Contents

[Exercise AOP.1 – Basic Spring AOP 2](#_Toc26192936)

[The Exercise 1 4](#_Toc26192937)

[DI 4](#_Toc26192938)

[Basic AOP 11](#_Toc26192939)

[Exercise SSL.1 – Bank Application Dependency Injection 15](#_Toc26192940)

[Exersice 1 16](#_Toc26192941)

[1.1 Taks : DI 17](#_Toc26192942)

[Excersice 2 20](#_Toc26192943)

[2.1 Log every call to any method in the bank.dao package (using the Logger). 20](#_Toc26192944)

[2.2 Use the Spring StopWatch functionality to measure the duration of all service 22](#_Toc26192945)

[2.3 Log every JMS message that is sent (using the Logger) 23](#_Toc26192946)

[2.5 Be sure to inject the logger into the advice class as shown below. 25](#_Toc26192947)

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**

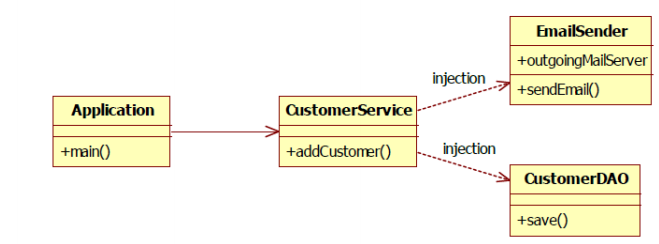
# Exercise AOP.1 – Basic Spring AOP

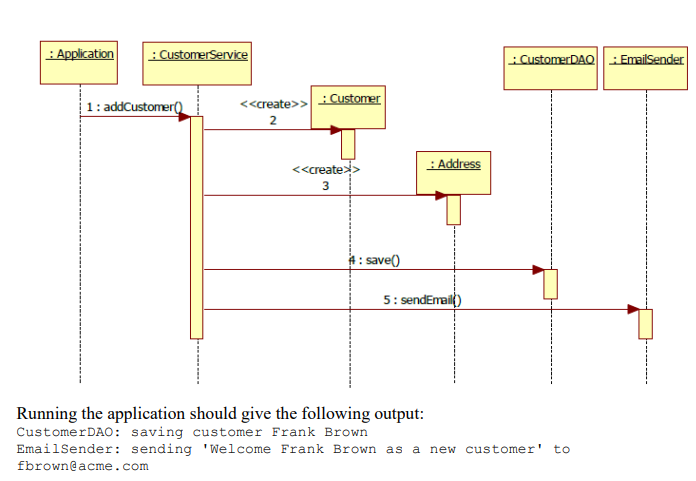
**The Setup:**

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.





## 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**

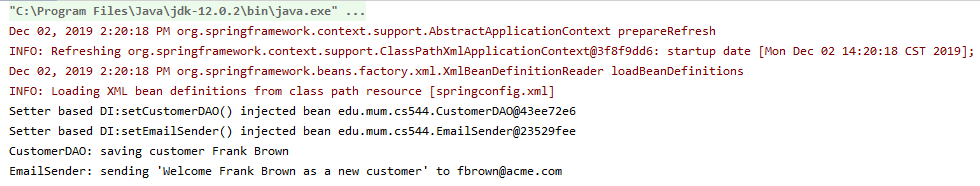
**public class** App   
{  
 **public static void** main(String[] args) {  
ApplicationContext context = **new** ClassPathXmlApplicationContext(**"springconfig.xml"**);  
 ICustomerService customerService = context.getBean(**"customerService"**, ICustomerService.**class**);  
  
 customerService.addCustomer(**"Frank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



#### Property Based DI

##### 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**



#### Constructor based DI

##### 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**);

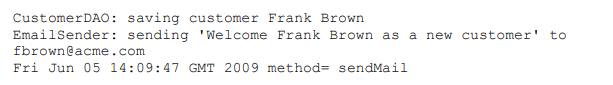
**CustomerService.java**

**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

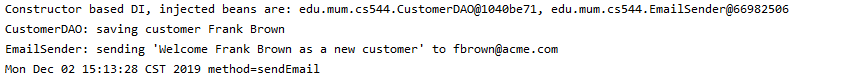
Reconfigure the application so that whenever the sendMail method on the EmailSender is called, a log message is created (using an after advice AOP annotation). Remember to configure Spring to look for your aspect annotations! **This should produce the following output:**



**LogAspect**

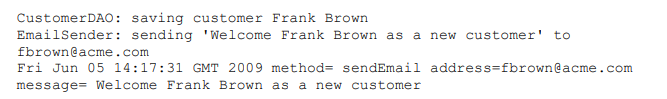
@Aspect  
@Component  
**public class** LogAspect {  
  
 @After(**"execution(\* edu.mum.cs544.EmailSender.sendEmail(..))"**)  
 **public void** logAfter(JoinPoint joinPoint){  
 *//A* System.***out***.print(**new** Date() + **" method="** + joinPoint.getSignature().getName());  
 }  
}

**Output**



#### B. Log arguments of sendEmail()

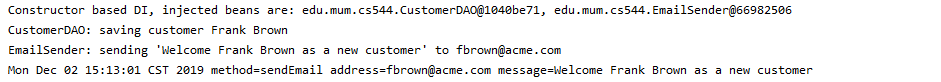
Now change the log advice in such a way that the email address and the message are logged as well. You should be able to retrieve the email address and the message through the arguments of the sendEmail() method. **This should produce the following output:**



**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**



#### 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:**



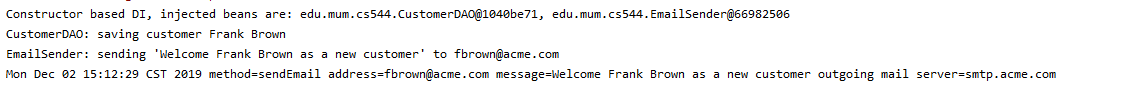
**Aspect**

@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());  
}

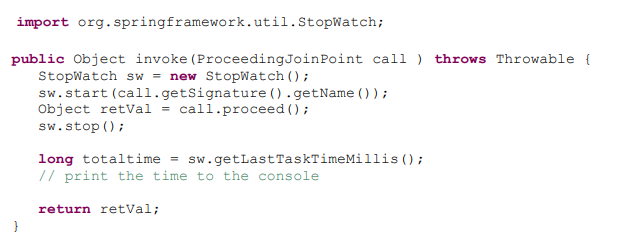
}

**Output**

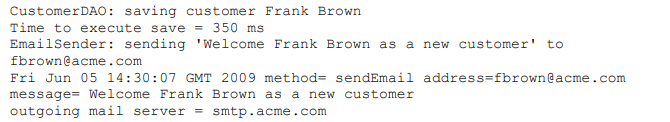


#### D. Write a new advice that calculates the duration of the method calls to the DAO

object and outputs the result to the console. Spring provides a stopwatch utility that can be used for this by using the following code:

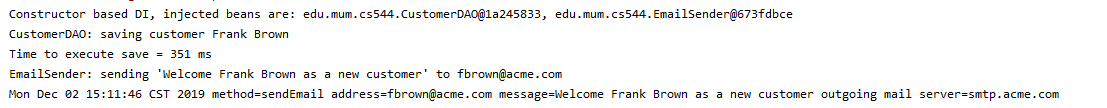


**This should produce the following output:**



@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**



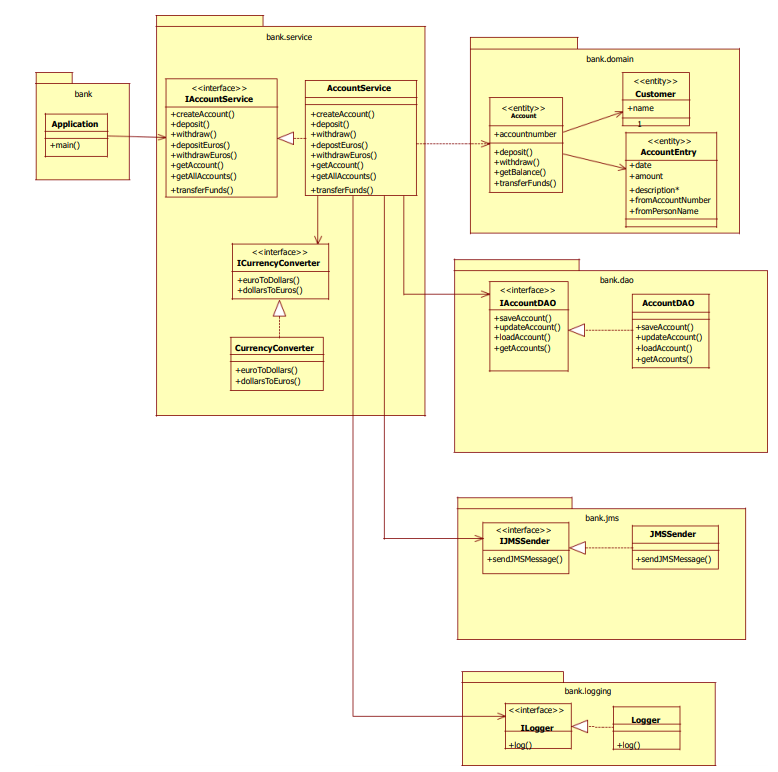
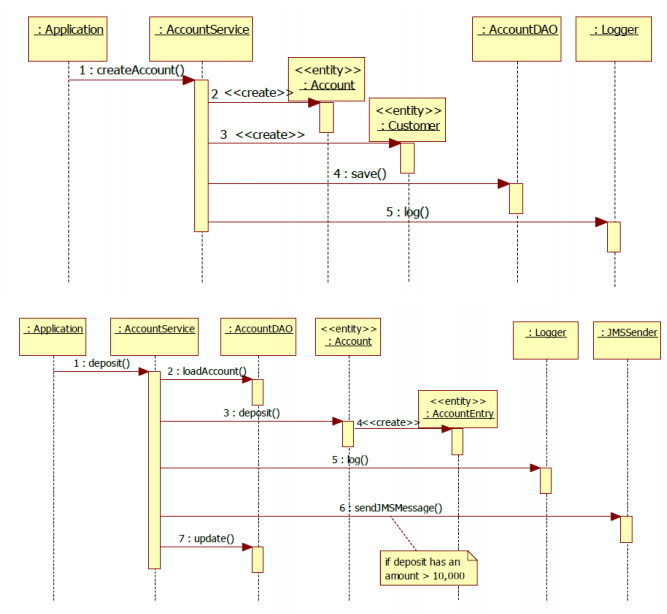
# Exercise SSL.1 – Bank Application Dependency Injection

**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.

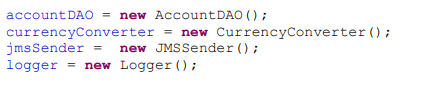
**The Application:**



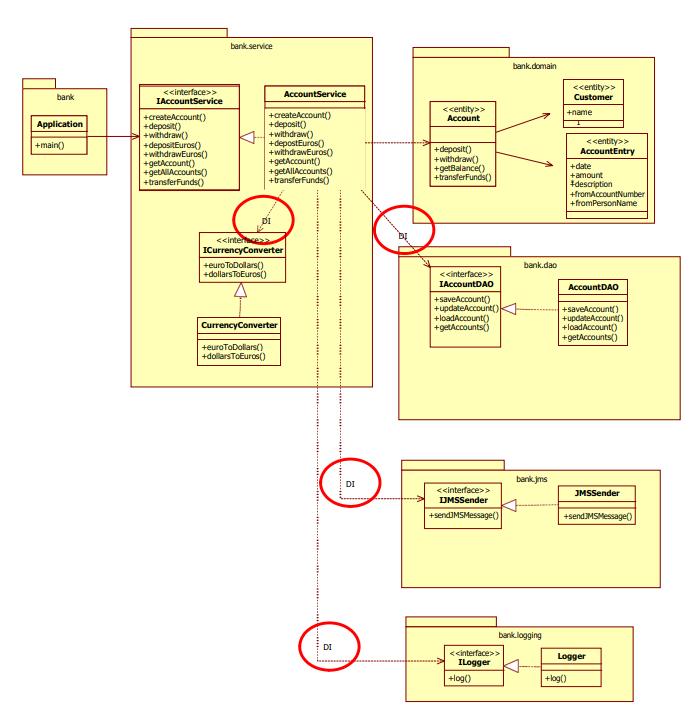
## Exersice 1

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:

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{

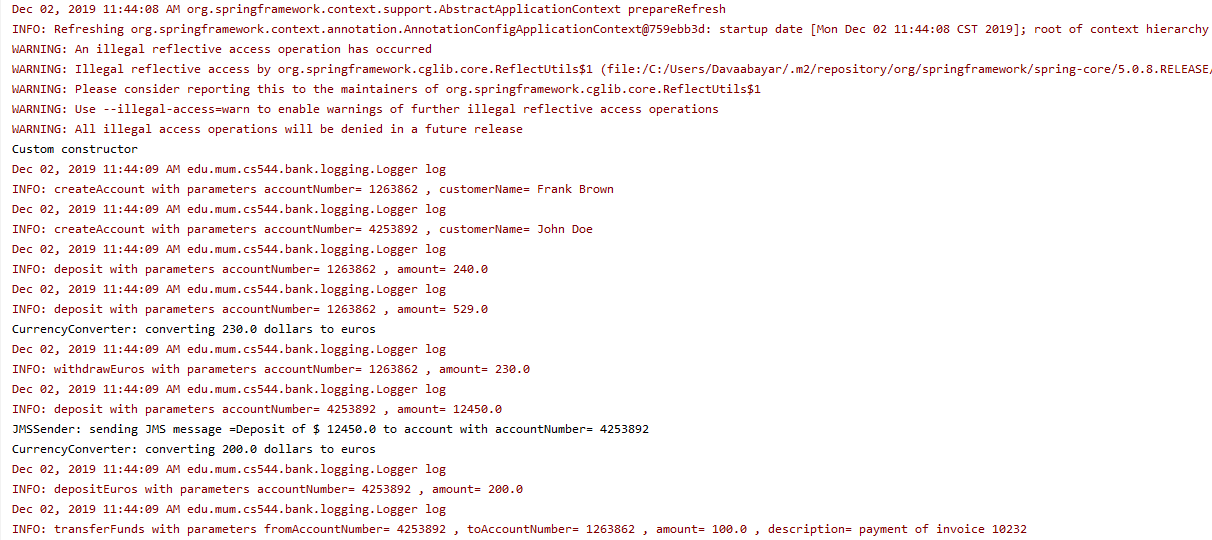
@Repository  
**public class** AccountDAO **implements** IAccountDAO {

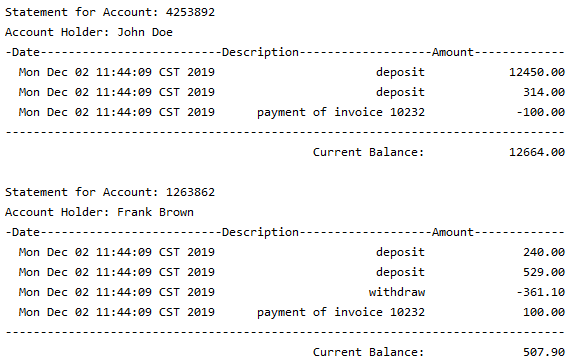
1. Constructor based DI for AccountService.java
   1. @Service for AccountService

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

* 1. Remove previous constructor
  2. 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;  
}





## 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

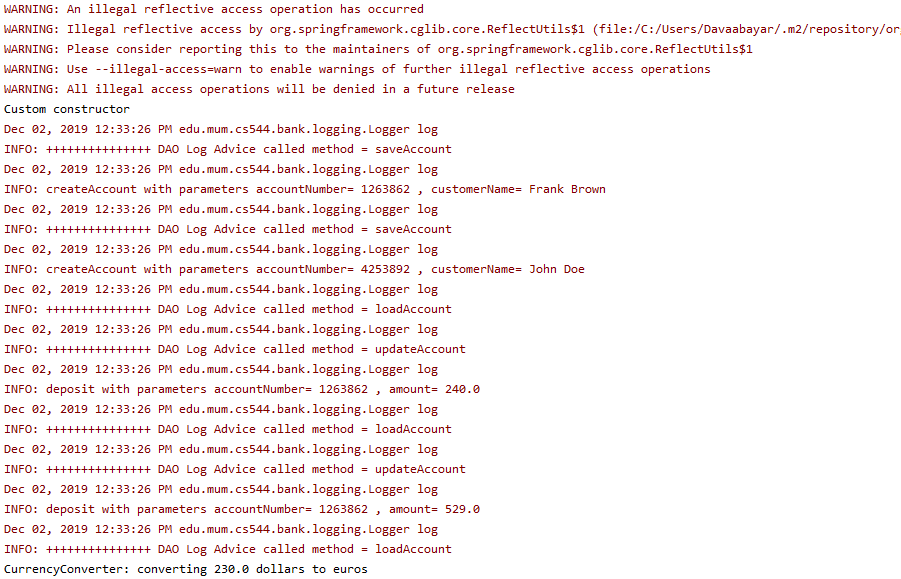
<**dependency**>  
 <**groupId**>org.aspectj</**groupId**>  
 <**artifactId**>aspectjrt</**artifactId**>  
 <**version**>1.9.2</**version**>  
</**dependency**>  
<**dependency**>  
 <**groupId**>org.aspectj</**groupId**>  
 <**artifactId**>aspectjweaver</**artifactId**>  
 <**version**>1.9.2</**version**>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

1. Enable JautoProxy in AppConfig

@Configuration  
@ComponentScan(**"edu.mum.cs544.bank"**)  
@EnableAspectJAutoProxy  
**public class** AppConfig {  
}

1. Add new Aspect ***DaoLogAspect.java***

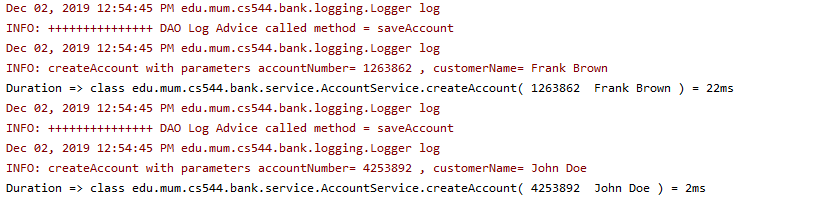
**import** edu.mum.cs544.bank.logging.ILogger;  
**import** org.aspectj.lang.JoinPoint;  
**import** org.aspectj.lang.annotation.After;  
**import** org.aspectj.lang.annotation.Aspect;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.stereotype.Component;  
  
@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());  
 }  
}



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

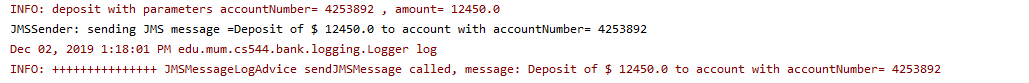
Use @Around advice

**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;  
 }  
}

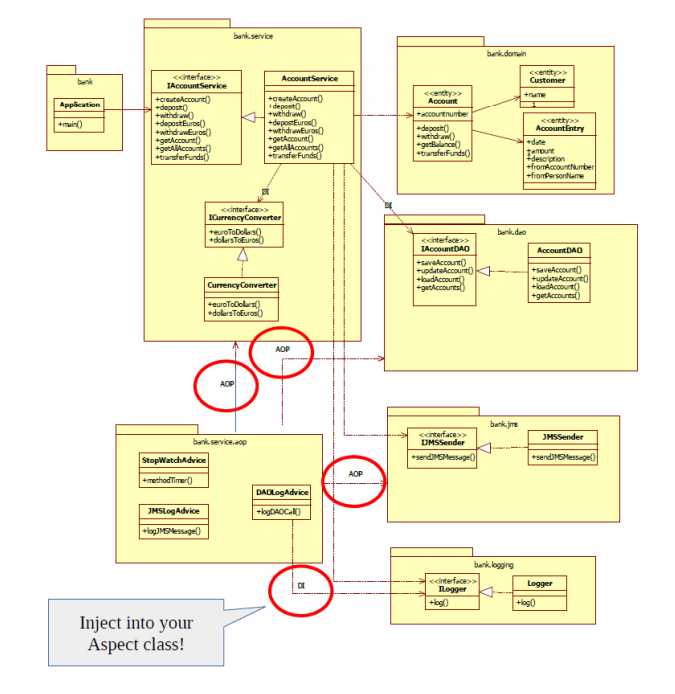


### 2.3 Log every JMS message that is sent (using the Logger)

**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]);  
 }  
}



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



@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());  
 }  
}

