Lesson 7: Aspect Oriented Programming

Exercise 6.1 – Basic Spring AOP

The Setup:

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

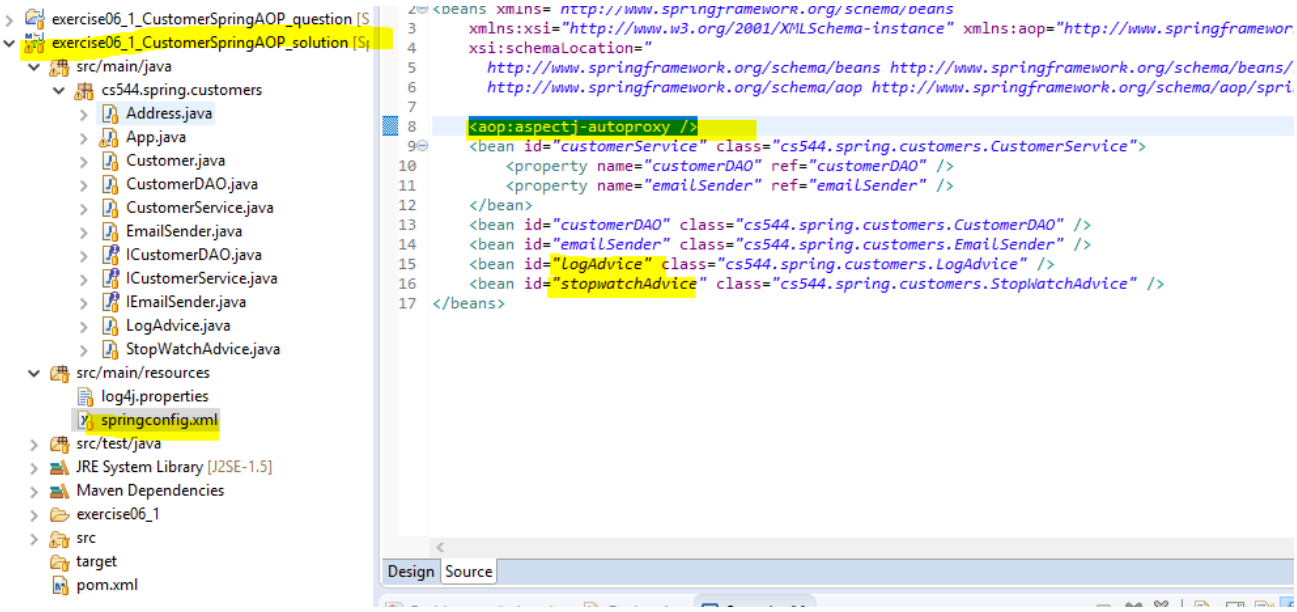
Start by opening exercise6_1 from C:\CS544\exercises\ and add the **Spring dependencies** to it, and then add the following AspectJ dependencies as well:

```
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjrt</artifactId>
    <version>1.8.5</version>
</dependency>
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.8.5</version>
</dependency>
```

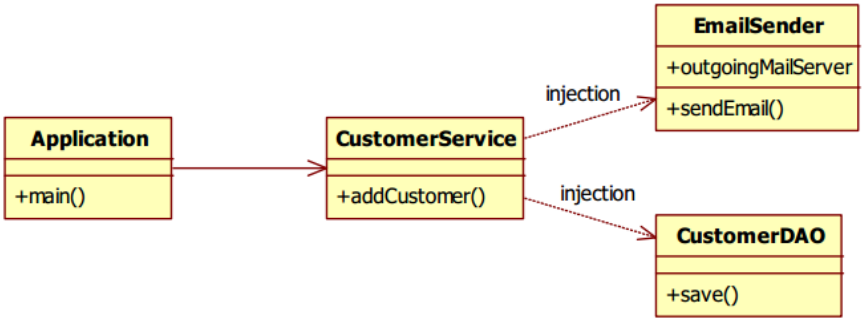
<dependency>
<groupId>org.aspectj</groupId>
<artifactId>aspectjrt</artifactId>
<version>1.8.5</version>
</dependency>
<dependency>
<groupId>org.aspectj</groupId>
<artifactId>aspectjweaver</artifactId>
<version>1.8.5</version>
</dependency>

Also be aware that your springconfig.xml file for this exercise will require the aop namespace.

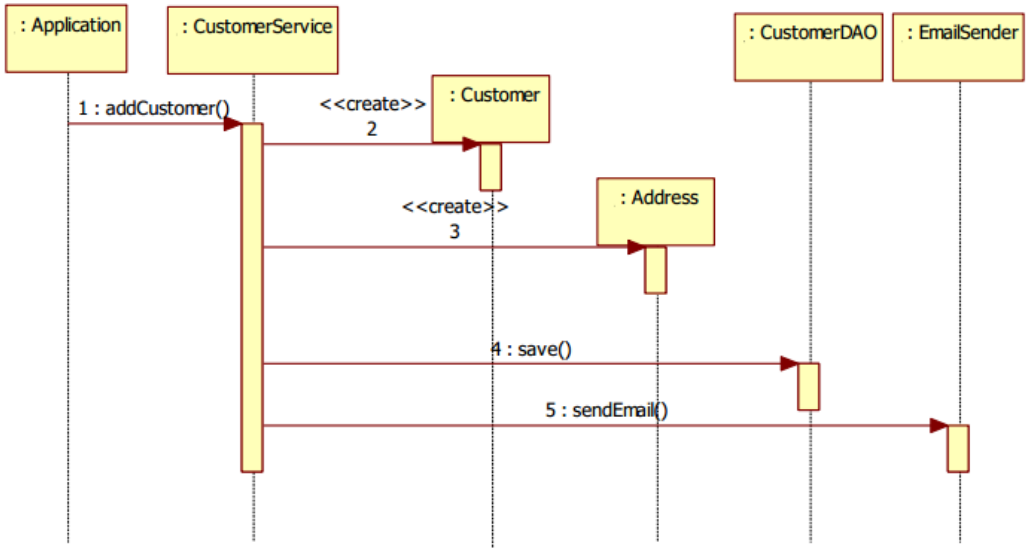
<aop:aspectj-autoproxy />



The Application:



The application provided 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.



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). 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
```

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
```

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
```

```java
package cs544.spring.customers;

import java.util.Date;

@Aspect
public class LogAdvice {
    @After("execution(* cs544.spring.customers.EmailSender.sendEmail(..)) && args(email, message)")
    public void log(JoinPoint joinpoint, String email, String message) {
        System.out.println("---LogAdvice: "+ new Date() //now
                + "\n method= "+ joinpoint.getSignature().getName() //method name
                + "\n email address= "+ email //args of the method
                + "\n message= " + message);//args of the method
        IEmailSender emailSender = (IEmailSender) joinpoint.getTarget();//get the object
        System.out.println("outgoing mail server = "+emailSender.getOutgoingMailServer());
    }

}
```

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;
}
```

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
```

- exercise06_1_CustomerSpringAOP_solution
  - src/main/java
    - cs544.spring.customers
      - Address.java
      - App.java
      - Customer.java
      - CustomerDAO.java
      - CustomerService.java
      - EmailSender.java
      - ICustomerDAO.java
      - ICustomerService.java
      - IEmailSender.java
      - LogAdvice.java
      - StopWatchAdvice.java
  - src/main/resources
    - log4j.properties
    - springconfig.xml
  - src/test/java

```java
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.*;
import org.springframework.util.StopWatch;
@Aspect
public class StopWatchAdvice {

    @Around("execution(* cs544.spring.customers.CustomerDAO.*(..))")
    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();
        System.out.println("-----------StopWatchAdvice:  "+"Time to execute "
        +call.getSignature().getName()+" = "+totaltime+" ms");

        return retVal;

    }
```