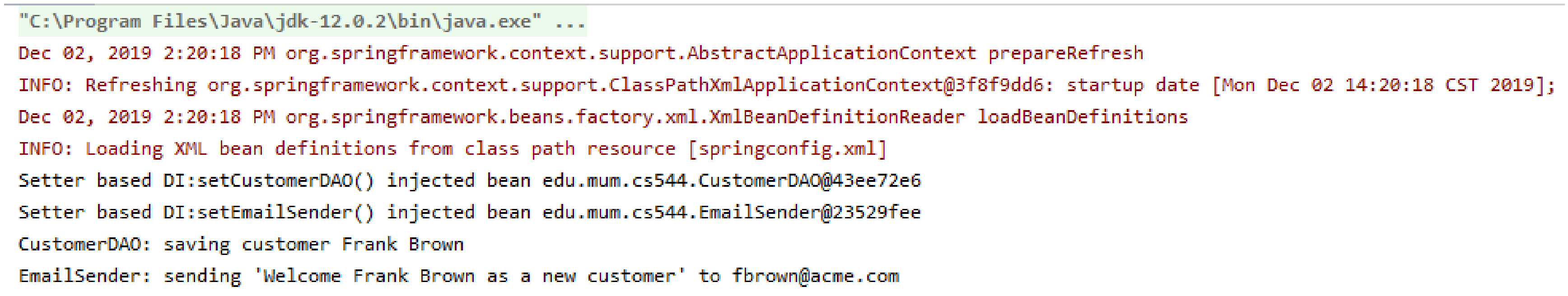
App.java

```
public class App
{
   public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("springconfig.xml");
      ICustomerService customerService = context.getBean("customerService", ICustomerService.class);

      customerService.addCustomer("Frank Brown", "fbrown@acme.com",
            "mainstreet 5", "Chicago", "60613");
   }
}
```

CustomerService.java

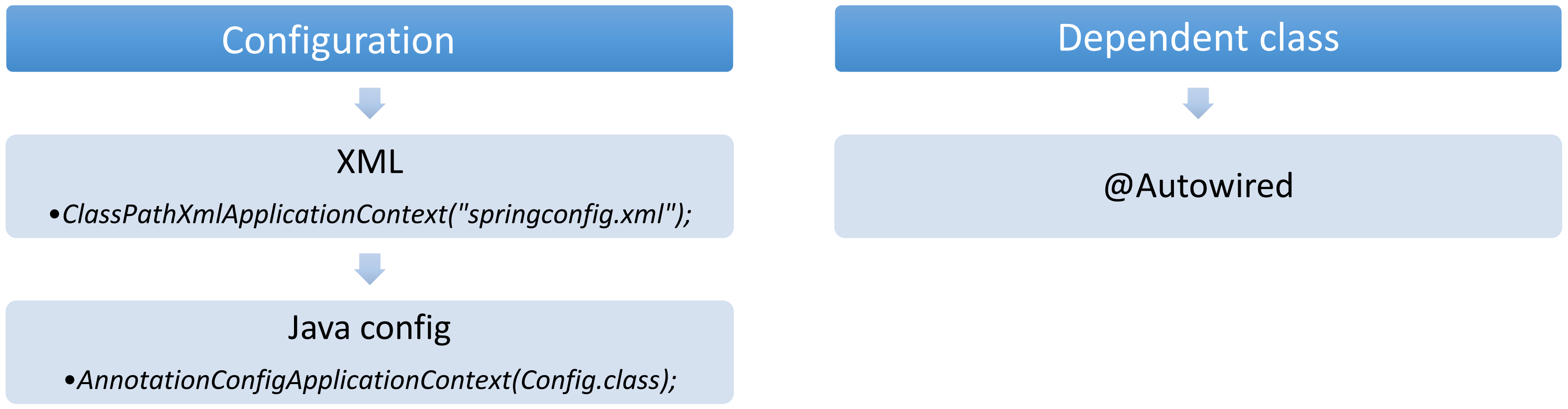
```
package edu.mum.cs544;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class CustomerService implements ICustomerService {
   /*
    * 1. Setter based DI START
    * */
   private ICustomerDAO customerDAO;
   private IEmailSender emailSender;

   public void setCustomerDAO(ICustomerDAO customerDAO) {
      System.out.println("Setter based DI:setCustomerDAO() injected bean " + customerDAO);
      this.customerDAO = customerDAO;
   }

   public void setEmailSender(IEmailSender emailSender) {
      System.out.println("Setter based DI:setEmailSender() injected bean " + emailSender);
      this.emailSender = emailSender;
   }
   /*
    * 1. Setter based DI END
    * */
}
```

Result



```
"C:\Program Files\Java\jdk-12.0.2\bin\java.exe" ...
Dec 02, 2019 2:20:18 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@3f8f9dd6: startup date [Mon Dec 02 14:20:18 CST 2019];
Dec 02, 2019 2:20:18 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [springconfig.xml]
Setter based DI:setCustomerDAO() injected bean edu.mum.cs544.CustomerDAO@43ee72e6
Setter based DI:setEmailSender() injected bean edu.mum.cs544.EmailSender@23529fee
CustomerDAO: saving customer Frank Brown
EmailSender: sending 'Welcome Frank Brown as a new customer' to fbrown@acme.com
```

Property Based DI


Configuration
XML
•ClassPathXmlApplicationContext("springconfig.xml");
Java config
•AnnotationConfigApplicationContext(Config.class);
Dependent class
@Autowired

Using Java Config

Config.java

```
@Configuration
@ComponentScan("edu.mum.cs544")
@EnableAspectJAutoProxy
public class Config {
```

}

App.java

```
      ApplicationContext context = new AnnotationConfigApplicationContext(Config.class);
```

Using xml config

Springconfig.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"
      xmlns:aop="http://www.springframework.org/schema/aop"
      xmlns:context="http://www.springframework.org/schema/context"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
     http://www.springframework.org/schema/beans/spring-beans.xsd
     http://www.springframework.org/schema/aop
     http://www.springframework.org/schema/aop/spring-aop.xsd http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

   <context:component-scan base-package="edu.mum.cs544"/>
</beans>
```

App.java

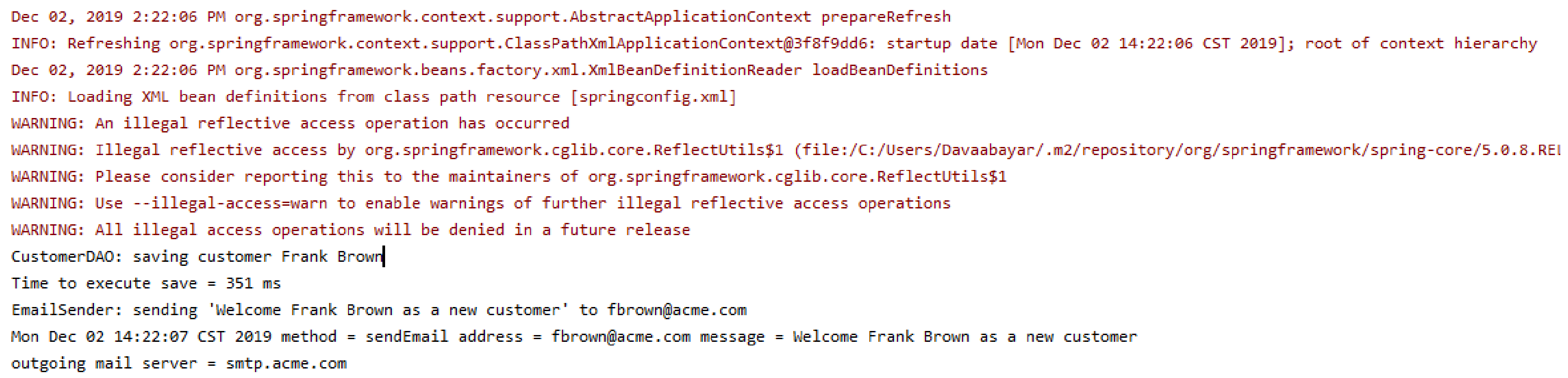
```
     ApplicationContext context = new ClassPathXmlApplicationContext("springconfig.xml");
```

CustomerService.java

```
package edu.mum.cs544;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class CustomerService implements ICustomerService {
   /*
    * 2. Property based DI START
```

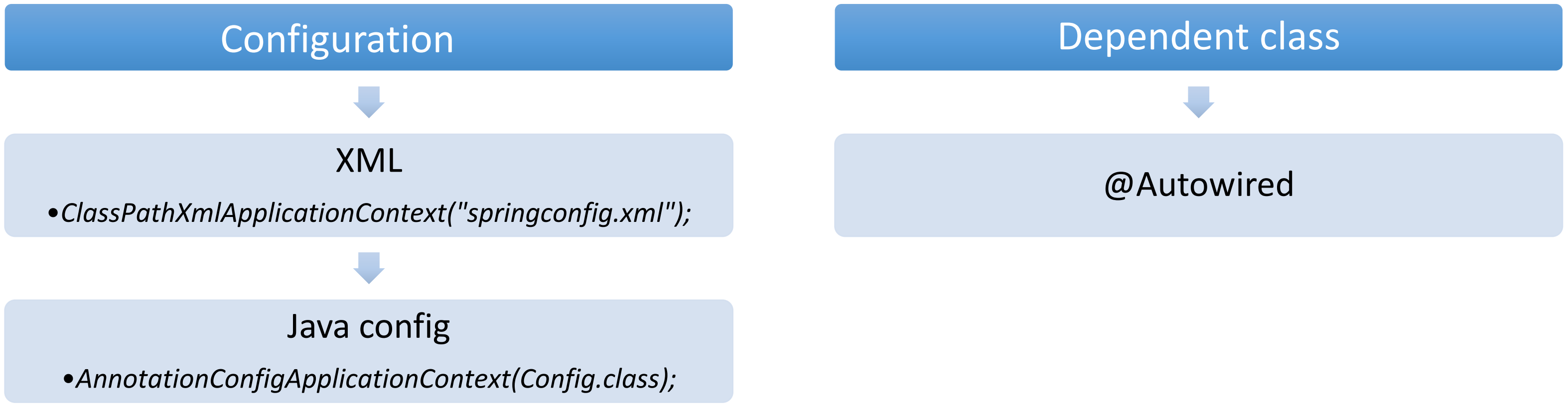
```
  * */
@Autowired
private ICustomerDAO customerDAO;
@Autowired
private IEmailSender emailSender;

public void addCustomer(String name, String email, String street,
                  String city, String zip) {
   Customer customer = new Customer(name, email);
   Address address = new Address(street, city, zip);
   customer.setAddress(address);
   customerDAO.save(customer);
   emailSender.sendEmail(email, "Welcome " + name + " as a new customer");
}
}
```

Result



```
Dec 02, 2019 2:22:06 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@3f8f9dd6: startup date [Mon Dec 02 14:22:06 CST 2019]; root of context hierarchy
Dec 02, 2019 2:22:06 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [springconfig.xml]
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.springframework.cglib.core.ReflectUtils$1 (file:/C:/Users/Davaabayar/.m2/repository/org/springframework/spring-core/5.0.8.RE
WARNING: Please consider reporting this to the maintainers of org.springframework.cglib.core.ReflectUtils$1
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
CustomerDAO: saving customer Frank Brown
Time to execute save = 351 ms
EmailSender: sending 'Welcome Frank Brown as a new customer' to fbrown@acme.com
Mon Dec 02 14:22:07 CST 2019 method = sendEmail address = fbrown@acme.com message = Welcome Frank Brown as a new customer
outgoing mail server = smtp.acme.com
```

Constructor based DI


Configuration
XML
•ClassPathXmlApplicationContext("springconfig.xml");
Java config
•AnnotationConfigApplicationContext(Config.class);
Dependent class
@Autowired

Using Xml Config

Spring.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"
      xmlns:aop="http://www.springframework.org/schema/aop"
      xmlns:context="http://www.springframework.org/schema/context"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
     http://www.springframework.org/schema/beans/spring-beans.xsd
     http://www.springframework.org/schema/aop
     http://www.springframework.org/schema/aop/spring-aop.xsd http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

   <context:component-scan base-package="edu.mum.cs544"/>
</beans>
```

App.class

```
ApplicationContext context = new ClassPathXmlApplicationContext("springconfig.xml");
```

Using Java config

Config.java

```
@Configuration
@ComponentScan("edu.mum.cs544")
@EnableAspectJAutoProxy
public class Config {

}
```

App.class

```
ApplicationContext context = new AnnotationConfigApplicationContext(Config.class);
```

CustomerService.java

```
package edu.mum.cs544;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class CustomerService implements ICustomerService {
   /*
    * 3. Constructor based DI START
    * */
   private ICustomerDAO customerDAO;
   private IEmailSender emailSender;

   public CustomerService(ICustomerDAO customerDAO, IEmailSender emailSender) {
      this.customerDAO = customerDAO;
      this.emailSender = emailSender;
      System.out.println("Constructor based DI, injected beans are: " + customerDAO + ", " +emailSender);
   }
   /*
   * Constructor based DI END
   * */
```

}

Basic AOP

A. Log EmailSender.sendMail() method

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! **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 14:09:47 GMT 2009 method= sendMail
```

LogAspect

```
@Aspect
@Component
public class LogAspect {

    @After("execution(* edu.mum.cs544.EmailSender.sendEmail(..))")
    public void logAfter(JoinPoint joinPoint){
        //A
        System.out.print(new Date() + " method=" + joinPoint.getSignature().getName());
    }
}
```

Output

```
Constructor based DI, injected beans are: edu.mum.cs544.CustomerDAO@1040be71, edu.mum.cs544.EmailSender@66982506
CustomerDAO: saving customer Frank Brown
EmailSender: sending 'Welcome Frank Brown as a new customer' to fbrown@acme.com
Mon Dec 02 15:13:28 CST 2019 method=sendEmail
```

B. Log arguments of sendEmail()

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
```

LogAspect

```
@Aspect
@Component
public class LogAspect {

    @After("execution(* edu.mum.cs544.EmailSender.sendEmail(..))")
    public void logAfter(JoinPoint joinPoint){
        //A
        System.out.print(new Date() + " method=" + joinPoint.getSignature().getName());
        //B
        System.out.print(" address="+joinPoint.getArgs()[0] + " message=" + joinPoint.getArgs()[1]);
    }

}
```

Output

```
Constructor based DI, injected beans are: edu.mum.cs544.CustomerDAO@1040be71, edu.mum.cs544.EmailSender@66982506
CustomerDAO: saving customer Frank Brown
EmailSender: sending 'Welcome Frank Brown as a new customer' to fbrown@acme.com
Mon Dec 02 15:13:01 CST 2019 method=sendEmail address=fbrown@acme.com message=Welcome Frank Brown as a new customer
```

C. Log outgoing mail server in EmailService.

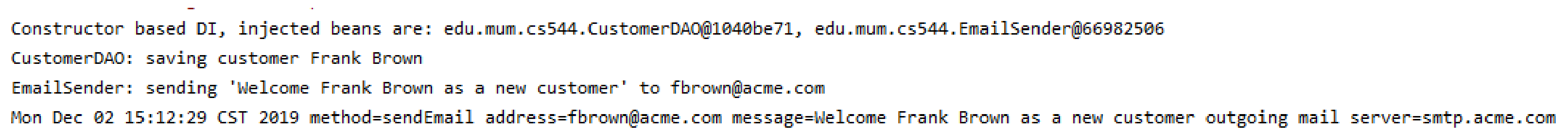
Change the log advice again, this time so that the outgoing mail server is logged as well. The outgoingMailServer is an attribute of the EmailSender object, which you can retrieve through the joinpoint.getTarget() method. **This should produce the following output:**

```
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
```

Aspect

```java
@Aspect
@Component
public class LogAspect {


      @After("execution(* edu.mum.cs544.EmailSender.sendEmail(..))")
      public void logAfter(JoinPoint joinPoint){
         //A
         System.out.print(new Date() + " method=" + joinPoint.getSignature().getName());
         //B
         System.out.print(" address="+joinPoint.getArgs()[0] + " message=" + joinPoint.getArgs()[1]);
         //C
         System.out.print(" outgoing mail server=" + ((EmailSender) joinPoint.getTarget()).getOutgoingMailServer());
      }
}
```

Output



```
Constructor based DI, injected beans are: edu.mum.cs544.CustomerDAO@1040be71, edu.mum.cs544.EmailSender@66982506
CustomerDAO: saving customer Frank Brown
EmailSender: sending 'Welcome Frank Brown as a new customer' to fbrown@acme.com
Mon Dec 02 15:12:29 CST 2019 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:

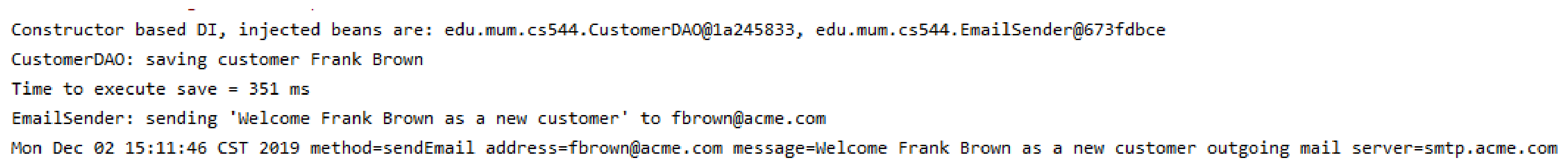
```
import org.springframework.util.StopWatch;

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
```

```
@Aspect
@Component
public class LogAspect {

    @After("execution(* edu.mum.cs544.EmailSender.sendEmail(..))")
    public void logAfter(JoinPoint joinPoint){
        //A
        System.out.print(new Date() + " method=" + joinPoint.getSignature().getName());
        //B
        System.out.print(" address="+joinPoint.getArgs()[0] + " message=" + joinPoint.getArgs()[1]);
        //C
        System.out.print(" outgoing mail server=" + ((EmailSender) joinPoint.getTarget()).getOutgoingMailServer());
    }
    @Around("execution(* edu.mum.cs544.CustomerDAO.*(..))")
    public Object invoke(ProceedingJoinPoint call) throws Throwable{
        StopWatch sw = new StopWatch();
        sw.start(call.getSignature().getName());
        Object returnVal = call.proceed();
        sw.stop();
        long totaltime = sw.getLastTaskTimeMillis();
        System.out.println("Time to execute "+call.getSignature().getName()+" = "+totaltime+" ms");
        return returnVal;
    }
}
```

Output



```
Constructor based DI, injected beans are: edu.mum.cs544.CustomerDAO@1a245833, edu.mum.cs544.EmailSender@673fdbce
CustomerDAO: saving customer Frank Brown
Time to execute save = 351 ms
EmailSender: sending 'Welcome Frank Brown as a new customer' to fbrown@acme.com
Mon Dec 02 15:11:46 CST 2019 method=sendEmail address=fbrown@acme.com message=Welcome Frank Brown as a new customer outgoing mail server=smtp.acme.com
```

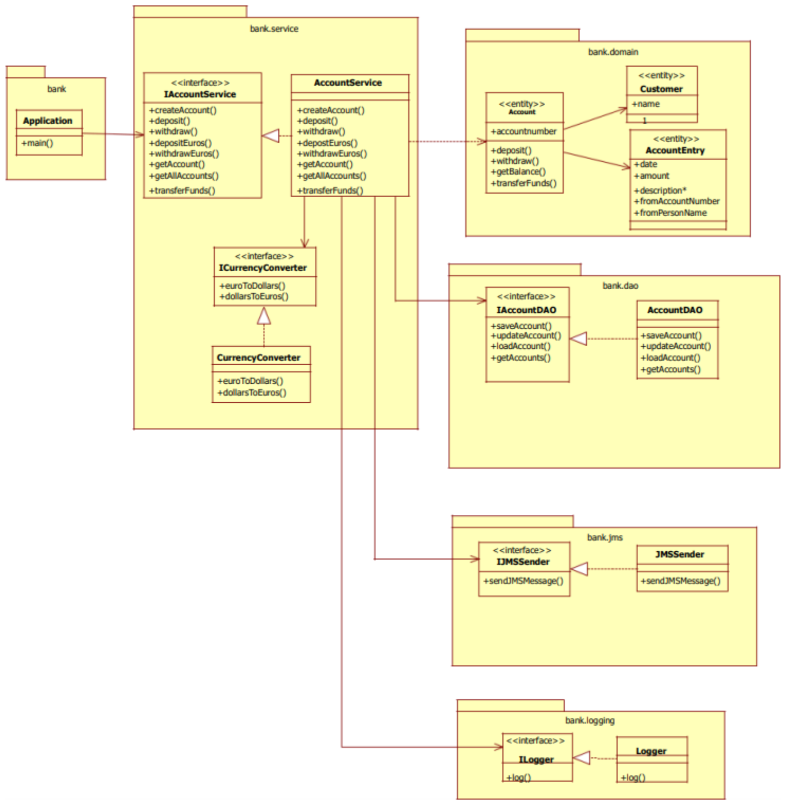
Exercise SSL.1 – Bank Application Dependency Injection

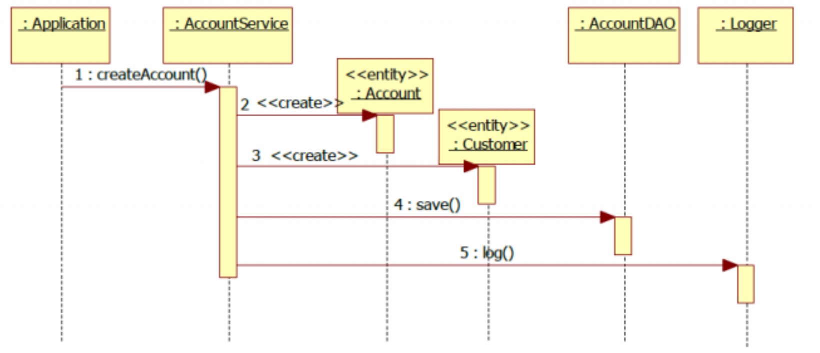
The Setup:

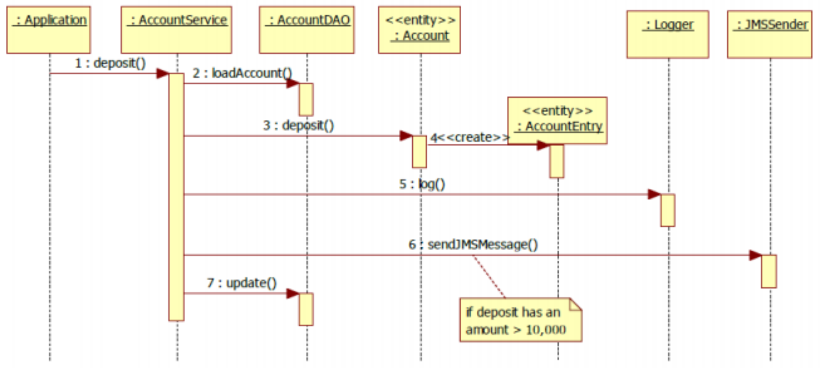
This exercise introduces the bank application. The bank application is a small application that embodies most of the architectural needs of a more real world enterprise application. Although the application that we start with in this exercise does not use any of Spring's features (yet), many areas in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

The Application:







Exersice 1

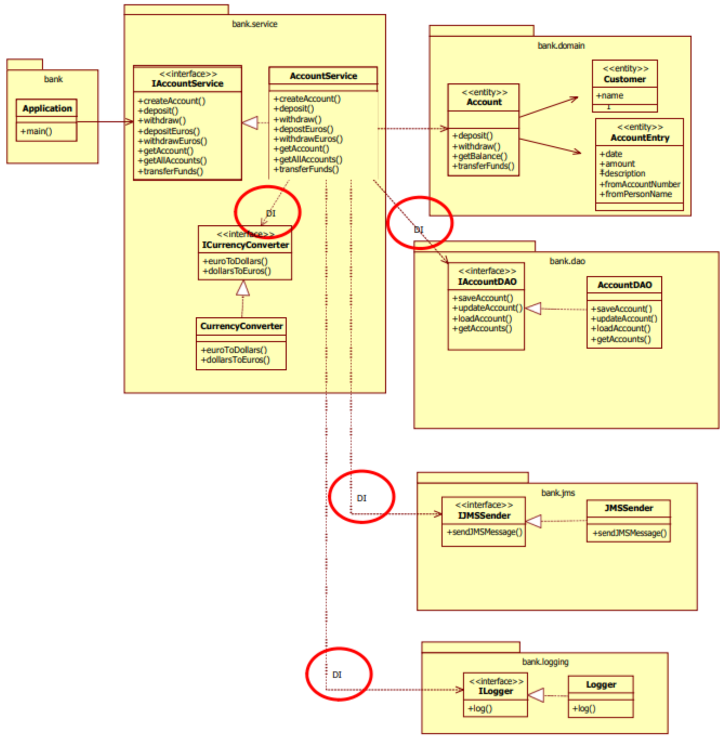
Source code: Lab2-Bank_Application

1.1 Taks : DI

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.



Solution

1. Write AppConfig.class using @Configuration, @ComponentScan(**"edu.mum.cs544.bank"**)

2. Put Component based notation to classes we need DI.

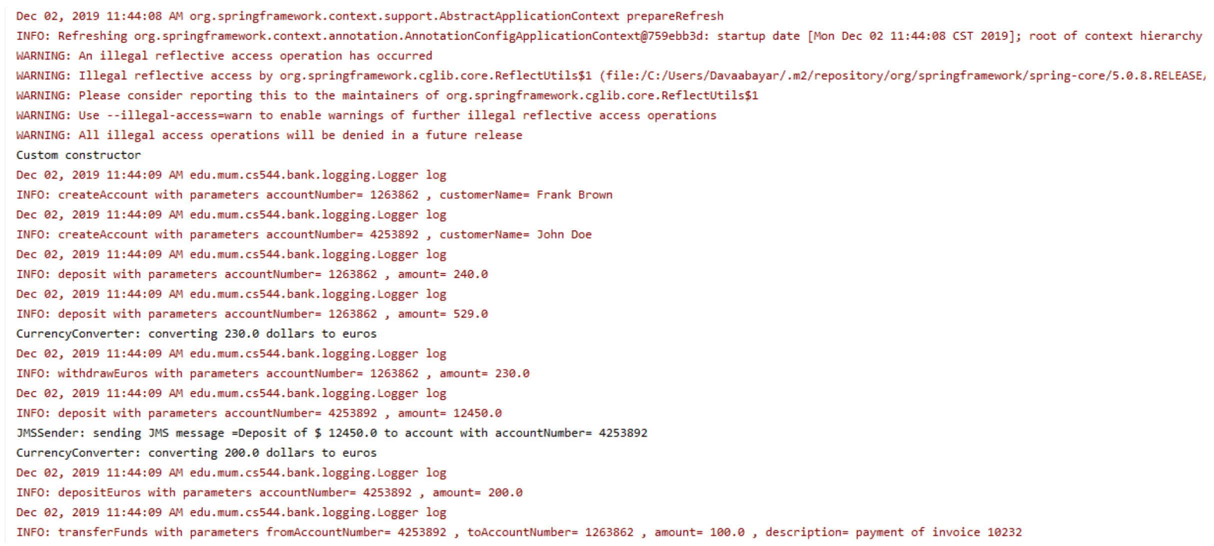
```
@Component
public class Logger implements ILogger{

@Component
public class JMSSender implements IJMSSender{

@Component
public class CurrencyConverter implements ICurrencyConverter{
@Repository
public class AccountDAO implements IAccountDAO {
```

3. Constructor based DI for AccountService.java
   a. @Service for AccountService

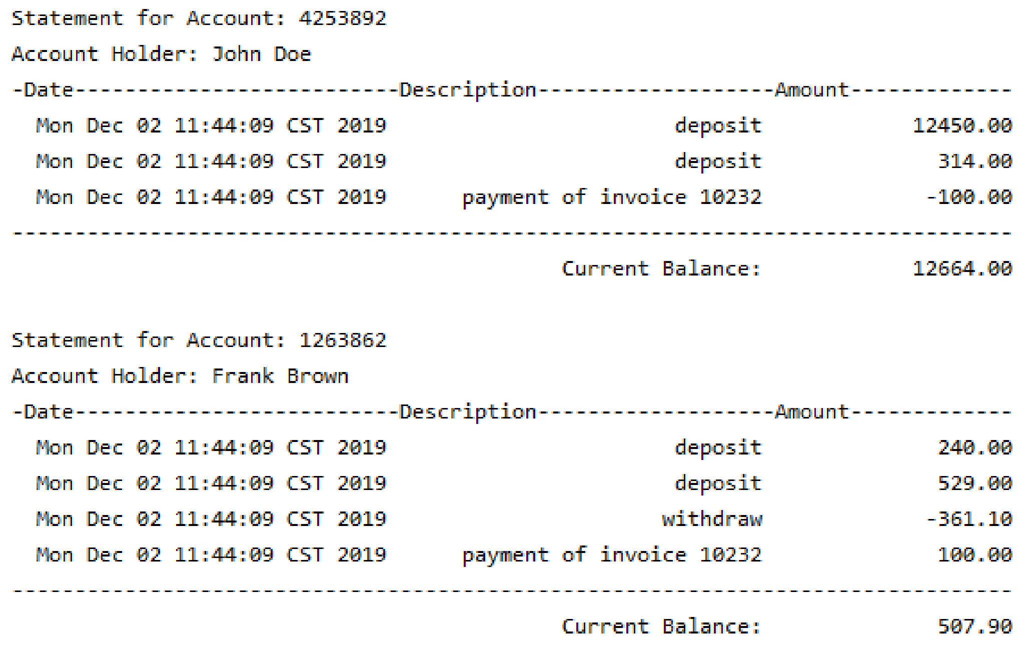
```
@Service
public class AccountService implements IAccountService {
```

   b. Remove previous constructor
   c. Add custom constructor

```
public AccountService(IAccountDAO accountDAO, ICurrencyConverter currencyConverter, IJMSSender jmsSender,
ILogger logger) {
   System.out.println("Custom constructor");
   this.accountDAO = accountDAO;
   this.currencyConverter = currencyConverter;
   this.jmsSender = jmsSender;
   this.logger = logger;
}
```



```
Dec 02, 2019 11:44:08 AM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@759ebb3d: startup date [Mon Dec 02 11:44:08 CST 2019]; root of context hierarchy
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.springframework.cglib.core.ReflectUtils$1 (file:/C:/Users/Davaabayar/.m2/repository/org/springframework/spring-core/5.0.8.RELEASE,
WARNING: Please consider reporting this to the maintainers of org.springframework.cglib.core.ReflectUtils$1
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
Custom constructor
Dec 02, 2019 11:44:09 AM edu.mum.cs544.bank.logging.Logger log
INFO: createAccount with parameters accountNumber= 1263862 , customerName= Frank Brown
Dec 02, 2019 11:44:09 AM edu.mum.cs544.bank.logging.Logger log
INFO: createAccount with parameters accountNumber= 4253892 , customerName= John Doe
Dec 02, 2019 11:44:09 AM edu.mum.cs544.bank.logging.Logger log
INFO: deposit with parameters accountNumber= 1263862 , amount= 240.0
Dec 02, 2019 11:44:09 AM edu.mum.cs544.bank.logging.Logger log
INFO: deposit with parameters accountNumber= 1263862 , amount= 529.0
CurrencyConverter: converting 230.0 dollars to euros
Dec 02, 2019 11:44:09 AM edu.mum.cs544.bank.logging.Logger log
INFO: withdrawEuros with parameters accountNumber= 1263862 , amount= 230.0
Dec 02, 2019 11:44:09 AM edu.mum.cs544.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
Dec 02, 2019 11:44:09 AM edu.mum.cs544.bank.logging.Logger log
INFO: depositEuros with parameters accountNumber= 4253892 , amount= 200.0
Dec 02, 2019 11:44:09 AM edu.mum.cs544.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-------------
  Mon Dec 02 11:44:09 CST 2019                      deposit            12450.00
  Mon Dec 02 11:44:09 CST 2019                      deposit              314.00
  Mon Dec 02 11:44:09 CST 2019      payment of invoice 10232            -100.00
--------------------------------------------------------------------------------
                                            Current Balance:           12664.00

Statement for Account: 1263862
Account Holder: Frank Brown
-Date--------------------------Description-------------------Amount-------------
  Mon Dec 02 11:44:09 CST 2019                      deposit              240.00
  Mon Dec 02 11:44:09 CST 2019                      deposit              529.00
  Mon Dec 02 11:44:09 CST 2019                     withdraw             -361.10
  Mon Dec 02 11:44:09 CST 2019      payment of invoice 10232             100.00
--------------------------------------------------------------------------------
                                            Current Balance:             507.90
```

Excersice 2

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).

Use AOP to:

1. Log every call to any method in the bank.dao package (using the Logger).
2. 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.
3. Log every JMS message that is sent (using the Logger)
4. 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.
5. Be sure to inject the logger into the advice class as shown below.

Source code: Lab2-AOP-2

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

1. Add dependency

```
<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>
```

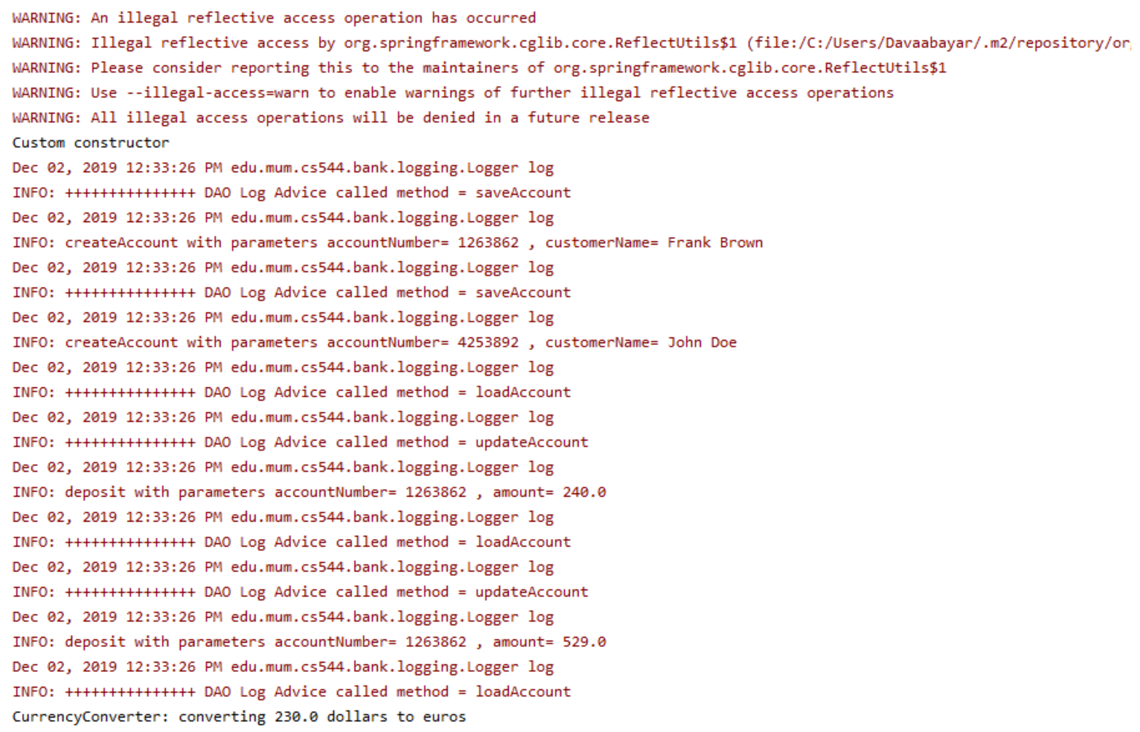
2. Enable JautoProxy in AppConfig

```
@Configuration
@ComponentScan("edu.mum.cs544.bank")
@EnableAspectJAutoProxy
```

```
public class AppConfig {
}
```

3. Add new Aspect ***DaoLogAspect.java***

```
import edu.mum.cs544.bank.logging.ILogger;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Aspect
@Component
public class DoaLogAdvice {
    @Autowired
    private ILogger iLogger;

    @After("execution(* edu.mum.cs544.bank.dao.*.*(..))")
    public void log(JoinPoint joinPoint){
        iLogger.log("+++++++++++++++ DAO Log Advice called method = " + joinPoint.getSignature().getName());
    }
}
```



```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.springframework.cglib.core.ReflectUtils$1 (file:/C:/Users/Davaabayar/.m2/repository/or
WARNING: Please consider reporting this to the maintainers of org.springframework.cglib.core.ReflectUtils$1
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
Custom constructor
Dec 02, 2019 12:33:26 PM edu.mum.cs544.bank.logging.Logger log
INFO: +++++++++++++++ DAO Log Advice called method = saveAccount
Dec 02, 2019 12:33:26 PM edu.mum.cs544.bank.logging.Logger log
INFO: createAccount with parameters accountNumber= 1263862 , customerName= Frank Brown
Dec 02, 2019 12:33:26 PM edu.mum.cs544.bank.logging.Logger log
INFO: +++++++++++++++ DAO Log Advice called method = saveAccount
Dec 02, 2019 12:33:26 PM edu.mum.cs544.bank.logging.Logger log
INFO: createAccount with parameters accountNumber= 4253892 , customerName= John Doe
Dec 02, 2019 12:33:26 PM edu.mum.cs544.bank.logging.Logger log
INFO: +++++++++++++++ DAO Log Advice called method = loadAccount
Dec 02, 2019 12:33:26 PM edu.mum.cs544.bank.logging.Logger log
INFO: +++++++++++++++ DAO Log Advice called method = updateAccount
Dec 02, 2019 12:33:26 PM edu.mum.cs544.bank.logging.Logger log
INFO: deposit with parameters accountNumber= 1263862 , amount= 240.0
Dec 02, 2019 12:33:26 PM edu.mum.cs544.bank.logging.Logger log
INFO: +++++++++++++++ DAO Log Advice called method = loadAccount
Dec 02, 2019 12:33:26 PM edu.mum.cs544.bank.logging.Logger log
INFO: +++++++++++++++ DAO Log Advice called method = updateAccount
Dec 02, 2019 12:33:26 PM edu.mum.cs544.bank.logging.Logger log
INFO: deposit with parameters accountNumber= 1263862 , amount= 529.0
Dec 02, 2019 12:33:26 PM edu.mum.cs544.bank.logging.Logger log
INFO: +++++++++++++++ DAO Log Advice called method = loadAccount
CurrencyConverter: converting 230.0 dollars to euros
```

2.2 Use the Spring StopWatch functionality to measure the duration of all service

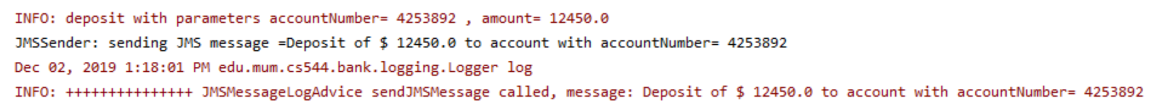
Use @Around advice

```java
package edu.mum.cs544.bank;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.stereotype.Component;
import org.springframework.util.StopWatch;

@Aspect
@Component
public class MeasureServiceAdvice {
    @Around("execution(* edu.mum.cs544.bank.service.*.*(..))")
    public Object measureDuration(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {
        StopWatch sw = new StopWatch();
        sw.start(proceedingJoinPoint.getSignature().getName());
        Object retVal = proceedingJoinPoint.proceed();
        sw.stop();
        long totaltime = sw.getLastTaskTimeMillis();
        System.out.print("Duration => "+ proceedingJoinPoint.getTarget().getClass() + "." +
proceedingJoinPoint.getSignature().getName() + "(");
        Object[] args = proceedingJoinPoint.getArgs();
        for(int i=0; i<args.length; i++){
            System.out.print(" " + args[i] + " ");
        }
        System.out.print(") = " + totaltime + "ms\n");
        return retVal;
    }
}
```

```
Dec 02, 2019 12:54:45 PM edu.mum.cs544.bank.logging.Logger log
INFO: +++++++++++++++ DAO Log Advice called method = saveAccount
Dec 02, 2019 12:54:45 PM edu.mum.cs544.bank.logging.Logger log
INFO: createAccount with parameters accountNumber= 1263862 , customerName= Frank Brown
Duration => class edu.mum.cs544.bank.service.AccountService.createAccount( 1263862  Frank Brown ) = 22ms
Dec 02, 2019 12:54:45 PM edu.mum.cs544.bank.logging.Logger log
INFO: +++++++++++++++ DAO Log Advice called method = saveAccount
Dec 02, 2019 12:54:45 PM edu.mum.cs544.bank.logging.Logger log
INFO: createAccount with parameters accountNumber= 4253892 , customerName= John Doe
Duration => class edu.mum.cs544.bank.service.AccountService.createAccount( 4253892  John Doe ) = 2ms
```

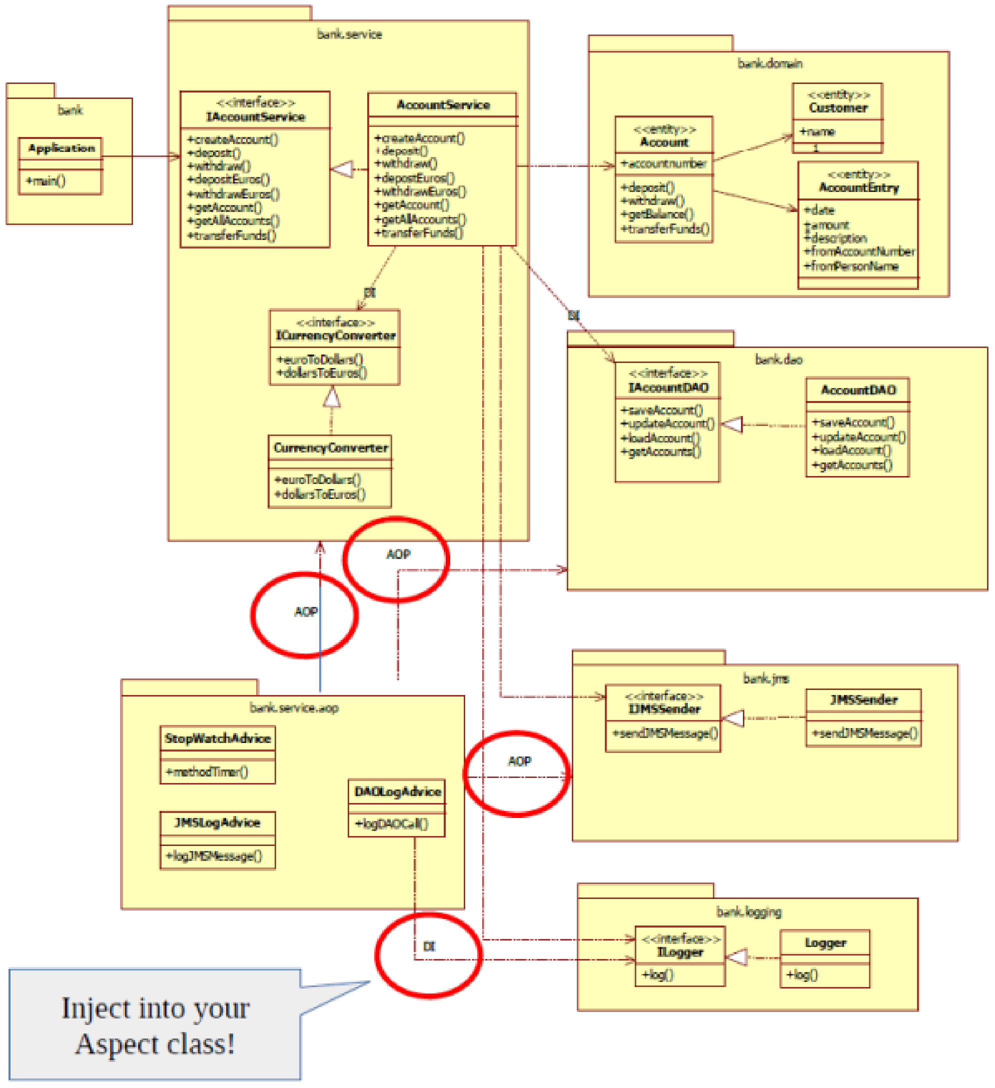
2.3 Log every JMS message that is sent (using the Logger)

```
import edu.mum.cs544.bank.logging.ILogger;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Aspect
@Component
public class JMSMessageLogAdvice {
    @Autowired
    private ILogger iLogger;

    @After("execution(* edu.mum.cs544.bank.jms.JMSSender.sendJMSMessage(..))")
    public void log(JoinPoint joinPoint){
        iLogger.log("++++++++++++++ JMSMessageLogAdvice " + joinPoint.getSignature().getName() + " called, message: 
" + joinPoint.getArgs()[0]);
    }
}
```

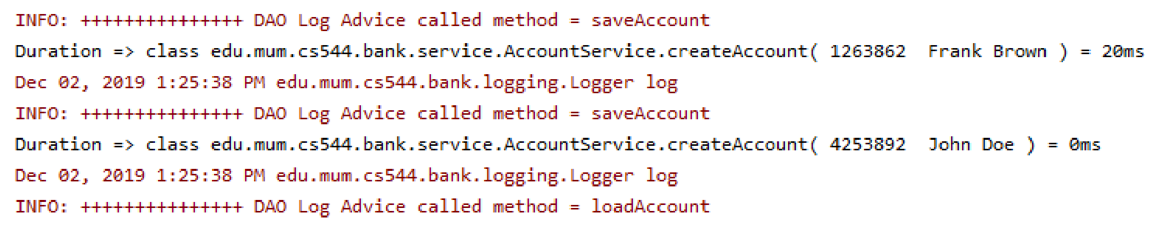


```
INFO: deposit with parameters accountNumber= 4253892 , amount= 12450.0
JMSSender: sending JMS message =Deposit of $ 12450.0 to account with accountNumber= 4253892
Dec 02, 2019 1:18:01 PM edu.mum.cs544.bank.logging.Logger log
INFO: ++++++++++++++ JMSMessageLogAdvice sendJMSMessage called, message: Deposit of $ 12450.0 to account with accountNumber= 4253892
```

2.5 Be sure to inject the logger into the advice class as shown below.



```java
@Aspect
@Component
public class DoaLogAdvice {
    @Autowired
    private ILogger iLogger;

    @After("execution(* edu.mum.cs544.bank.dao.*.*(..))")
    public void logDAOCall(JoinPoint joinPoint){
        iLogger.log("+++++++++++++++ DAO Log Advice called method = " + joinPoint.getSignature().getName());
    }
}
```



```
INFO: +++++++++++++++ DAO Log Advice called method = saveAccount
Duration => class edu.mum.cs544.bank.service.AccountService.createAccount( 1263862  Frank Brown ) = 20ms
Dec 02, 2019 1:25:38 PM edu.mum.cs544.bank.logging.Logger log
INFO: +++++++++++++++ DAO Log Advice called method = saveAccount
Duration => class edu.mum.cs544.bank.service.AccountService.createAccount( 4253892  John Doe ) = 0ms
Dec 02, 2019 1:25:38 PM edu.mum.cs544.bank.logging.Logger log
INFO: +++++++++++++++ DAO Log Advice called method = loadAccount
```