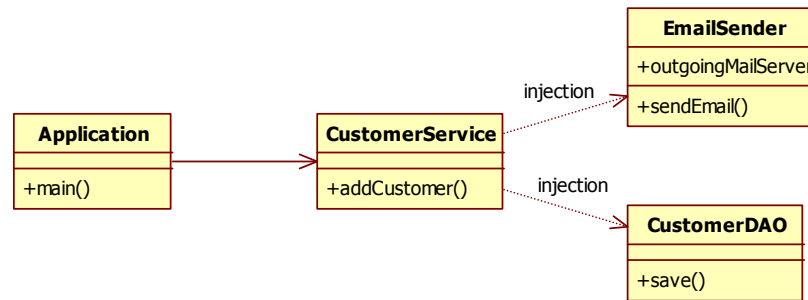
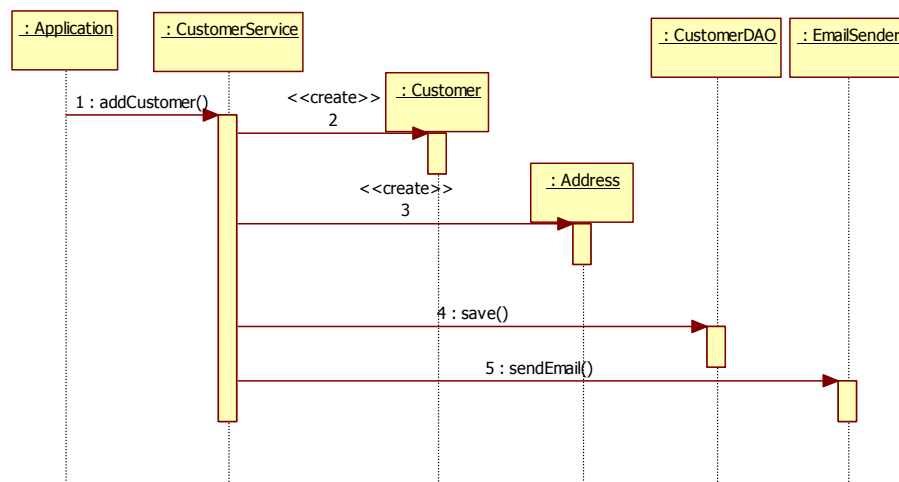


Spring AOP Lab

Import the given CustomerApplication (in the directory given applications) into Eclipse. Then add all JAR files in the directory **libraries** to the classpath of this 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
```

- 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
```

- 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
```

The last part of this lab is using AOP in our Bank application.

Start with the bank application project that was the result of the Spring DI lab.

Add the following functionality with AOP:

- Log all database access to the console
- Measure the time for all service level classes