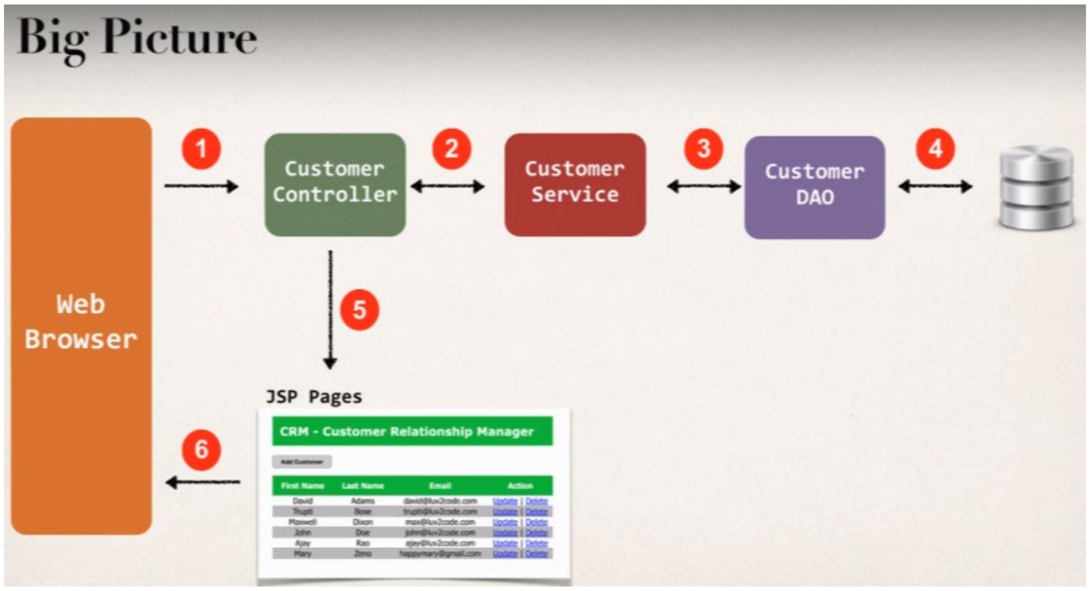
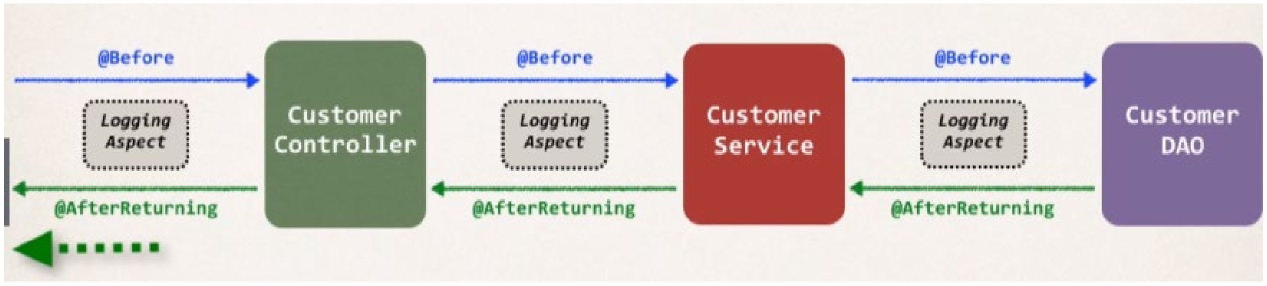
Section 45: AOP: Add AOP Logging to Spring MVC App - Real-Time Project
1 / 4 | 30min

➢ **Add AOP logging support to our Spring MVC app (CRM Web app)**



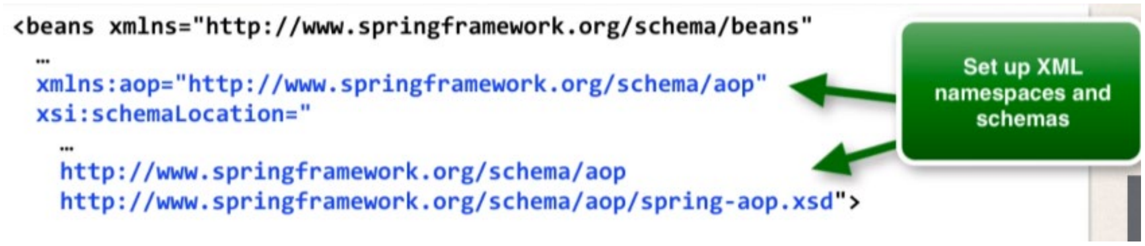
Logging Aspect



Development Process

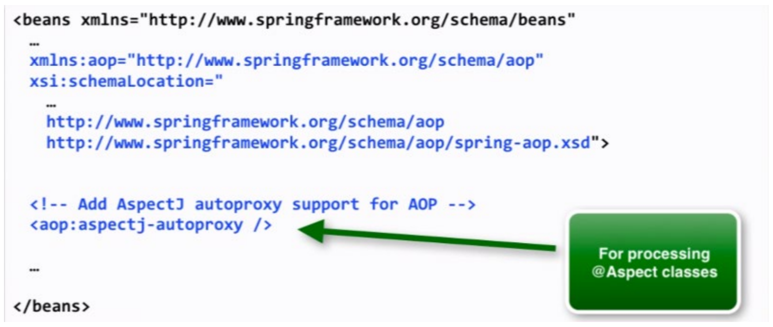
1. Add AspectJ Jar File to web project
   Add jar file to your web project: **WEB-INF/lib** directory

2. Enable AspectJ Auto Proxy



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

Now we need to enable the AutoProxy support, and it will allow the spring to do the processing on @Aspect classes that we create using spring AOP support.

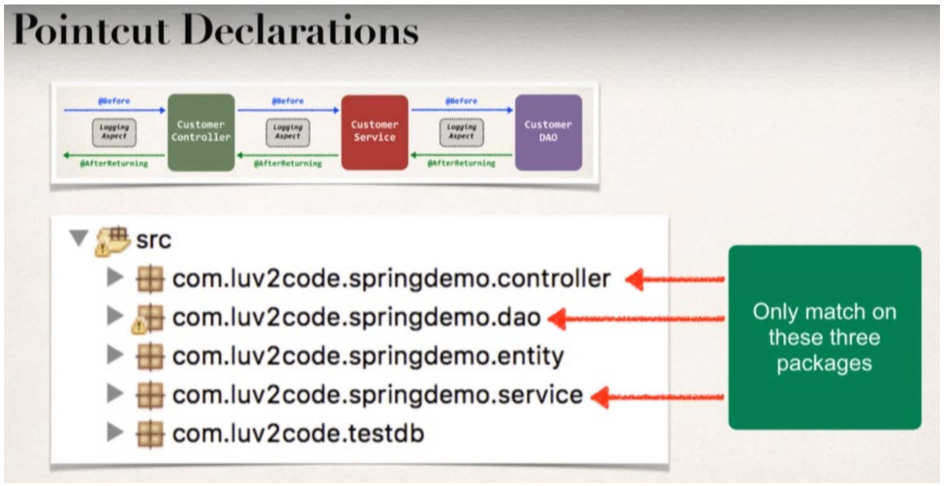


```
<beans xmlns="http://www.springframework.org/schema/beans"
  …
  xmlns:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="
    …
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop.xsd">

  <!-- Add AspectJ autoproxy support for AOP -->
  <aop:aspectj-autoproxy />

  …

</beans>
```

We can also the do the same thing if we are using pure java configuration by adding the annotation **`@EnableAspectJAutoProxy`** into our config file and then it will enable the processing for Aspect classes.

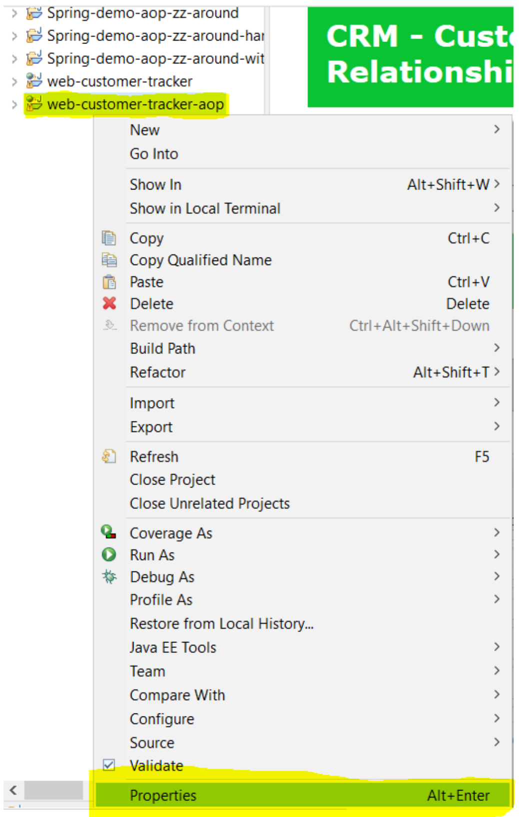
3. Create Aspect
    - Adding logging support
    - Setup pointcut declaration
    - Add @Before advice
    - Add @AfterReturning advice

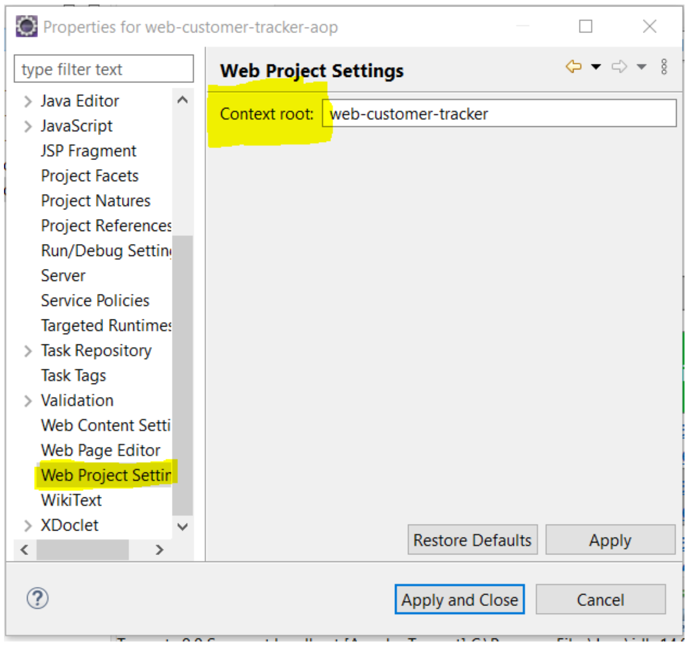
❖ We will create individual declarations for all these packages and we will combine them together using the techniques that we have learnt in the Spring AOP and then apply for our logging services in Spring MVC web App.


Pointcut Declarations
@Before
Logging Aspect
Customer Controller
@AfterReturning
Customer Service
Customer DAO
src
com.luv2code.springdemo.controller
com.luv2code.springdemo.dao
com.luv2code.springdemo.entity
com.luv2code.springdemo.service
com.luv2code.testdb
Only match on these three packages

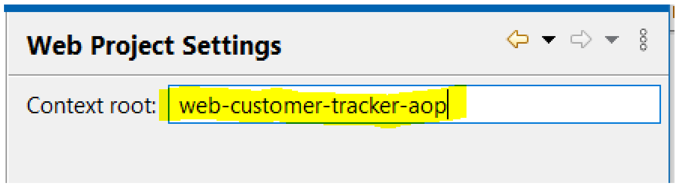
❖ **Project prep Work:**
1. Copy the previous Spring MVC CRM web application and paste with new name `web-customer-tracker-aop.`
2. Now change the Context-root:
    - Go to `Properties`
    - Go to `Web Project Settings`
    - We need to update the `Context-root`.
        - Update it with **<u>`web-customer-tracker-aop`</u>**

Now our web application will open with new url having “aop” appened to it.





Context root: web-customer-tracker



Context root: web-customer-tracker-aop

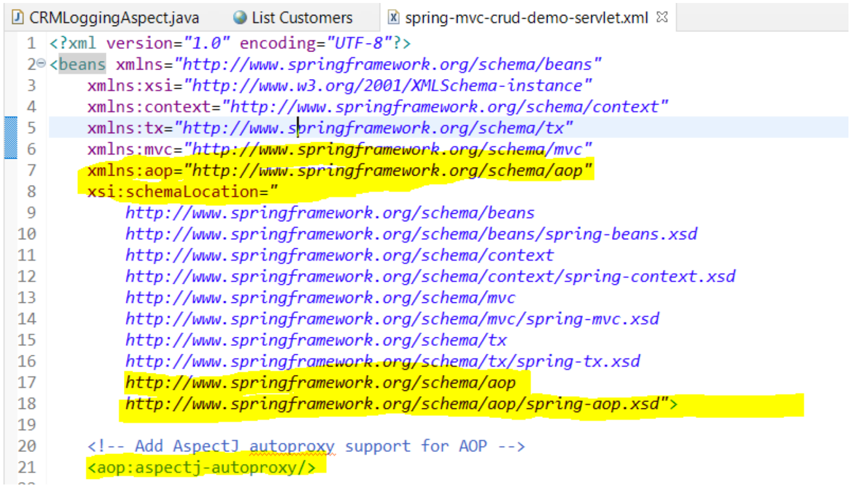
Now in the link to get the index page of our web app; we need to provide

http://localhost:8080/web-customer-tracker-aop/customer/list

So that we can differentiate between our previous web application and this new web app with AOP.

❖ **Updating the Config file:**

File: spring-mvc-crud-demo-servlet.xml



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

    <!-- Add AspectJ autoproxy support for AOP -->
    <aop:aspectj-autoproxy/>
```

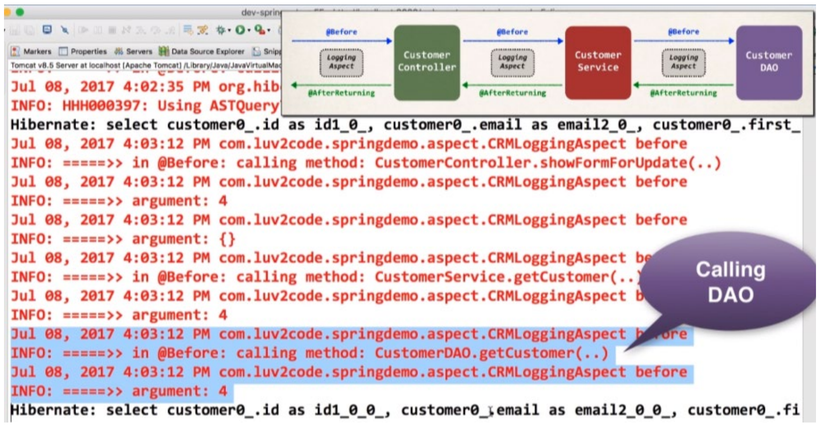
Now we need to create a new package with name aspect where we will define our aspect classes having some advices.

Package name: `com.luv2code.springdemo.aspect`

Class name: CRMLoggingAspect.java
Here we have defined the advices for the controller, service, and dao packages only.

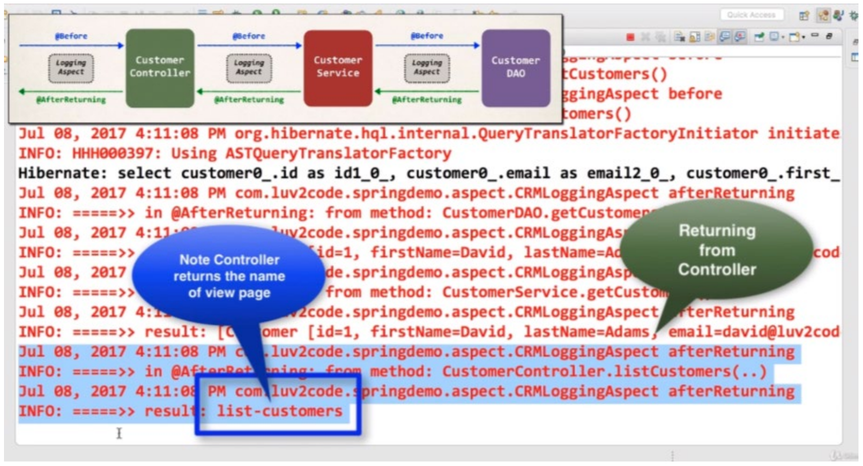
```java
@Aspect
@Component
public class CRMLoggingAspect {

    //setup Logger
    private Logger myLogger = Logger.getLogger(getClass().getName());

    //setup pointcut declarations
    @Pointcut("execution(* com.luv2code.springdemo.controller.*.*(..))")
    private void forControllerPackage() {}

    //do the same thing for service and dao package
    @Pointcut("execution(* com.luv2code.springdemo.service.*.*(..))")
    private void forServicePackage() {}

    @Pointcut("execution(* com.luv2code.springdemo.dao.*.*(..))")
    private void fordaoPackage() {}

    @Pointcut("forControllerPackage() || forServicePackage() || fordaoPackage()")
    private void forAppFlow() {}

    //add @Before advice
    @Before("forAppFlow()")
    private void before(JoinPoint theJoinpoint) {

            //display method we are calling
            String theMethod = theJoinpoint.getSignature().toShortString();
            myLogger.info("===>> in @before: calling method: "+theMethod);

            //display the argumeents to the method
            Object[] args = theJoinpoint.getArgs();

            //loop thru and display args
            for(Object tempArgs : args) {
                    myLogger.info("===>> argument: "+tempArgs);
            }

    }


    //add @AfterReturning Advice
    @AfterReturning(
                    pointcut = "forAppFlow()",
            returning = "theResult"
                    )
    public void afterRetruning(JoinPoint theJoinpoint, Object theResult) {

            //display the method we are returning from
            String theMethod = theJoinpoint.getSignature().toShortString();
            myLogger.info("===>> in @AfterReturning: calling method: "+theMethod);

            //display the data returned
            myLogger.info("====>>> result: "+theResult);

    }

}
```

Going to update page in the console, we can see the id that is being passed as an argument and it is getting displayed onto the console through controller, service and dao package.



```
Jul 08, 2017 4:02:35 PM org.hib
INFO: HHH000397: Using ASTQuery
Hibernate: select customer0_.id as id1_0_, customer0_.email as email2_0_, customer0_.first_
Jul 08, 2017 4:03:12 PM com.luv2code.springdemo.aspect.CRMLoggingAspect before
INFO: =====>> in @Before: calling method: CustomerController.showFormForUpdate(..)
Jul 08, 2017 4:03:12 PM com.luv2code.springdemo.aspect.CRMLoggingAspect before
INFO: =====>> argument: 4
Jul 08, 2017 4:03:12 PM com.luv2code.springdemo.aspect.CRMLoggingAspect before
INFO: =====>> argument: {}
Jul 08, 2017 4:03:12 PM com.luv2code.springdemo.aspect.CRMLoggingAspect be
INFO: =====>> in @Before: calling method: CustomerService.getCustomer(..)
Jul 08, 2017 4:03:12 PM com.luv2code.springdemo.aspect.CRMLoggingAspect b
INFO: =====>> argument: 4
Jul 08, 2017 4:03:12 PM com.luv2code.springdemo.aspect.CRMLoggingAspect b   ore
INFO: =====>> in @Before: calling method: CustomerDAO.getCustomer(..)
Jul 08, 2017 4:03:12 PM com.luv2code.springdemo.aspect.CRMLoggingAspect before
INFO: =====>> argument: 4
Hibernate: select customer0_.id as id1_0_0_, customer0_email as email2_0_0_, customer0_.fi
```

For @AfterReturning output:



```
                                                              tCustomers()
                                                              ggingAspect before
                                                              tomers()
Jul 08, 2017 4:11:08 PM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator initiate
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select customer0_.id as id1_0_, customer0_.email as email2_0_, customer0_.first_
Jul 08, 2017 4:11:08 PM com.luv2code.springdemo.aspect.CRMLoggingAspect afterReturning
INFO: =====>> in @AfterReturning: from method: CustomerDAO.getCustome
Jul 08, 2017 4:11             uv2code.springdemo.aspect.CRMLoggingAs
INFO: =====>>                  id=1, firstName=David, lastName=Ad                        cod
Jul 08, 2017                  de.springdemo.aspect.CRMLoggingAsp
INFO: =====>>                  from method: CustomerService.getCusto
Jul 08, 2017 4:1              2code.springdemo.aspect.CRMLoggingAspect  fterReturning
INFO: =====>> result: [C   omer [id=1, firstName=David, lastName=Adams, email=david@luv2cod
Jul 08, 2017 4:11:08 PM c  .luv2code.springdemo.aspect.CRMLoggingAspect afterReturning
INFO: =====>> in @Af                   method: CustomerController.listCustomers(..)
Jul 08, 2017 4:11:08 PM com luv2code. pringdemo.aspect.CRMLoggingAspect afterReturning
INFO: =====>> result: list-customers
```

We can see here that the controller returning a view page after getting executed; and @AfterReturning intercept the return value on the console logs.