CourseCube®
(Formerly Java Learning Center)

Spring 5.2
AOP

Author
Srinivas Dande

Java Learning Center

Spring AOP

- **AOP – Aspect Oriented Programming**
- When you are developing any Enterprise Application, you need the following services commonly:
    1) Low Level Services
    2) Middle Level Services
    3) High Level Services

1) Low Level Services:

- Some of the Low Level Services are IO, Threading, Networking etc which will be supplied by servers freely.

2) Middle Level Services:

- Some of the Middle Level Services are Transactions, Security, Logging, Messaging etc which has to be implemented by you.

3) High Level Services:

- High Level service of Enterprise Application is nothing but Business Logic which you are writing for Business Operations.

- Normally, when you implement Business Operation you need to write the code for Business Logic and Middle Level Services.

Consider the following requirement:

```
class AccountService {
  Logger log=Logger.getRootLogger(...);

        public void deposit(...){

         if(x.isCallerInRole("Teller")){
                try{
                    log.info(.....);
                    tx.begin();
                         OP1;
                         OP2;
                    tx.commit();
                    log.info(...);
                }catch(Exception e){
                    tx.rollback();
                    log.info(...);
                }
        }else{
                throw Some SecurityException;
        }

   }

}
```

- **In the above code,**
  - Core Business logic is mixed with middle level services like Transaction, Security and Logging.
  - When you want to change the existing Transactions API or Security API or Logging API with new one then you need to re-modify the entire application.
  - This may give maintenance problem.

Spring AOP

- AOP stands for Aspect Oriented Programming.

- AOP is new kind of Programming technique which is mainly used to separate the commonly required middle level services logic from core business logic of the application.

- Transactions, Security, Logging and Messaging etc are the middle level services which are also called as cross-cutting concerns.

With AOP

Busness Services	Middle Level Services
class AccountService {	class TxService {
public void deposit(...){	void begin(){ ... }
OP1;	void commit(){ ... }
OP2;	void rollback(){ ... }
OP3;	}
}	
...	class SecurityService {
}	public void verifyUser(){
	...
class CustomerService {	}
public void addCustomer(...){	}
OP1;	
OP2;	class LogService {
}	void log(..){
...	...
}	}
	}

- **In the above code,**
  - Core Business logic is completely separated from middle level services.
  - Now when you want to change the existing Transactions, Security or Logging implementations with new one then that will not impact the business services.

Following are various AOP Frameworks available

1) Spring AOP
2) AspectJ
3) JBoss AOP

Spring AOP Terminology

1) **Aspect**
2) **Advice**
3) **JoinPoint**
4) **PointCut**
5) **Advisor**
6) **Target**
7) **Proxy**
8) **Weaving**

1) Aspect

- Commonly Required Middle Level services which you are implementing for your Enterprise application are called as Aspects.
- Security, Transactions, Logging etc are aspects.

2) Advice

- Implementation of a Middle Level Service is called as Advice.
- Implementation of an Aspect is called as Advice.
- i.e. Advice is a class which contains Code for Aspects like Security, Txs,logging etc.

Ex:

```
class TxService { ... }
class SecurityService { ... }
class LogService { ... }
```

3) JoinPoint

- JoinPoint is a point in the Program execution where you want to apply advices.
- JoinPoint is a point in the Program execution where you want to run middle level services code.

Ex:

```
try{
        txs.begin();      Before Business Operation
        as.deposit();     Business Operation
        txs.commit();     After Business Operation returns the Control Successfully
}catch(Exception e){
        txs.rollback();   After Business Operation throws Exception
}
```

- **Spring AOP supports the following JoinPoints**

1) MethodBefore	Before invoking method
2) MethodReturing	When method returns the control successfully.
3) MethodThrowing	When method throws an Exception
4) MethodAfter	When method returns control any way.
5) MethodAround	Before and After method Invocation.

4) PointCut

- Collection of JoinPoints is called as PointCut.
- By default, Advices will be applied for all the business operations of all the Business Services.
- When you want apply the advices for some specified business operations of specified Business Services then you must define point-cut with the required AspectJ Expression.

Ex1:

execution(* com.jlcindia.*Service.*(..))

Advice will be applied for
getBal() of AccountService
mydeposit() of AccountService
mywithdraw()of AccountService
addCustomer()of CustomerService
updateCustomer()of CustomerService

Ex2:
execution(* com.jlcindia.AccountService.my*(..))

Advice will be applied for
mydeposit() of AccountService
mywithdraw() of AccountService

Ex3:
execution(* com.jlcindia.CustomerService.update*(..))

Advice will be applied for
updateCustomer() of CustomerService

Define the PointCut Using AspectJ Expression

Syntax

execution(modifierPattern returnTypePattern?
businessServicePattern.methodPattern(ParamsPattern) throws ExceptionPattern?)

5) Advisor

- Advisor is combination of Advice and PointCut.

6) Target

- Target is an Object of your business service before applying the Advices or Advisors.

7) Proxy

- Proxy is an Object of your business service after applying the Advices or Advisors.

8) Weaving

- Weaving is the process of applying the Advices or Advisors to the Target objects at given pointcuts to get the Proxy objects.

Using Annotation based AOP

1) **@Aspect**
2) **@PointCut**
3) **@Before**
4) **@AfterReturning**
5) **@AfterThrowing**
6) **@After**
7) **@Around**

Lab49: Working Steps:

A) Mark your Configuration class with @EnableAspectJAutoProxy

```
@Configuration
@EnableAspectJAutoProxy
public class JLCConfig {
        ....
        ...
}
```

B) Write your Business Service.

- AccountService.java

C) Configure AccountService bean in the configuration class.

```
@Bean(name="myas")
public AccountService accountService(){
return new AccountService();
}
```

D) Write your Middle Level Service called Tx Service.

- TxService.java

E) Configure TxService bean in the configuration class.

```
@Bean
public TxService txService(){
return new TxService();
}
```

F) Apply the required Aspects for your TxService.

1) Mark the TxService with **@Aspect**.

2) Write the special method and mark that with **@Pointcut**

3) Specify the AspectJ Pointcut Expression with **@Pointcut** annotation.

4) Use **@Before** annotation for the method which you want invoke before invoking the Business Operation.

   Ex:
   ```
   @Before("myjlc()")
   public void begin() {
   System.out.println("TS- begin");
   }
   ```

5) Use **@AfterReturning** annotation for the method which you want invoke after returning the control from the Business Operation successfully.

   Ex:
   ```
    @AfterReturning("myjlc()")
   public void commit() {
   System.out.println("TS- commit");
   }
   ```

6) Use **@AfterThrowing** annotation for the method which you want invokes after Business Operation throws the Exception.

   Ex:
   ```
   @AfterThrowing("myjlc()")
   public void rollback() {
           System.out.println("TS- rollback");
   }
   ```

Example using Annotation Based AOP

Lab50:

- Annotation based AOP with **Autoproxying**
- **@Before**, **@AfterReturning**, **@AfterThrowing**,
- **AspectJ Pointcuts Expressions** based on annotations with **@PointCut**
- **One Busniess Service** and **One Middle Level Service**

Lab50: Files required

1. Lab50.java	**2. AccountService.java**
3. TxService.java	**4. InsufficientFundsException.java**
5. JLCAppConfig.java	

1. Lab50.java

```
package com.coursecube.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
```

```
* @Website : www.coursecube.com
* */
public class Lab50 {
public static void main(String[] args) {
ApplicationContext ctx = new AnnotationConfigApplicationContext(JLCAppConfig.class);

AccountService as = (AccountService) ctx.getBean("myas");
as.mydeposit();
System.out.println("=============== ");
as.getBal();
System.out.println("=============== ");
try {
as.mywithdraw();
} catch (Exception e) {
System.out.println("Sorry --- ");
}
}
}
```

2. AccountService.java

```
package com.coursecube.spring;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
public class AccountService {
        public void getBal() {
                System.out.println("getBal- begin");
                System.out.println("getBal- done");
                System.out.println("getBal- end");
        }

        public void mydeposit() {
                System.out.println("deposit- begin");
                System.out.println("deposit- done");
                System.out.println("deposit- end");
        }

        public void mywithdraw() throws Exception {
                System.out.println("withdraw- begin");
                if (1 == 1) {
                        throw new InsufficientFundsException();
                }
                System.out.println("withdraw- end");
        }
}
```

3. TxService.java

```
package com.coursecube.spring;

import org.aspectj.lang.annotation.*;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
@Aspect
public class TxService {

//@Pointcut(value = "execution(* com.coursecube.spring.AccountService.my*(..))")
	@Pointcut(value = "execution(* com.coursecube.spring.AccountService.*(..))")
	public void jlc() {
	}

	@Before("jlc()")
	public void begin() {
		System.out.println("** TS - begin");
	}

	@AfterReturning("jlc()")
	public void commit() {
		System.out.println("** TS - commit");
	}

	@AfterThrowing("jlc()")
	public void rollback() {
		System.out.println("** TS - rollback");
	}
}
```

4. InsufficientFundsException.java

```
package com.coursecube.spring;

/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */

public class InsufficientFundsException extends Exception {

}
```

5. JLCAppConfig.java

```
package com.coursecube.spring;

import org.springframework.context.annotation.*;
import org.springframework.context.annotation.EnableAspectJAutoProxy;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */

@Configuration
@EnableAspectJAutoProxy
public class JLCAppConfig {

	@Bean
	public TxService txService() {
		return new TxService();
	}

	@Bean(name = "myas")
	public AccountService accountService() {
		return new AccountService();
	}
}
```

What is happening with Lab50?

At Container Start-up

1) Spring Container scans and prepares the advice classes which are marked with @Aspect annotation.
2) Spring Container scans and prepares the Point-Cut Expressions specified inside the advice classes.
3) Spring Container scans and prepares the methods which are referring Point-Cut Specific special Method.
4) Spring Container scans and prepares the methods which are marked with Join-point related Annotations (@Before, @AfterReturning....)

When you invoke any business operation

1) Takes the method invoked by you and check whether that method is matching with any of Point-Cut Expressions specified.
2) When method is not matching with any of Point-Cut Expressions given then that method will be called directly without applying any advices.
3) When method is matching with any of Point-Cut Expressions given then specified advices will be applied for that method.

Example using Annotation Based AOP

Lab51:

- Annotation based AOP with **Autoproxying**
- **@Around, @AfterThrowing,**
- **AspectJ Pointcuts Expressions** based on annotations with **@PointCut**
- **One Busniess Service** and **One Middle Level Service**

Lab51: Files required

1. **Lab51.java**	Same as Lab50
2. **AccountService.java**	Same as Lab50
3. **TxService.java**	**Updated in Lab51**
4. **InsufficientFundsException.java**	Same as Lab50
5. **JLCAppConfig.java**	Same as Lab50

3. TxService.java

```
package com.coursecube.spring;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.*;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
@Aspect
public class TxService {
        @Pointcut(value = "execution(* com.coursecube.spring.AccountService.my*(..))")
        public void jlc() {
        }

        @Around("jlc()")
        public void runTx(ProceedingJoinPoint pjp) throws Throwable {
                System.out.println("** TS - begin");
                pjp.proceed();
                System.out.println("** TS - commit");
        }

        @AfterThrowing("jlc()")
        public void rollback() {
                System.out.println("** TS - rollback");
        }
}
```

Example using Annotation Based AOP

Lab52:

- Annotation based AOP with **Autoproxying**
- **@Before, @AfterReturning, @AfterThrowing, @After**
- **AspectJ Pointcuts Expressions** based on annotations with **@PointCut**
- **Multiple Busniess Services** and **Multiple Middle Level Service**

Lab52: Files required

1. Lab52.java	**2. AccountService.java**
3. CustomerService.java	**4. SecurityService.java**
5. TxService.java	**6. LogService.java**
7. InsufficientFundsException.java	**8. JLCAppConfig.java**

1. Lab52.java

```
package com.coursecube.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
public class Lab52 {
public static void main(String[] args) {
ApplicationContext ctx = new AnnotationConfigApplicationContext(JLCAppConfig.class);

CustomerService cs = (CustomerService) ctx.getBean("mycs");
cs.getCustomer();
System.out.println("=============== ");

cs.addCustomer();
System.out.println("=============== ");

AccountService as = (AccountService) ctx.getBean("myas");
as.mydeposit();
System.out.println("=============== ");

as.getBal();
System.out.println("=============== ");
try {
as.mywithdraw();
} catch (Exception e) {
System.out.println("Sorry --- ");
}
}
}
```

2. AccountService.java

```
package com.coursecube.spring;

import org.springframework.stereotype.Component;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
@Component("myas")
public class AccountService {
	public void getBal() {
		System.out.println("getBal- begin");
		System.out.println("getBal- done");
		System.out.println("getBal- end");
	}

	public void mydeposit() {
		System.out.println("deposit- begin");
		System.out.println("deposit- done");
		System.out.println("deposit- end");
	}

	public void mywithdraw() throws Exception {
		System.out.println("withdraw- begin");
		if (1 == 1) {
			throw new InsufficientFundsException();
		}
		System.out.println("withdraw- end");
	}
}
```

3. CustomerService.java

```
package com.coursecube.spring;

import org.springframework.stereotype.Component;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
@Component("mycs")
public class CustomerService {
	public void getCustomer() {
		System.out.println("getCustomer - begin");
		System.out.println("getCustomer - done");
		System.out.println("getCustomer - end");
	}
	public void addCustomer() {
		System.out.println("addCustomer - begin");
		System.out.println("addCustomer - done");
		System.out.println("addCustomer - end");
	}
}
```

4. SecurityService.java

```
package com.coursecube.spring;

import org.aspectj.lang.annotation.*;
import org.springframework.stereotype.Component;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
@Aspect
@Component
public class SecurityService {

	@Pointcut(value = "execution(* com.coursecube.spring.*Service.*(..))")
	public void jlc() {
	}

	@Before("jlc()")
	public void verifyUser() {
		System.out.println("** SS - verifyUser");
	}
}
```

5. TxService.java

```
package com.coursecube.spring;

import org.aspectj.lang.annotation.*;
import org.springframework.stereotype.Component;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
@Aspect
@Component
public class TxService {
	@Pointcut(value = "execution(* com.coursecube.spring.AccountService.my*(..))")
	public void jlc1() {
	}
```

```
        @Pointcut(value = "execution(* com.coursecube.spring.CustomerService.add*(..))")
        public void jlc2() {
        }

        @Before("jlc1() || jlc2()")
        public void begin() {
                System.out.println("** TS - begin");
        }

        @AfterReturning("jlc1() || jlc2()")
        public void commit() {
                System.out.println("** TS - commit");
        }

        @AfterThrowing("jlc1() || jlc2()")
        public void rollback() {
                System.out.println("** TS - rollback");
        }
}
```

6. LogService.java

```
package com.coursecube.spring;

import org.aspectj.lang.annotation.*;
import org.springframework.stereotype.Component;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
@Aspect
@Component
public class LogService {
        @Pointcut(value = "execution(* com.coursecube.spring.AccountService.my*(..))")
        public void jlc1() {
        }

        @Pointcut(value = "execution(* com.coursecube.spring.CustomerService.add*(..))")
        public void jlc2() {
        }
        @Before("jlc1() or jlc2()")
        public void logBefore() {
                System.out.println("** LS- logBefore");
        }
        @AfterReturning("jlc1() or jlc2()")
        public void logReturning() {
                System.out.println("** LS- logReturning");
        }
        @AfterThrowing("jlc1() or jlc2()")
        public void logThrowing() {
                System.out.println("** LS- logThrowing");
        }

        @After("jlc1() or jlc2()")
        public void logOk() {
                System.out.println("** LS...logOk.()..");
        }
}
```

7. InsufficientFundsException.java

```
package com.coursecube.spring;

/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */

public class InsufficientFundsException extends Exception {

}
```

8. JLCAppConfig.java

```
package com.coursecube.spring;

import org.springframework.context.annotation.*;
import org.springframework.context.annotation.EnableAspectJAutoProxy;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
@Configuration
@EnableAspectJAutoProxy
@ComponentScan(basePackages = {"com.coursecube.spring"})
public class JLCAppConfig {


}
```

Example using Annotation Based AOP

Lab53:

- Annotation based AOP with **Autoproxying**
- **@Around, @AfterThrowing, @After**
- **AspectJ Pointcuts Expressions** based on annotations with **@PointCut**
- **Multiple Busniess Services** and **Multiple Middle Level Service**

Lab53: Files required

1. Lab53.java	Same as Lab52
2. AccountService.java	Same as Lab52
3. CustomerService.java	Same as Lab52
4. SecurityService.java	**Updated in Lab53**
5. TxService.java	**Updated in Lab53**
6. LogService.java	**Updated in Lab53**
7. InsufficientFundsException.java	Same as Lab52
8. JLCAppConfig.java	Same as Lab52

4. SecurityService.java

```
package com.coursecube.spring;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.stereotype.Component;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
@Aspect
@Component
public class SecurityService {
        @Pointcut(value = "execution(* com.coursecube.spring.*Service.*(..))")
        public void jlc() {
        }

        @Around("jlc()")
        public void verifyUser(ProceedingJoinPoint pjp) throws Throwable {
                System.out.println("** verifyUser begin..");
                pjp.proceed();
                System.out.println("** verifyUser End..");
        }
}
```

5. TxService.java

```
package com.coursecube.spring;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.stereotype.Component;

/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */

@Aspect
@Component
public class TxService {

	@Pointcut(value = "execution(* com.coursecube.spring.AccountService.my*(..))")
	public void jlc1() {
	}

	@Pointcut(value = "execution(* com.coursecube.spring.CustomerService.add*(..))")
	public void jlc2() {
	}


	@Around("jlc1() || jlc2()")
	public void runTx(ProceedingJoinPoint pjp) throws Throwable {
		System.out.println("** TS - begin");
		pjp.proceed();
		System.out.println("** TS - commit");
	}

	@AfterThrowing("jlc1() || jlc2()")
	public void rollback() {
		System.out.println("** TS - rollback");
	}
}
```

6. LogService.java

```
package com.coursecube.spring;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.stereotype.Component;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */

@Aspect
@Component
public class LogService {
	@Pointcut(value = "execution(* com.coursecube.spring.AccountService.my*(..))")
	public void jlc1() {
	}

	@Pointcut(value = "execution(* com.coursecube.spring.CustomerService.add*(..))")
	public void jlc2() {
	}

	@Around("jlc1() or jlc2()")
	public void log(ProceedingJoinPoint pjp) throws Throwable {
		System.out.println("** LS- logBefore");
		pjp.proceed();
		System.out.println("** LS- logReturning");
	}

	@AfterThrowing("jlc1() or jlc2()")
	public void logThrowing() {
		System.out.println("** LS- logThrowing");
	}

	@After("jlc1() or jlc2()")
	public void logOk() {
		System.out.println("** LS...logOk.()..");
	}
}
```