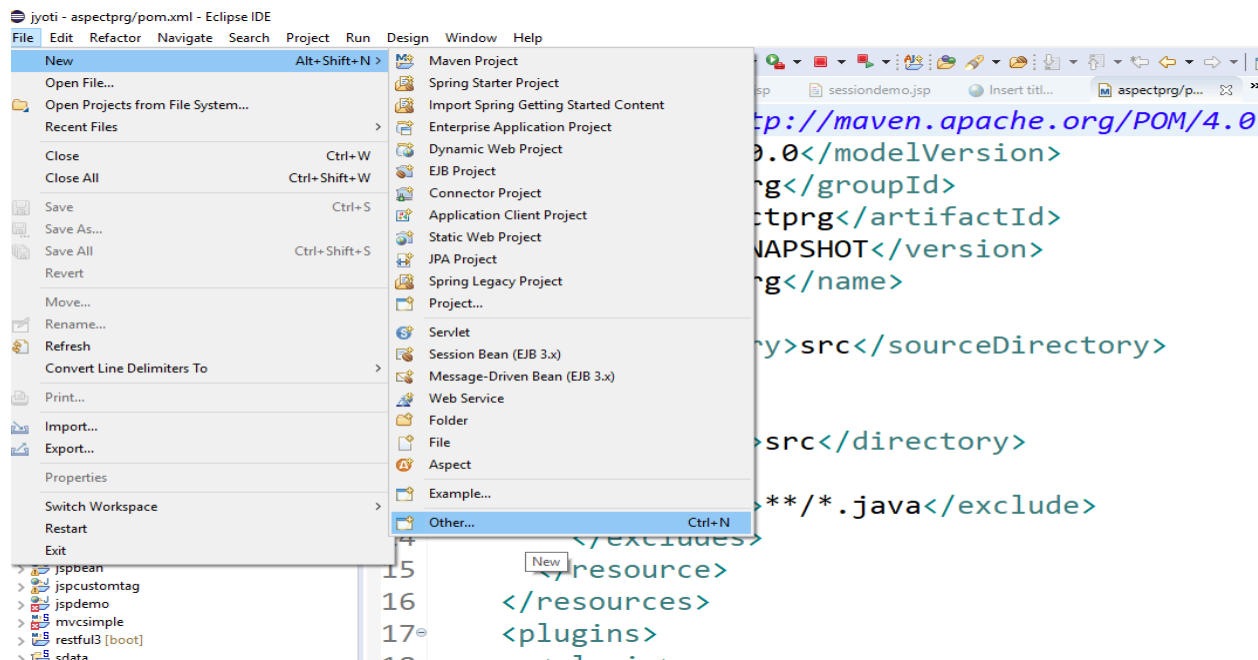
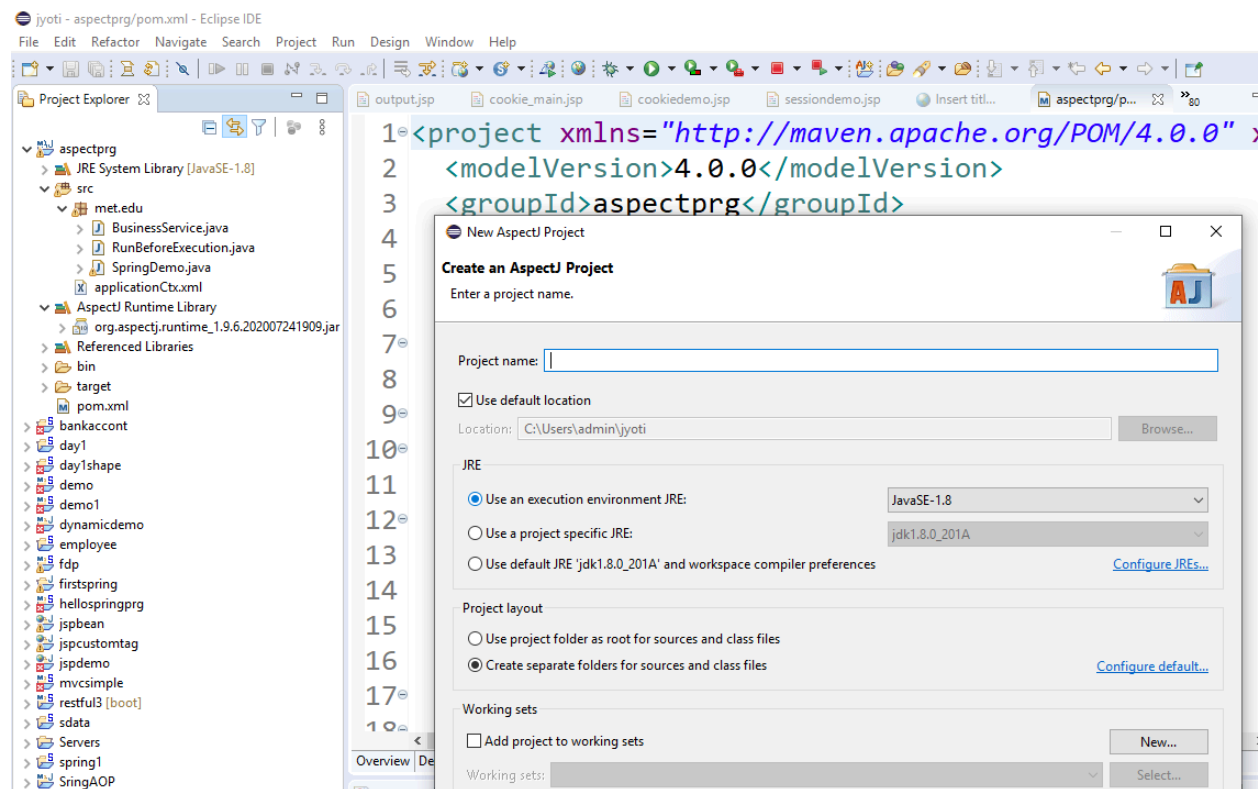


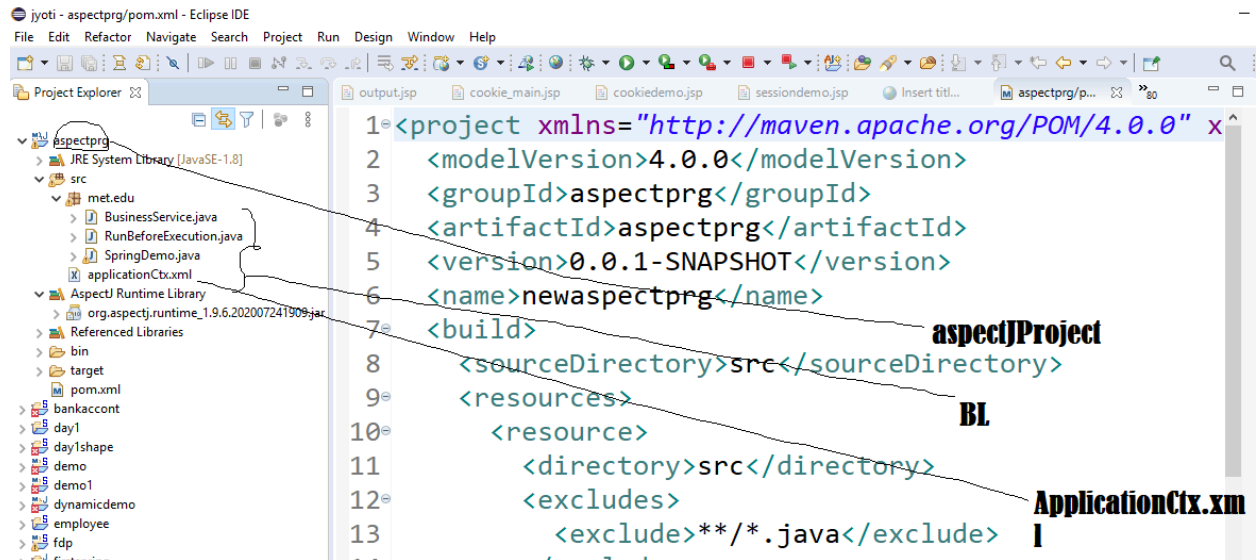
Spring and AOP : Aspect Oriented Programming with Spring

1. Select New-> other->aspectJProject



Give name to the project as aspectprg





Right click on src -> new package as met.edu

File : BusinessService.java

package met.edu;

public class BusinessService {

public void runMyBusinessLogic() {

System.out.println("***");**

System.out.println("Running business logic...");

System.out.println("***");**

}

public void testThrowException() {

throw new NullPointerException();

}

}

File: RunBeforeExecution.java

```
package met.edu;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Aspect
public class RunBeforeExecution {

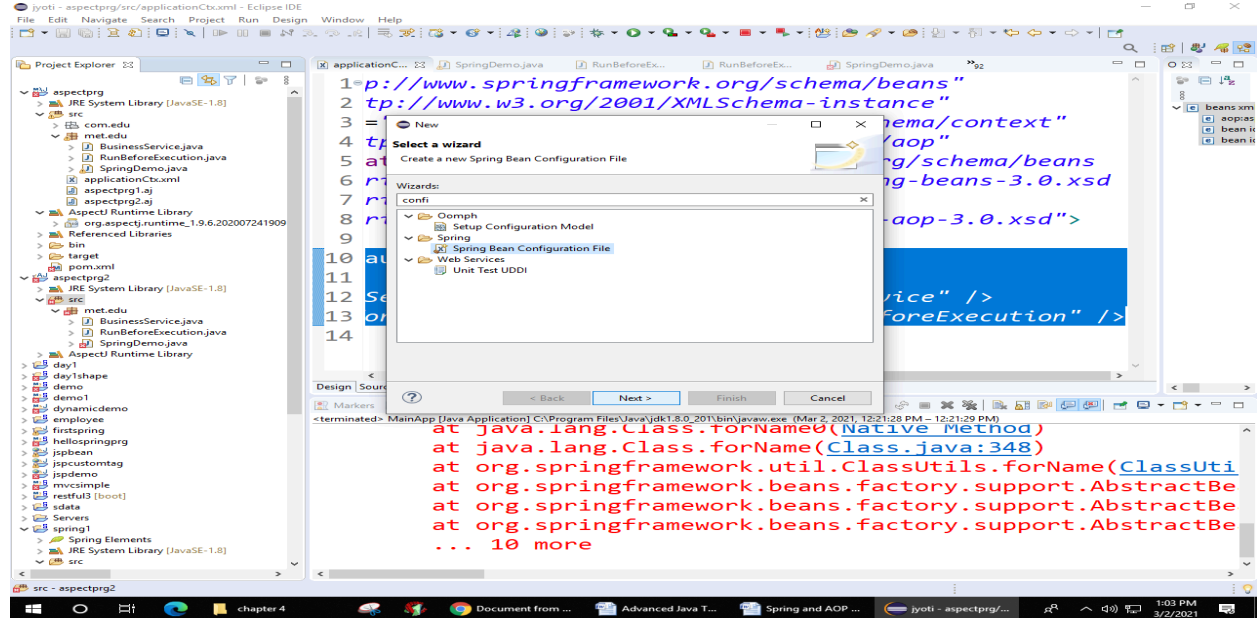
    @Before("execution(* met.edu.BusinessService.runMyBusinessLogic(..)")
    public void before(JoinPoint joinPoint) throws Throwable {
        System.out.println("Inside RunBeforeExecution.before() method...");
        System.out.println("inserted before : " + joinPoint.getSignature().getName());
    }
}
```

File: SpringDemo.java

```
package met.edu;
import java.util.*;
import org.springframework.context.ConfigurableApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class SpringDemo {
    static ConfigurableApplicationContext context;
    public static void main(String[] args) {
        String confFile = "applicationCtx.xml";

        context = new ClassPathXmlApplicationContext(confFile);
        BusinessService busServ = (BusinessService) context.getBean("busService");
        busServ.runMyBusinessLogic();
        System.out.println("Press 'Enter' to terminate");
        new Scanner(System.in).nextLine();
        System.out.println("Exiting");
        System.exit(1);
    }
}
```

Right click on src -> new applicationCtx.xml



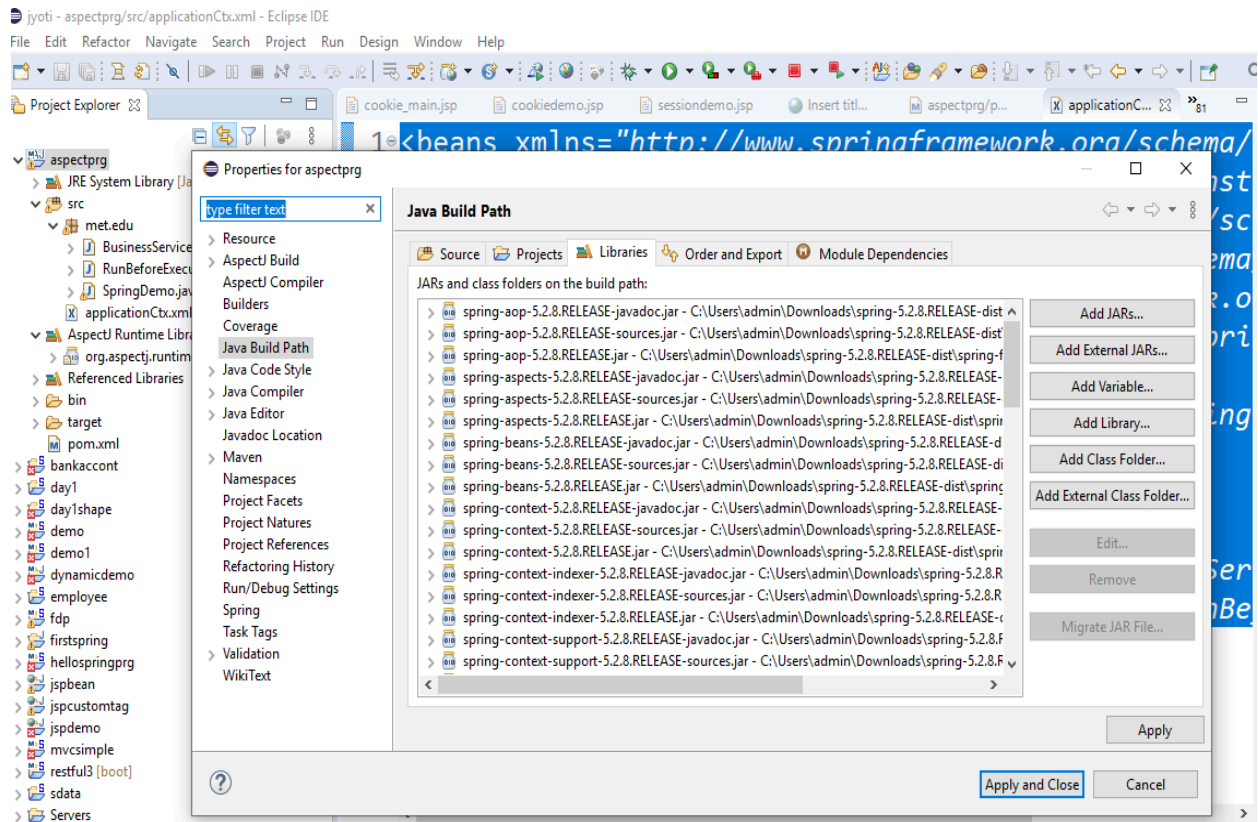
File: applicationCtx.xml

```
<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:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">

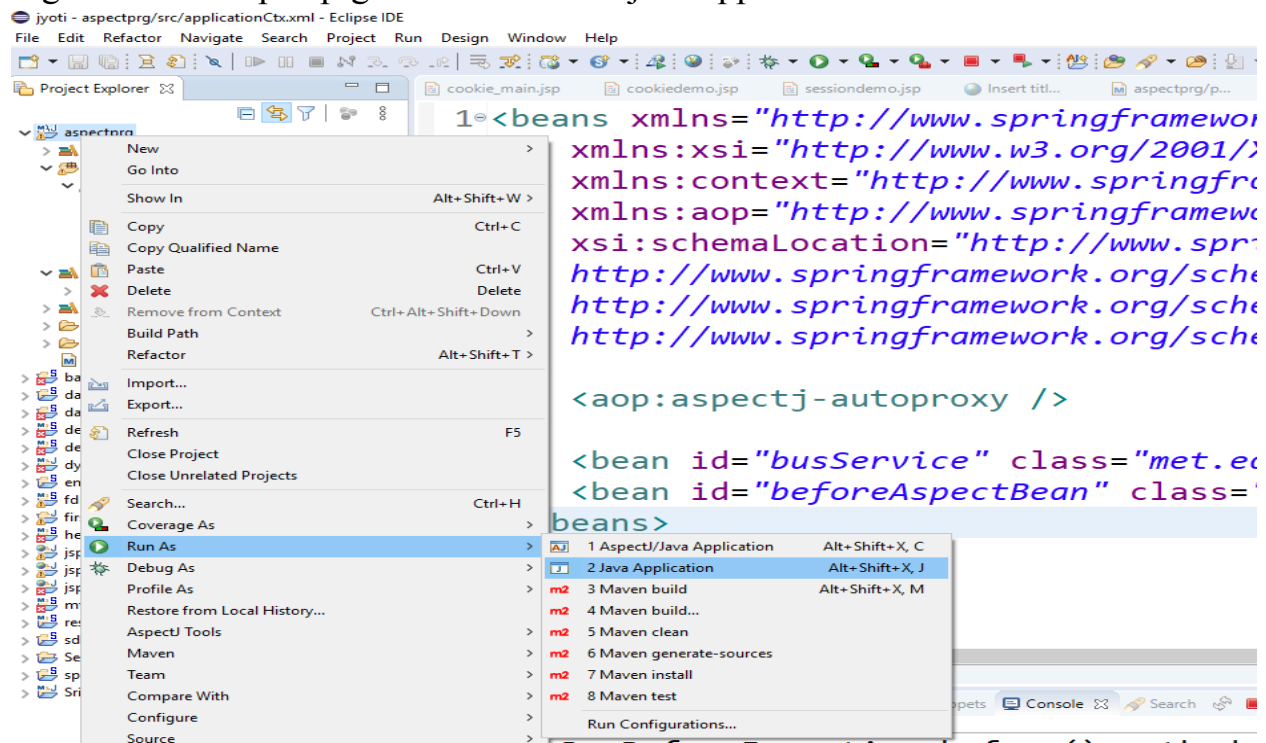
  <aop:aspectj-autoproxy />

  <bean id="busService" class="met.edu.BusinessService" />
  <bean id="beforeAspectBean" class="met.edu.RunBeforeExecution" />
</beans>
```

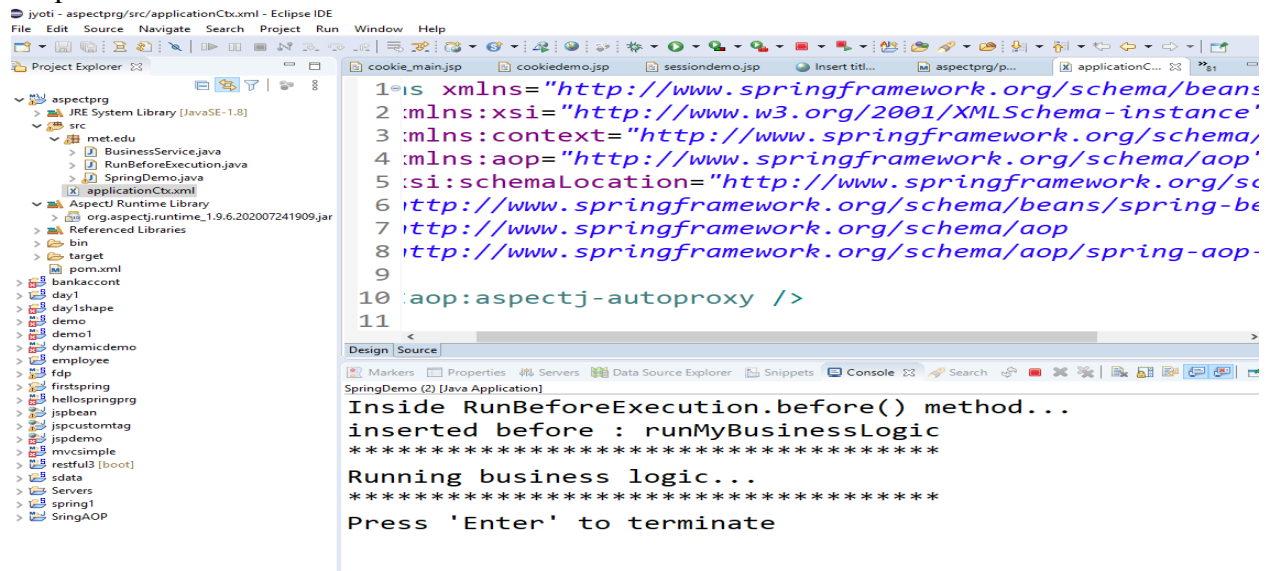
Right click on aspectprg-> buildpath->configurebuildpath
Select Libraries and add external jar files and click apply



Right click on aspectprg and click on run java application



Output



The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer displays the 'aspectprg' project structure, including 'src' (with 'BusinessService.java', 'RunBeforeExecution.java', 'SpringDemo.java', and 'applicationCtx.xml') and 'AspectJ Runtime Library'. The main editor shows 'applicationCtx.xml' with the following XML content:

```
1<?xml xmlns="http://www.springframework.org/schema/beans"
2  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3  xmlns:context="http://www.springframework.org/schema/context"
4  xmlns:aop="http://www.springframework.org/schema/aop"
5  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
6  http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd
7  http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd"
8  >
9
10 <aop:aspectj-autoproxy />
11
```

The console at the bottom shows the output of the application:

```
SpringDemo (2) [Java Application]
Inside RunBeforeExecution.before() method...
inserted before : runMyBusinessLogic
*****
Running business logic...
*****
Press 'Enter' to terminate
```

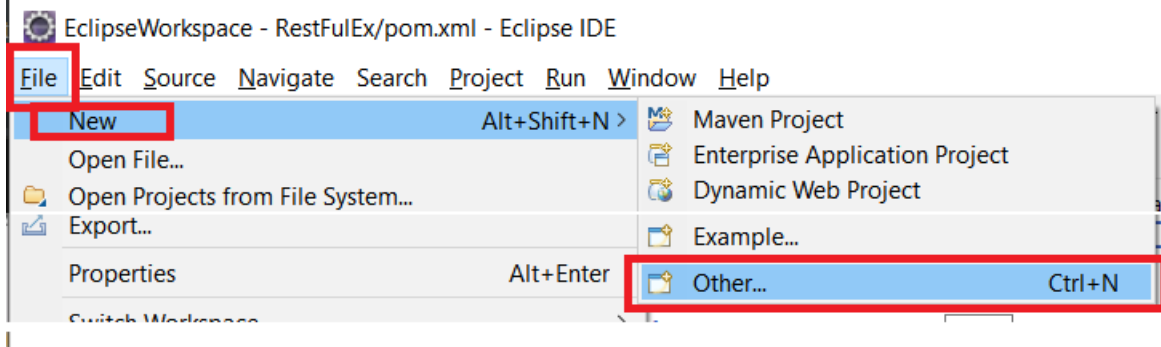
Aspect Oriented Programming

1. Write a program to demonstrate Spring AOP – before advice.
2. Write a program to demonstrate Spring AOP – after advice.
3. Write a program to demonstrate Spring AOP – around advice.
4. Write a program to demonstrate Spring AOP – after returning advice.
5. Write a program to demonstrate Spring AOP – after throwing advice.
6. Write a program to demonstrate Spring AOP – pointcuts.

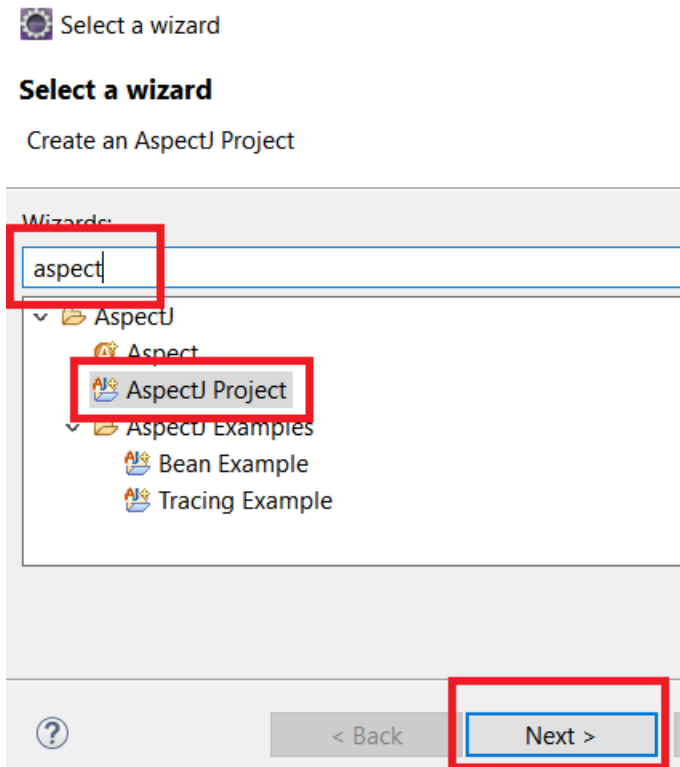
Steps to Create an AOP Project

Step 1 : Creating AspectJ Project.

1.1 : Open Eclipse. Go To File > New > Other.



1.2 : Search for 'aspect' and Select 'AspectJ Project'. Then Click on Next.



1.3 : Enter Project Name of your wish, and click on Finish.

Create an AspectJ Project

Create an AspectJ Project in the workspace or in an external location

Project name:	<input type="text" value="sample"/>	<input type="button" value="Finish"/>
---------------	-------------------------------------	---------------------------------------

1.4 : If asked to create module-info.java file, select 'Don't Create'.


Create module-info.java

Create a new module-info.java file.



Module name:	<input type="text" value="sample"/>
<input type="checkbox"/> Generate comments (configure templates and default value here)	
<div><input type="button" value="Create"/><input type="button" value="Don't Create"/></div>	

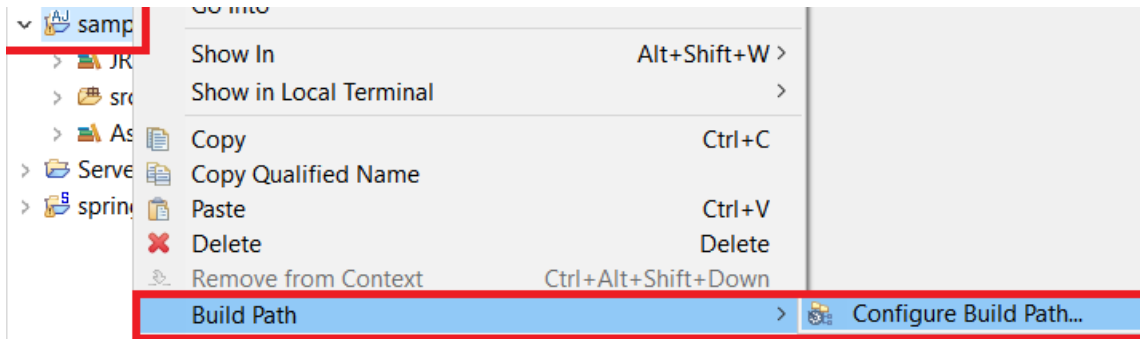
1.5 : Finally if you are asked to Open Java Perspective, just choose **NO**.

	Open the Java perspective?
This perspective is designed to support Java development. It offers a Package Explorer, a Type Hierarchy, and Java-specific navigation actions.	
<input type="checkbox"/> Remember my decision	
<div><input type="button" value="Open Perspective"/><input type="button" value="No"/></div>	

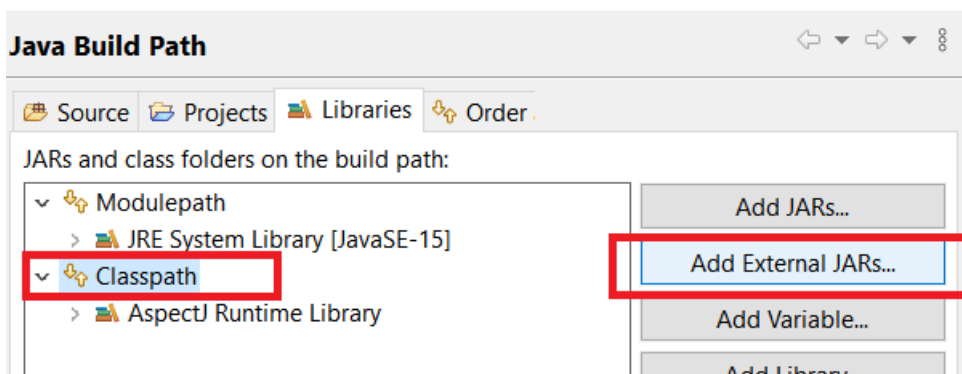
This creates your AspectJ project.

Step 2 : Adding the Spring Libraries.

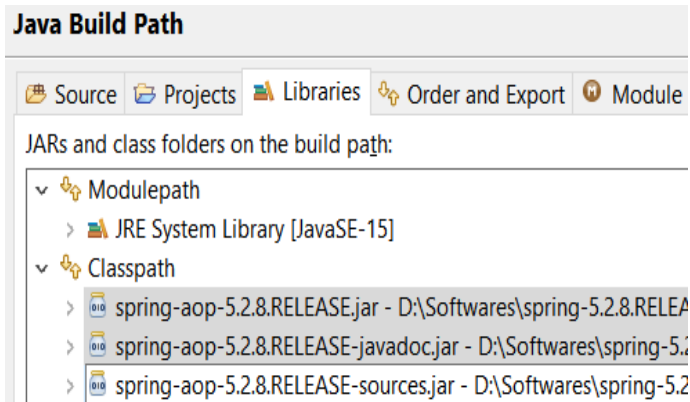
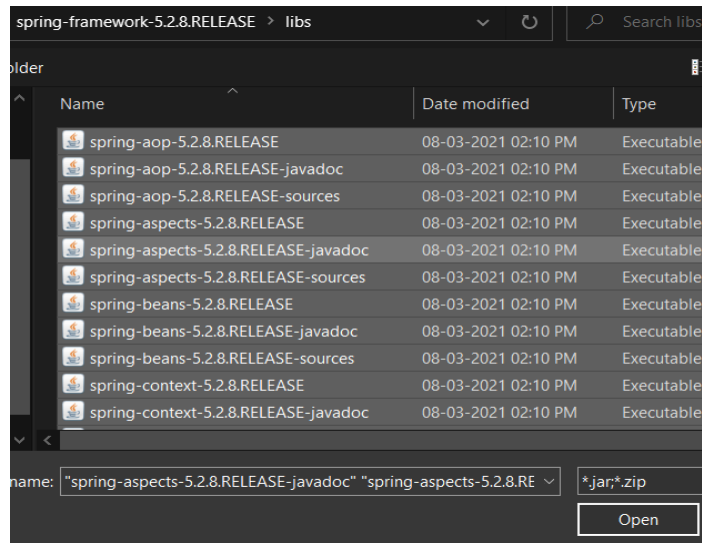
2.1 : Right click on your Newly created AspectJ project, Choose Build Path > Configure Build Path.



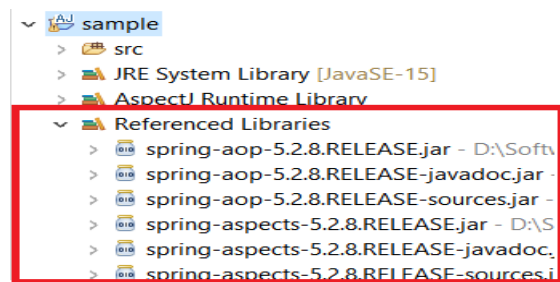
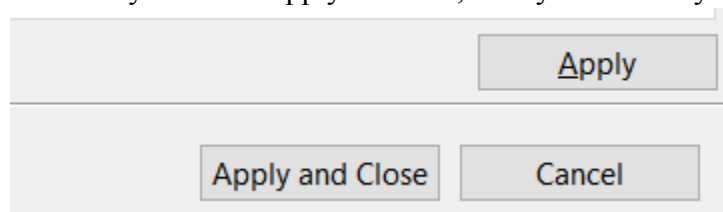
2.2 On Java Build Path wizard, Choose **Classpath** and then select **Add External JARs**.



2.3 : Choose all the Spring Libraries you've downloaded, and click on OPEN. This will add all libraries to Classpath.



2.4 Finally click on Apply & Close, now you are ready to work with Aspects in Spring.



Problem Statement 1 : Write a program to demonstrate Spring AOP – before advice.

Solution :

beforeaop.java

```
package bvimit.edu;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;

@Aspect
public class beforeaop {

    @Pointcut("execution(int beforeoperation.*(..))")
    public void p() {}

    @Before("p()")
    public void myadvice(JoinPoint jp)
    {
        System.out.println("before advice");
    }
}
```

beforeoperation.java

```
package bvimit.edu;

public class beforeoperation {
    public void msg() {System.out.println("method 1");}
    public int m(){System.out.println("method 2 with return");return 2;}
    public int k(){System.out.println("method 3 with return");return 3;}
}
```

aopctx1.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"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="opBean" class="bvimit.edu.beforeoperation"></bean>

    <bean id="trackMyBean" class="bvimit.edu.beforeaop"></bean>
```

```
<bean  
class="org.springframework.aop.aspectj.annotation.AnnotationAwareAspectJAutoProxyCreator"  
></bean>
```

```
</beans>
```

beforetest.java

```
package bvimit.edu;
```

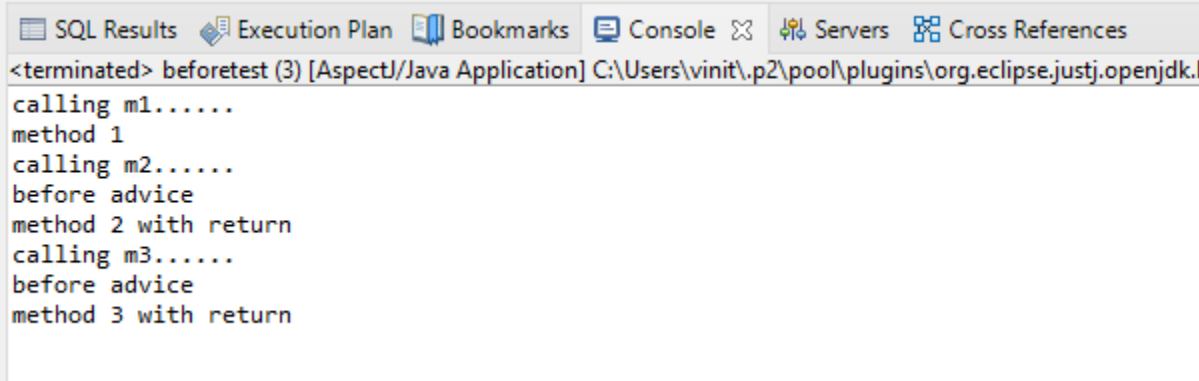
```
import org.springframework.context.ApplicationContext;
```

```
import org.springframework.context.support.ClassPathXmlApplicationContext;
```

```
public class beforetest {
```

```
    public static void main(String[] args) {  
        ApplicationContext context = new  
ClassPathXmlApplicationContext("aopctx1.xml");  
        beforeoperation e = (beforeoperation) context.getBean("opBean");  
        System.out.println("calling m1.....");  
        e.msg();  
        System.out.println("calling m2.....");  
        e.m();  
        System.out.println("calling m3.....");  
        e.k();  
  
    }  
  
}
```

Output :



```
<terminated> beforetest (3) [AspectJ/Java Application] C:\Users\vinit\.p2\pool\plugins\org.eclipse.justj.openjdk.l  
calling m1.....  
method 1  
calling m2.....  
before advice  
method 2 with return  
calling m3.....  
before advice  
method 3 with return
```

Problem Statement 2 : Write a program to demonstrate Spring AOP – after advice.

Solution :

Afteraopdata.java

```
package bvimit.edu;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;

@Aspect
public class afteraopdata {

    @Pointcut("execution(int afteroperation.*(..))")
    public void p() {}

    @After("p()")
    public void myadvice(JoinPoint jp)
    {
        System.out.println("after advice");
    }
}
```

afteroperation.java

```
package bvimit.edu;

public class afteroperation {
    public void msg() {System.out.println("method 1");}
    public int m() {System.out.println("method 2 with return");return 2;}
    public int k() {System.out.println("method 3 with return");return 3;}
}
```

aopctx.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"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="opBean" class="bvimit.edu.afteroperation"></bean>

    <bean id="trackMyBean" class="bvimit.edu.afteraopdata"></bean>
```

```

<bean
class="org.springframework.aop.aspectj.annotation.AnnotationAwareAspectJAutoProxyCreator"
></bean>
</beans>

```

aftertest.java

```
package bvimit.edu;
```

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
```

```

public class aftertest {

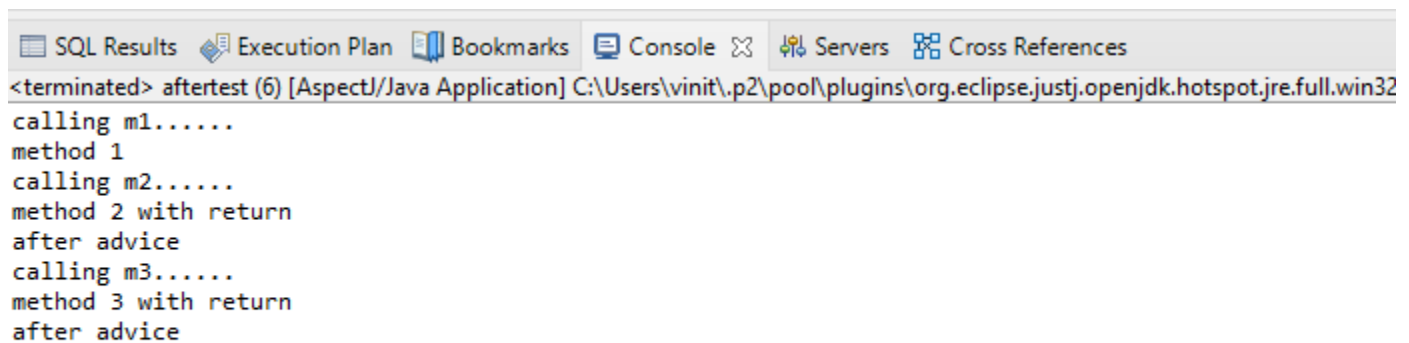
    public static void main(String[] args) {
        ApplicationContext context = new
ClassPathXmlApplicationContext("aopctx.xml");
        afteroperation e = (afteroperation) context.getBean("opBean");
        System.out.println("calling m1.....");
        e.msg();
        System.out.println("calling m2.....");
        e.m();
        System.out.println("calling m3.....");
        e.k();

    }

}

```

Output :



```

SQL Results  Execution Plan  Bookmarks  Console  Servers  Cross References
<terminated> aftertest (6) [AspectJ/Java Application] C:\Users\vinit\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32
calling m1.....
method 1
calling m2.....
method 2 with return
after advice
calling m3.....
method 3 with return
after advice

```

Problem Statement 3 : Write a program to demonstrate Spring AOP – around advice.

Solution :

Bankaopdata.java

```
package bvimit.edu;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;

@Aspect
public class Bankaopdata {

    @Pointcut("execution(* Bank.*(..))")
    public void a() {}

    @Around("a()")
    public Object myadvice(ProceedingJoinPoint p)throws Throwable
    {
        System.out.println("Around concern Before calling actual method");
        Object obj=p.proceed();
        System.out.println("Around Concern After calling actual method");
        return obj;
    }
}
```

Bank.java

```
package bvimit.edu;

public class Bank {
    public void welcome() {System.out.println("welcome to bank");}
    public int icici() {System.out.println("icici bank interest rate");return 7;}
    public int pnb() {System.out.println("pnb bank interest rate");return 6;}
}
```

Bankaopdata.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"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="opBean" class="bvimit.edu.Bank"></bean>
    <bean id="trackMyBean" class="bvimit.edu.Bankaopdata"></bean>

    <bean
class="org.springframework.aop.aspectj.annotation.AnnotationAwareAspectJAutoProxyCreator"
></bean>
</beans>
```

Banktest.java

```
package bvimit.edu;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

publicclass Banktest {

    privatestatic ApplicationContext context;

    publicstaticvoid main(String[] args) {
        context = new ClassPathXmlApplicationContext("Bankaopdata.xml");

        Bank e =(Bank) context.getBean("opBean");
        System.out.println("Calling welcome method...");
        e.welcome();
        System.out.println("Calling icici method...");
        e.icici();
        System.out.println("Calling pnb method...");
        e.pnb();
    }

}
```

Output :

SQL Results Execution Plan Bookmarks Console Servers Cross References

<terminated> Banktest (6) [AspectJ/Java Application] C:\Users\vinit\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201-C

Around concern Before calling actual method
welcome to bank
Around Concern After calling actual method
Around concern Before calling actual method
icici bank interest rate
Around Concern After calling actual method
Around concern Before calling actual method
pnb bank interest rate
Around Concern After calling actual method

Problem Statement 4 : Write a program to demonstrate Spring AOP – after returning advice.

Solution :

Bankaopdata.java

```
package bvimit.edu;
```

```
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;
```

```
@Aspect
```

```
public class Bankaopdata {
```

```
    @AfterReturning(
        pointcut = "execution(* Bank.*(..))",
        returning = "result")
```

```
    public void myadvice(JoinPoint jp, Object result)
```

```
    {
```

```
        System.out.println("AfterReturning concern");
```

```
        System.out.println("Result in advice" + result);
```

```
    }
```

```
}
```

Bank.java

```
package bvimit.edu;
```

```
public class Bank {
```

```
    public void welcome() {System.out.println("welcome to bank");}
```

```
    public int ici() {System.out.println("icici bank interest rate");return 7;}
```

```
    public int pnb() {System.out.println("pnb bank interest rate");return 6;}
```

```
}
```

Bankaopdata.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"
```

```
    xsi:schemaLocation="http://www.springframework.org/schema/beans
```

```
http://www.springframework.org/schema/beans/spring-beans.xsd">
```

```
<bean id="opBean" class="bvimit.edu.Bank"></bean>
```

```
<bean id="trackMyBean" class="bvimit.edu.Bankaopdata"></bean>
```

```
<bean  
class="org.springframework.aop.aspectj.annotation.AnnotationAwareAspectJAutoProxyCreator"  
></bean>  
</beans>
```

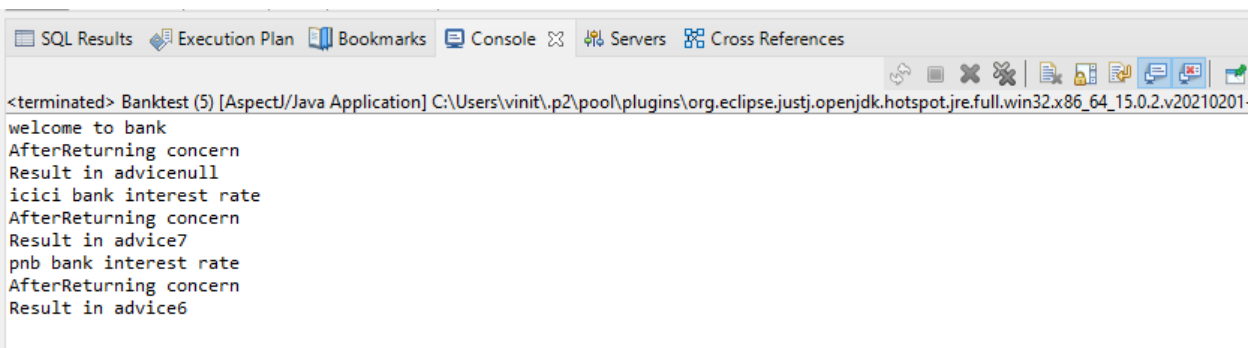
Banktest.java

```
package bvimit.edu;
```

```
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;
```

```
publicclass Banktest {  
  
    privatestatic ApplicationContext context;  
  
    publicstaticvoid main(String[] args) {  
        context = new ClassPathXmlApplicationContext("Bankaopdata.xml");  
  
        Bank e =(Bank) context.getBean("opBean");  
        //System.out.println("Calling welcome method...");  
        e.welcome();  
        //System.out.println("Calling icici method...");  
        e.icici();  
        //System.out.println("Calling pnb method...");  
        e.pnb();  
    }  
}
```

Output :



```
<terminated> Banktest (5) [AspectJ/Java Application] C:\Users\vinit\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.2.v20210201...  
welcome to bank  
AfterReturning concern  
Result in advicenull  
icici bank interest rate  
AfterReturning concern  
Result in advice7  
pnb bank interest rate  
AfterReturning concern  
Result in advice6
```

Problem Statement 5 : Write a program to demonstrate Spring AOP – after throwing advice.

Solution :

Operationaop_at.java

```
package bvimit.edu;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Aspect;

@Aspect
publicclass Operationaop_at {
    @AfterThrowing(
        pointcut = "execution(* Operation_at.*(..))", throwing = "error")
    publicvoid myadvice(JoinPoint jp, Throwable error)
    {
        System.out.println("AfterThrowing concern");
        System.out.println("Exception is: "+error);
        System.out.println("end of after throwing advice....");
    }
}
```

Operation_at.java

```
package bvimit.edu;
publicclass Operation_at {

    publicvoid validate(int att)throws Exception{
        if(att<75) {
            thrownew ArithmeticException("Not eligible for exam");
        }
        else {
            System.out.println("Eligible for exam");
        }
    }
}
```

validctx.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"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="opBean" class="bvimit.edu.Operation_at"></bean>

<bean id="trackMyBean" class="bvimit.edu.Operationaop_at"></bean>

<bean
class="org.springframework.aop.aspectj.annotation.AnnotationAwareAspectJAutoProxyCreator"
></bean></beans>

```

TestValidation.java

```

package bvimit.edu;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

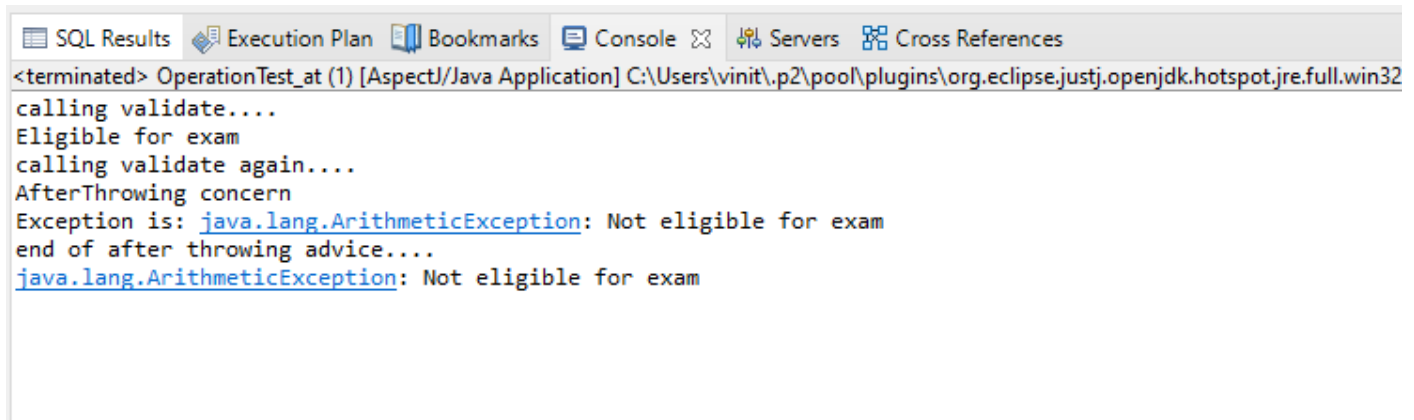
public class OperationTest_at {
private static ApplicationContext context;
        public static void main(String[] args) {
ApplicationContext context = new ClassPathXmlApplicationContext("validctx.xml");
        Operation_at op = (Operation_at) context.getBean("opBean");
        System.out.println("calling validate....");
        try {
            op.validate(85);
        } catch (Exception e) {System.out.println(e);}

        System.out.println("calling validate again....");

        try {
            op.validate(25);
        } catch (Exception e) {System.out.println(e);}
        }
}

```

Output :



```
<terminated> OperationTest_at (1) [AspectJ/Java Application] C:\Users\vinit\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32
calling validate....
Eligible for exam
calling validate again....
AfterThrowing concern
Exception is: java.lang.ArithmeticException: Not eligible for exam
end of after throwing advice....
java.lang.ArithmeticException: Not eligible for exam
```

Problem Statements 6: Write a program to demonstrate Spring AOP –pointcuts.

Solution:

Operation_pc.java

```
package bvimit.edu;
publicclass Operation_pc {

    publicvoid msg() {System.out.println("method 1");}
    publicint m() {System.out.println("method 2 with return");return 2;}
    publicint k() {System.out.println("method 3 with return");return 3;}
}
```

Aopdata_pc.java

```
package bvimit.edu;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Pointcut;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
@Aspect
public class Aopdata_pc {

    @Pointcut("execution(int Operation.*(..))")
    public void p(){}

    @After("p()")
    public void myadvice(JoinPoint jp)
    {
        System.out.println("After advice");
    }
    @Pointcut("execution(* Operation.*(..))")
    public void i(){}

    @Before("i()")
    public void myadvice1(JoinPoint jp)
    {
        System.out.println("Before advice");
    }
}
```

Test_pc.java

```
package bvimit.edu;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test_pc {
    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext("aopctx_pc.xml");
        Operation_pc e=(Operation_pc)context.getBean("opBean");
        System.out.println("calling m1...");
        e.msg();
        System.out.println("calling m2...");
        e.m();
    }
}
```

```

        System.out.println("calling m3...");
        e.k();
    }
}

```

aopctx_pc.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"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="opBean" class="bvimit.edu.Operation_pc"></bean>

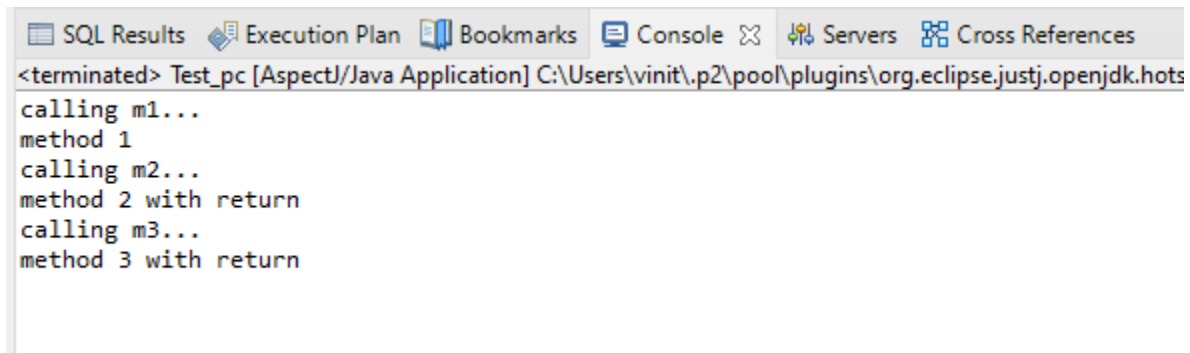
    <bean id="trackMyBean" class="bvimit.edu.Aopdata_pc"></bean>

    <bean
class="org.springframework.aop.aspectj.annotation.AnnotationAwareAspectJAutoProxyCreator"
></bean>

</beans>

```

Output:



The screenshot shows the Eclipse IDE's Console window. The title bar includes tabs for SQL Results, Execution Plan, Bookmarks, Console, Servers, and Cross References. The Console output shows the following sequence of events:

```

<terminated> Test_pc [AspectJ/Java Application] C:\Users\vinit\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot
calling m1...
method 1
calling m2...
method 2 with return
calling m3...
method 3 with return

```