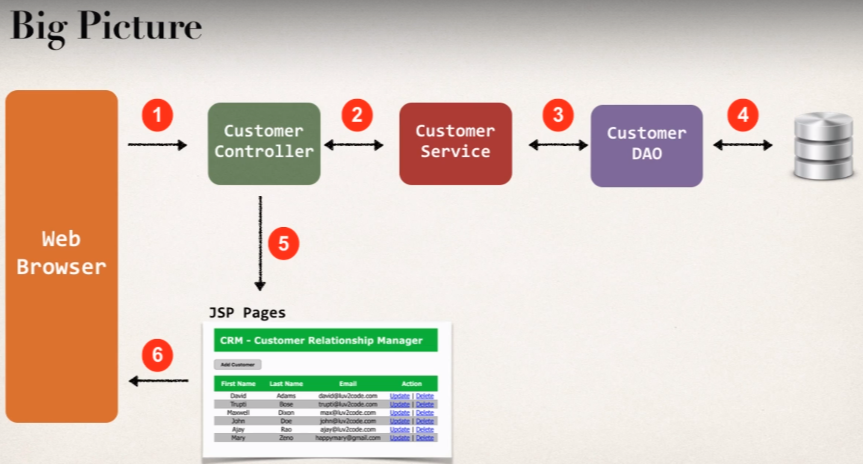
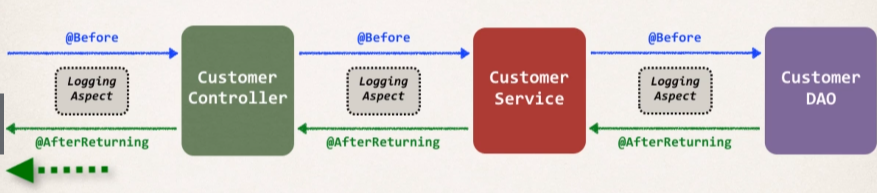


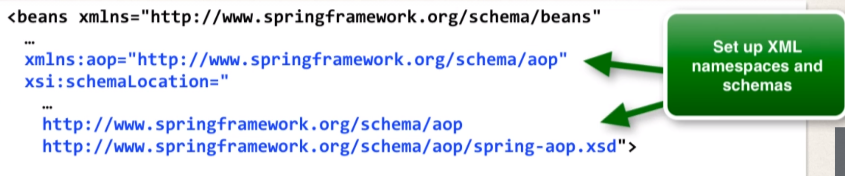
* **Add AOP logging support to our Spring MVC app (CRM Web app)**

**Logging Aspect**

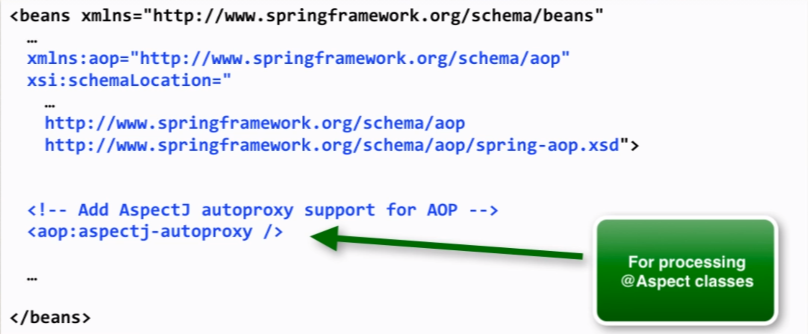
**Development Process**

1. Add AspectJ Jar File to web project

Add jar file to your web project: **WEB-INF/lib** directory

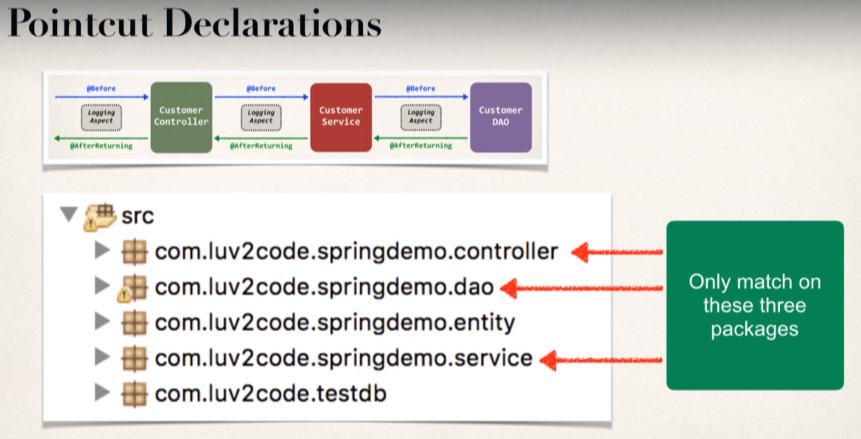
1. Enable AspectJ Auto Proxy

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.



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.

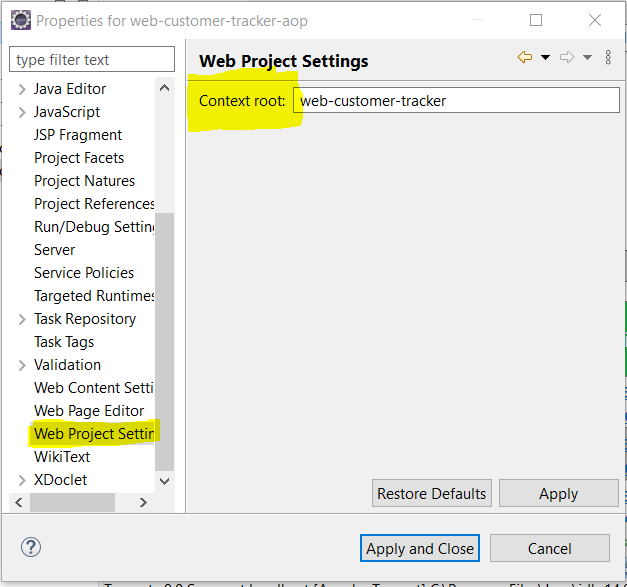
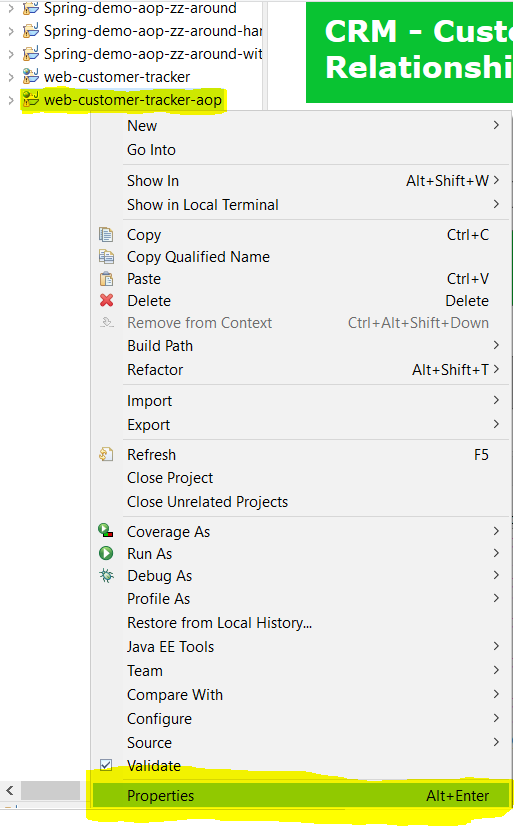
1. Create Aspect

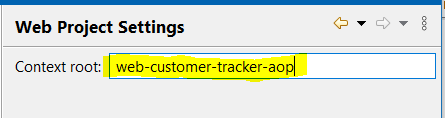
* Adding logging support
* Setup pointcut declaration
* Add @Before advice
* Add @AfterReturning advice
* We will create individual declarations for all these packages and we will combine them together using the techniques that we have learnt in the Spring AOP and then apply for our logging services in Spring MVC web App.
* **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 **web-customer-tracker-aop**

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



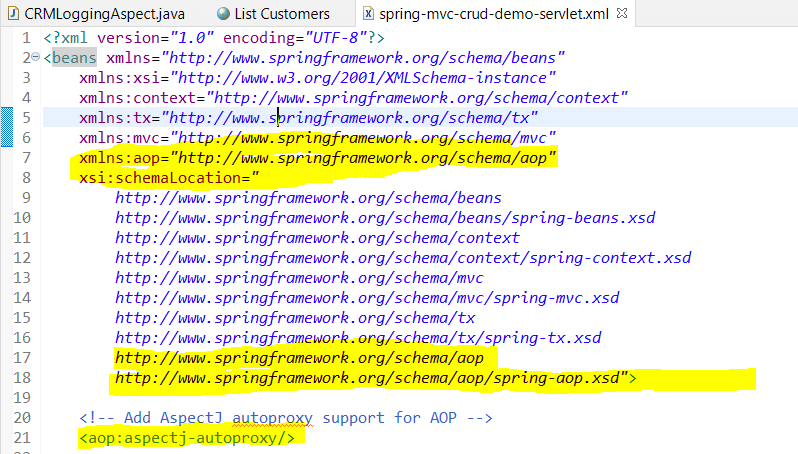


Now in the link to get the index page of our web app; we need to provide

<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](https://github.com/000GEV744/Spring-Git/blob/main/web-customer-tracker-aop/src/main/webapp/WEB-INF/spring-mvc-crud-demo-servlet.xml)

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](https://github.com/000GEV744/Spring-Git/blob/main/web-customer-tracker-aop/src/main/java/com/luv2code/springdemo/aspect/CRMLoggingAspect.java)

Here we have defined the advices for the controller, service, and dao packages only.

@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();

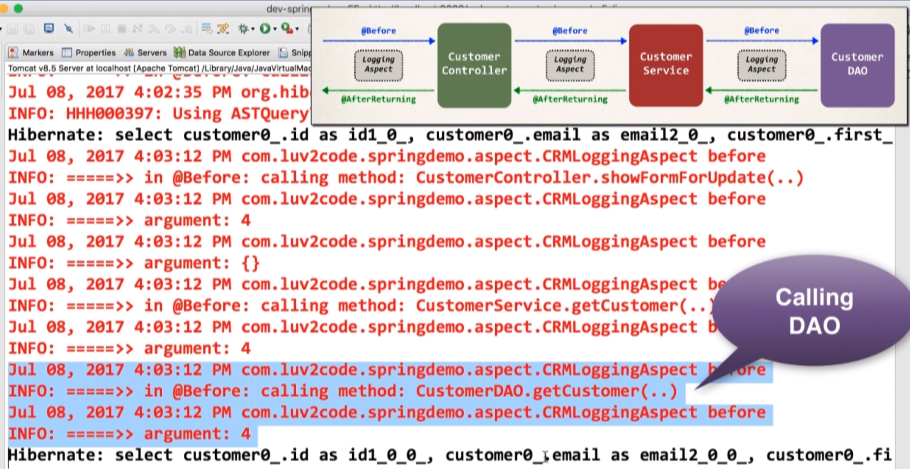
myLogger.info("===>> in @AfterReturning: calling method: "+theMethod);

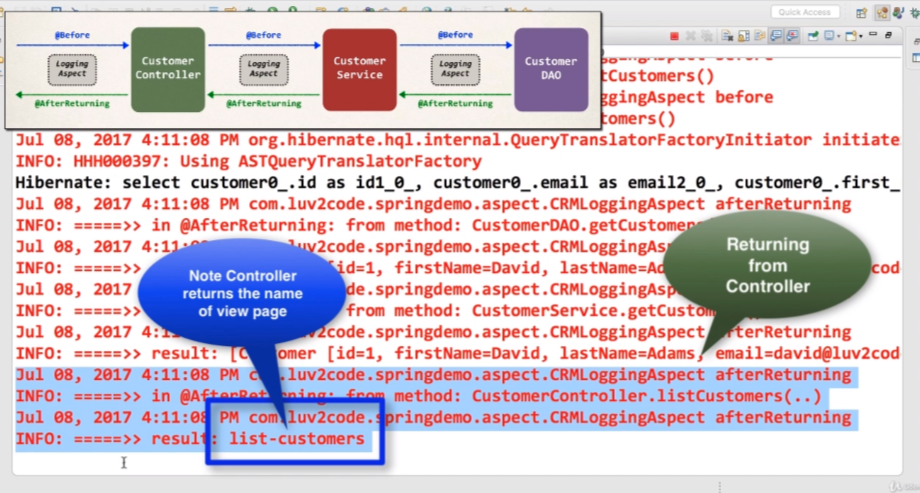
//display the data returned

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.

For @AfterReturning output:

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