# Basic Spring 4.0

Lesson 6: Aspect Oriented Programming (AOP)

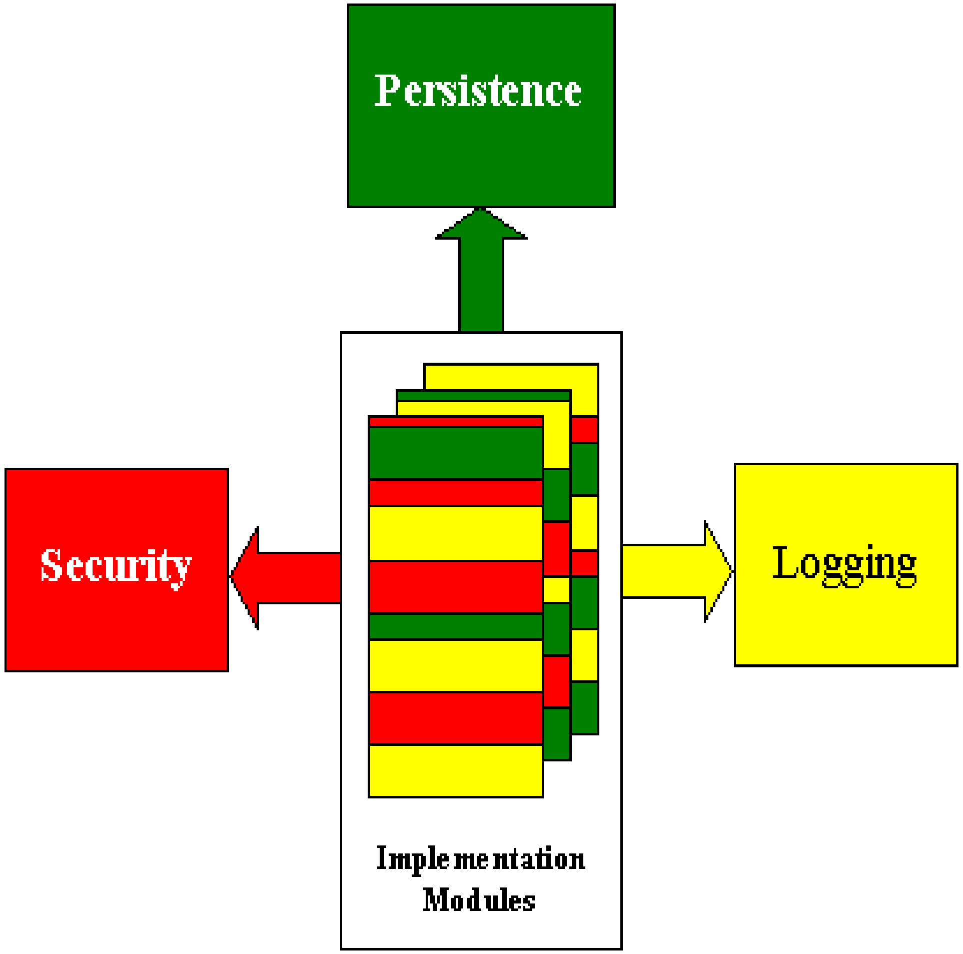
# Lesson Objectives

- Introduction to Spring AOP
  - Learn AOP basics and terminologies
  - Understand key AOP terminologies
  - Understand the different ways that Spring supports AOP

This image cannot currently be displayed.

# Introduction to AOP

- AOP complements OOP
- Aspects enable the modularization of concerns that cut across multiple types and objects
- AOP complements Spring IoC to provide a very capable middleware solution



## Understanding AOP: Example

- An Example:

```
void transfer(Account src, Account tgt, int amount) {
  if (src.getBalance() < amount) {
    throw new InsufficientFundsException();
  }
   src.withdraw(amount);
  tgt.deposit(amount);
}
```

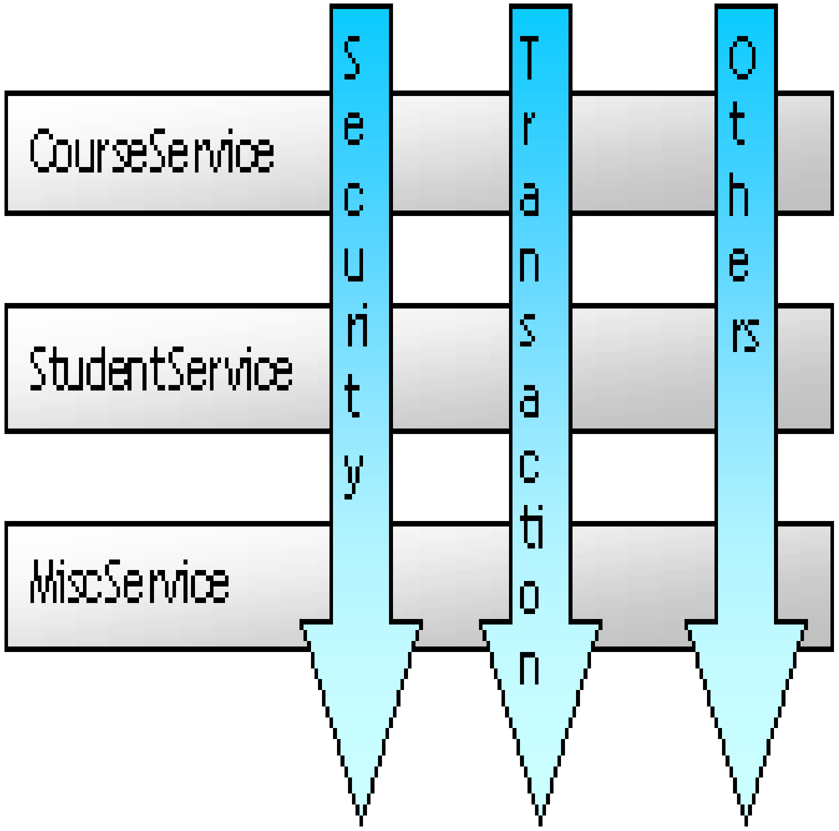
# Understanding AOP: Example

```
void transfer(Account src, Account tgt, int amount) {
      if (!getCurrentUser().canPerform(OP_TRANSFER))
                        throw new SecurityException();
      if (amount < 0)
                      throw new NegativeTransferException();
     if (src.getBalance() < amount) {
                        throw new InsufficientFundsException();  }
     Transaction tx = database.newTransaction();
      try {
            src.withdraw(amount);
              tgt.deposit(amount);
              tx.commit();
          systemLog.logOperation(OP_TRANSFER, src, tgt, amount);
  }
  catch(Exception e) {   tx.rollback();  }
```

# AOP and Spring

- AOP attempts to separate concerns, that is, break down program into distinct parts that overlap in functionality sparingly.

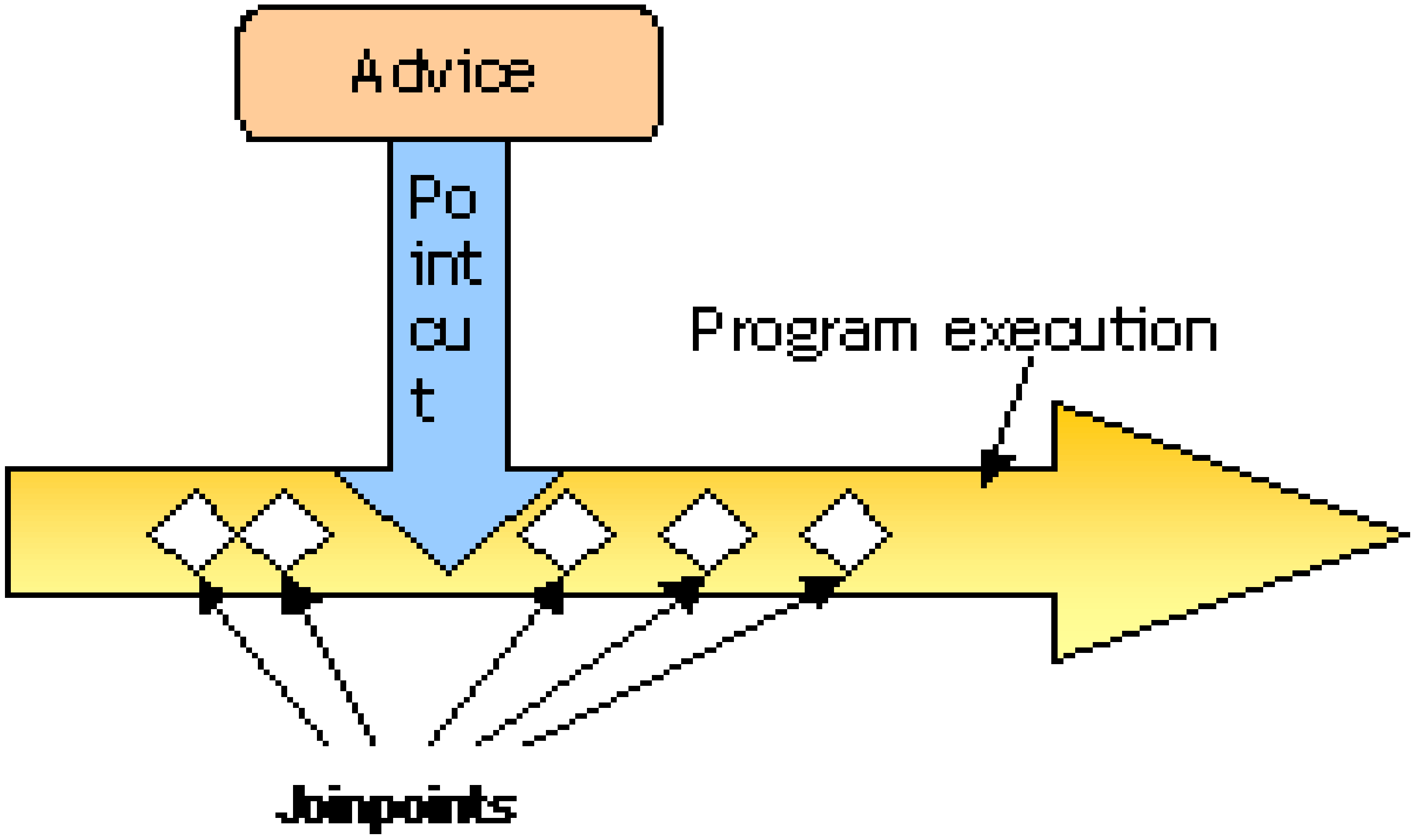
- In particular, AOP focuses on the modularization and encapsulation of cross-cutting concerns.



## AOP Terminology

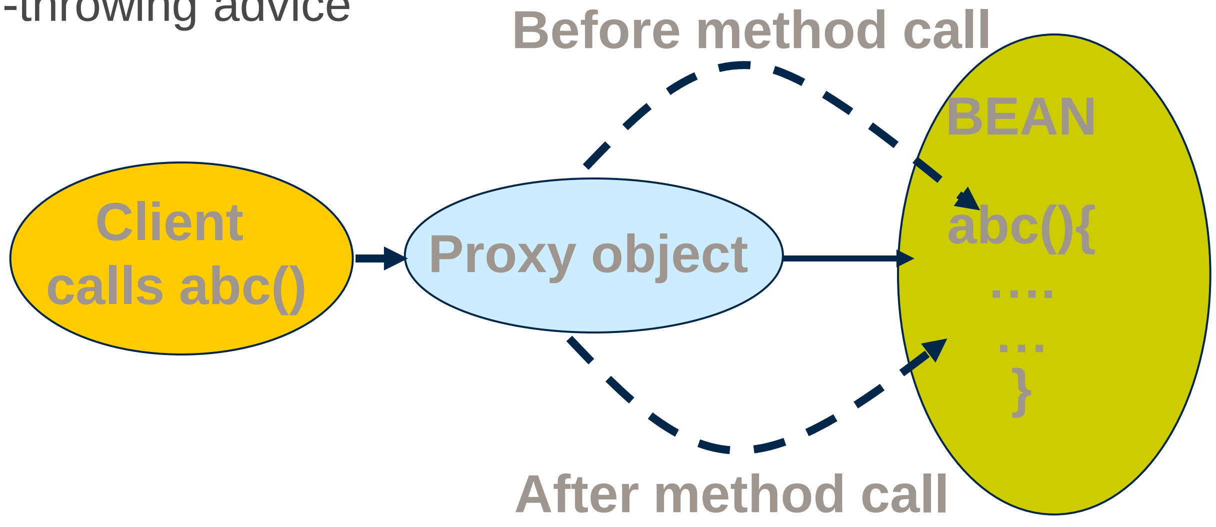
- Aspect :
  - the cross-cutting functionality being implemented
- Advice :
  - the actual implementation of aspect that is advising your application of a new behavior. It is inserted into application at joinpoints
- Join-point :
  - a point in the execution of the application where an aspect can be plugged in
- Point-cut :
  - defines at what joinpoints an advice should be applied
- Target :
  - the class being advised
- Proxy :
  - the object created after applying advise to the target

# AOP Terminology



## AOP Terminology

- Types of advices:
  - Before advice
  - After advice
  - After-returning advice
  - Around advice
  - After-throwing advice



# AOP Vs OOP

| AOP | OOP |
|---|---|
| **Aspect –** code unit that encapsulates pointcuts, advice, and attributes | **Class –** code unit that encapsulates methods and attributes |
| **Pointcut –** define the set of entry points (triggers) in which advice is executed | **Method signature –** define the entry points for the execution of method bodies |
| **Advice –** implementation of cross cutting concern | **Method bodies –**implementation of the business logic concerns |
| **Weaver –** construct code (source or object) with advice | **Compiler –** convert source code to object code |

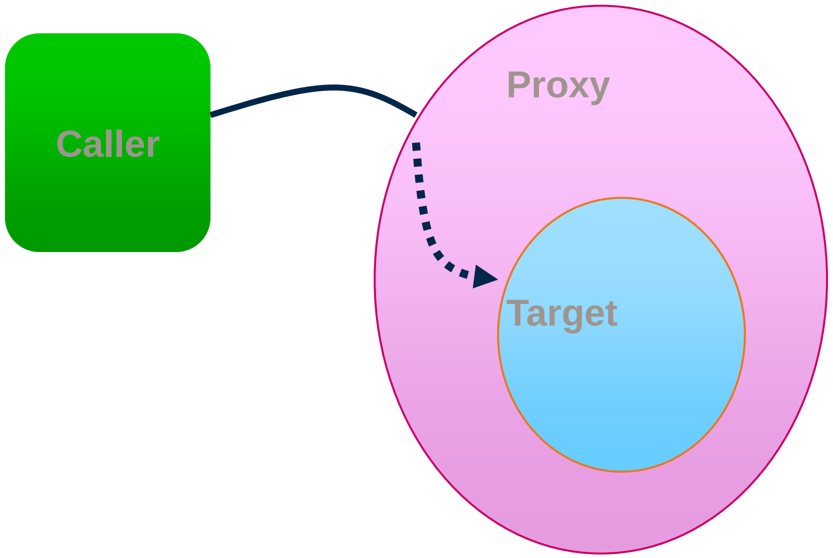
## AOP Frameworks

- There are three dominant AOP frameworks:
  - AspectJ (http://eclipse.org/aspectj)
  - JBoss AOP (http://www.jboss.org/jbossaop)
  - Spring AOP (http://www.springframework.org)
- AOP support in Spring borrows a lot from the AspectJ project.
- Spring supports AOP in the following four flavors:
  - Classic Spring proxy-based AOP
  - @AspectJ annotation-driven aspects
  - Pure-POJO aspects
  - Injected AspectJ aspects (available in all versions of Spring)

variations on Spring's proxy-based AOP

## AOP Frameworks

- Key points of Spring’s AOP framework:
  - All advices are written in Java
  - Spring advises objects at runtime
  - Spring’s AOP support is limited to method interception



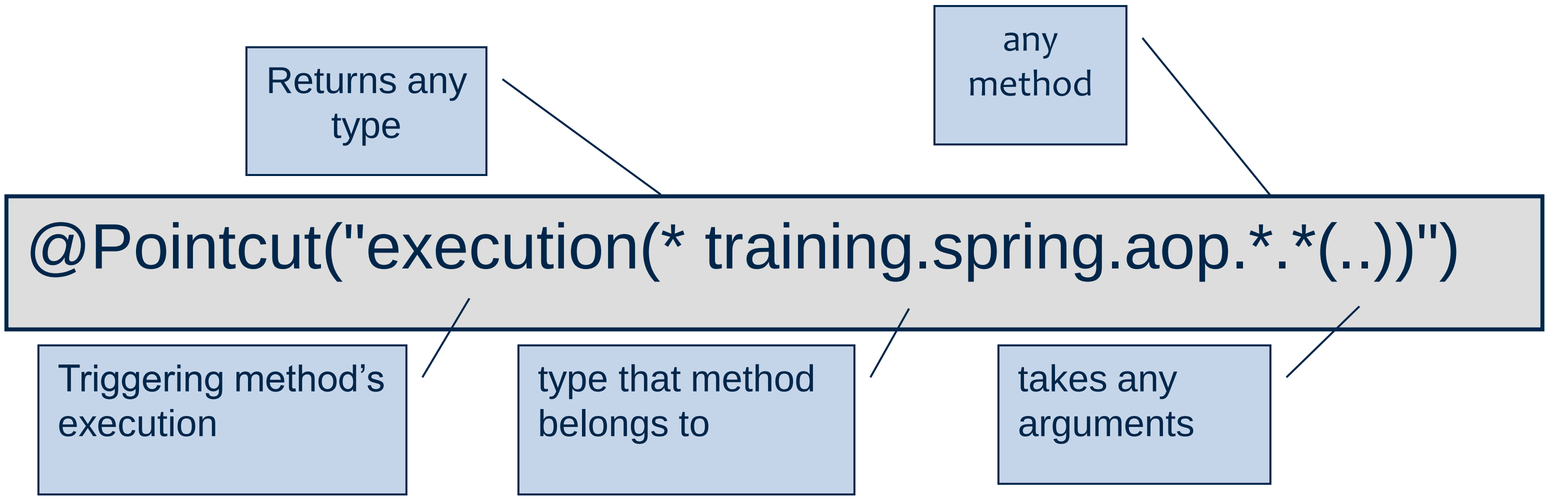
# AspectJ’s pointcut expression language

- AspectJ pointcut designators supported in Spring AOP

| AspectJ | Description |
|---|---|
| args() | Limits matching to the execution of methods whose arguments are instances of the given types |
| @args() | Limits matching to the execution of methods whose arguments are annotated with the given annotation types |
| execution() | Matches join points that are method executions |
| this() | Limits matching to those where the bean reference of the AOP proxy is of a given type |
| target() | Limits matching to those where the target object is of a given type |
| @target | Limits matching to join points where the class of the executing object has an annotation of the given type |
| within() | Limits matching to join points within certain types |
| @within | Limits matching to join points within types that have the given annotation |
| @annotation | Limits matching to those where the subject of the join point has the given annotation |

# AspectJ’s pointcut expression language

- Writing pointcuts



```
@Pointcut("execution(* training.spring.aop.*.*(..))")
```

```
@Pointcut("execution(* training.spring.aop.*.*(..))“ && within(training.spring.aop))
```

## Spring’s @AspectJ support

```
package training.spring.aop;;
public interface Business {
        void doSomeOperation();
}
```

```
package training.spring.aop;
public class BusinessImpl implements Business {
        public void doSomeOperation() {
                System.out.println("I do what I do best, i.e sleep.");
                try { Thread.sleep(2000);
                } catch (InterruptedException e) {
                        System.out.println("How dare you to wake me up?"); }
                System.out.println("Done with sleeping.");
      }}
```

# Spring’s @AspectJ support

```
@Aspect
public class BusinessProfiler {

        @Pointcut("execution(* training.spring.aop.*.*(..))")
        public void businessMethods() { }

        @Around("businessMethods()")
        public Object profile(ProceedingJoinPoint joinpoint) throws Throwable {
                long start = System.currentTimeMillis();
                System.out.println("Going to call the method.");
                Object output = joinpoint.proceed();
                System.out.println("Method execution completed.");
                long elapsedTime = System.currentTimeMillis() - start;
                System.out.println("Method executed");
                return output;
        }
}
```

is the aspect

defines a reusable pointcut within an aspect.

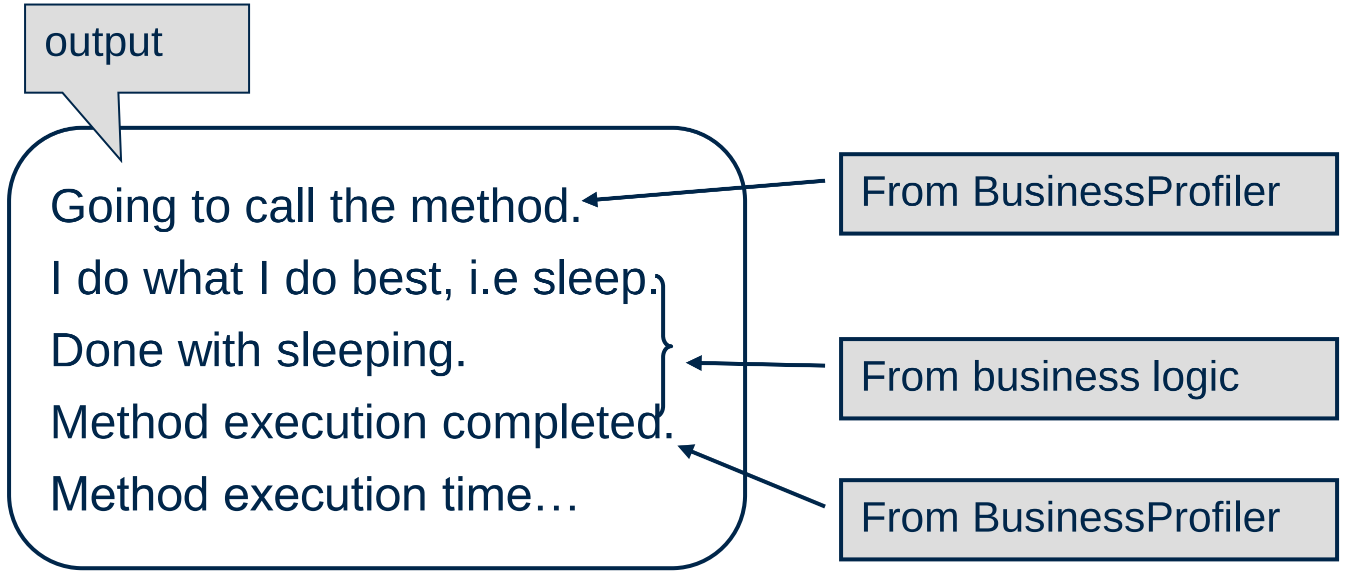
# Spring's @AspectJ support

```
<beans …
xmlns:aop="http://www.springframework.org/schema/aop"
…
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">
      <aop:aspectj-autoproxy />    <!-- Enable the @AspectJ support -->
     <bean id="businessProfiler" class="training.spring.aop.BusinessProfiler" />
     <bean id="myBusinessClass" class="training.spring.aop.BusinessImpl" />
</beans>
```

```
public class BusinessDemo {
     public static void main(String[] args) {
          ApplicationContext context =
                         new ClassPathXmlApplicationContext("business.xml");
          Business bc = (Business) context.getBean("myBusinessClass");
          bc.doSomeOperation();
        }}
```

# Demo: DemoSpring_AOP2

- This demo shows how to apply cross-cutting functionality into Spring application using AOP with @AspectJ support



```
Going to call the method.
I do what I do best, i.e sleep.
Done with sleeping.
Method execution completed.
Method execution time…
```

## Declaring aspects in XML

| AOP config element | Purpose |
|---|---|
| <aop:advisor> | Defines an AOP advisor. |
| <aop:after> | Defines an AOP after advice (regardless of whether the advised method returns successfully). |
| <aop:after-returning> | Defines an AOP after-returning advice. |
| <aop:after-throwing> | Defines an AOP after-throwing advice. |
| <aop:around> | Defines an AOP around advice. |
| <aop:aspect> | Defines an aspect. |
| <aop:aspectj-autoproxy> | Enables annotation-driven aspects using @AspectJ. |
| <aop:before> | Defines an AOP before advice. |
| <aop:config> | The top-level AOP element. Most <aop:*> elements must be contained within <aop:config>. |
| <aop:declare-parents> | Introduces additional interfaces to advised objects that are transparently implemented. |
| <aop:pointcut> | Defines a pointcut. |

6.3: Schema-based AOP support

## Declaring aspects in XML

```
public class MyAdvice {
    public void beforeMethodCall() {
        System.out.println("Before Method Call");
    }
     public void aroundMethodCall() {
         System.out.println("Around Method Call");
     }
     public void afterMethodCall() {
         System.out.println("After Method Call");
    }
     public void afterException() {
           System.out.println("After Exception thrown");
     }}
```

```
<bean id="advice" class="training.spring.schemaAOP.MyAdvice" />
```

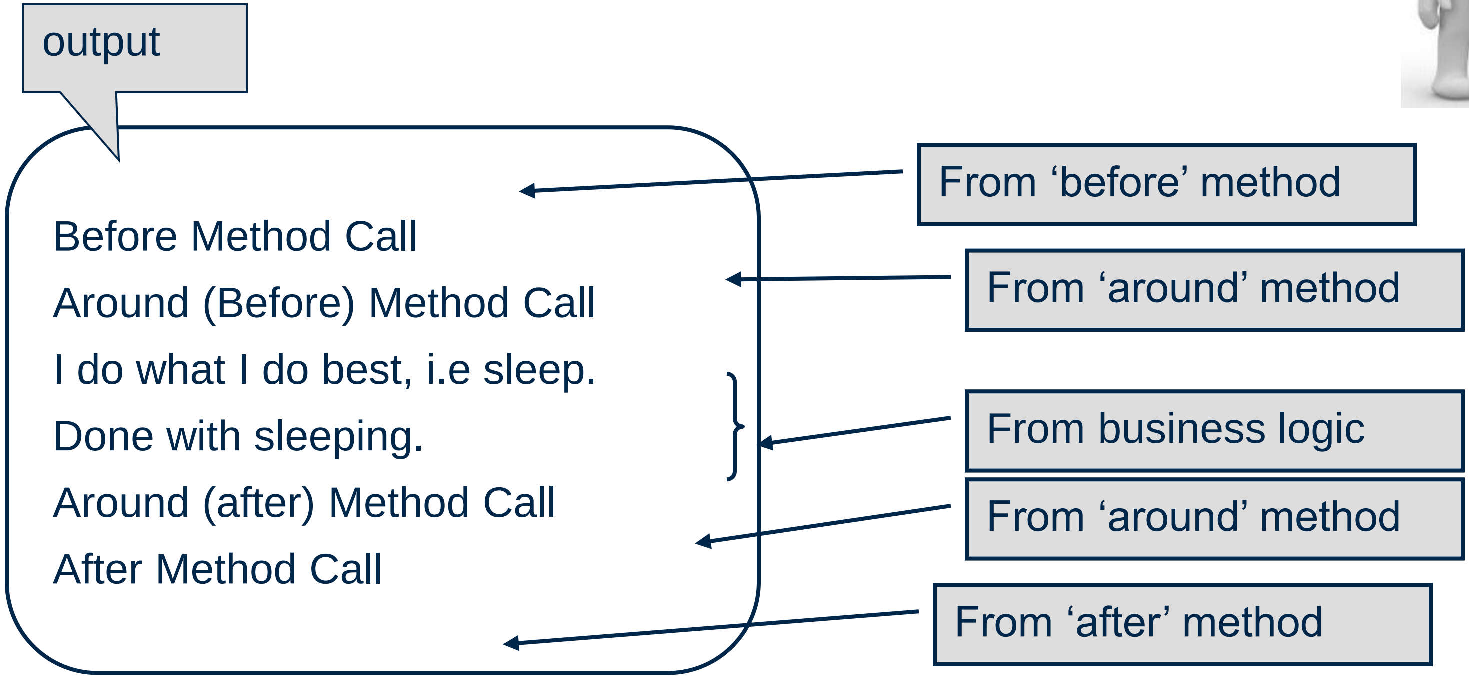
6.3: Schema-based AOP support

# Declaring aspects in XML

```
<beans … >
<bean id="advice" class="training.spring.schemaAOP.MyAdvice" />
<bean id="myBusinessClass" class="training.spring.schemaAOP.BusinessImpl" />
   <aop:config>
      <aop:aspect ref="advice">
          <aop:before
                 pointcut="execution(* training…..BusinessImpl.doBusiness(..))"
                 method="beforeMethod" />
          <aop:around
                 pointcut="execution(* training…..BusinessImpl.doBusiness(..))"
                 method="aroundMethod" />
             …
          <aop:after-throwing
                  pointcut="execution(* training…..BusinessImpl.doBusiness(..))"
                  method="afterException" />
        </aop:aspect>
     </aop:config>
  </beans>
```

# Demo: DemoSpring_AOP1

- These demos shows how to apply cross-cutting functionality into Spring application using Schema-based AOP support



```
Before Method Call
Around (Before) Method Call
I do what I do best, i.e sleep.
Done with sleeping.
Around (after) Method Call
After Method Call
```

## Integrating Log4j framework into AOP

- Logging has a lot of characteristics that make it a prime candidate for implementation as an aspect:
  - Logging code is often duplicated across an application, leading to a lot of redundant code across multiple components in the application.
  - Logging logic does not provide any business functionality; it's not related to the domain of a business application.

# Eg : Integrating Log4j framework into AOP

```
public interface SampleInterface {
   public void process();
   public String getName();
   public int getAge();
   public void setAge(int age);
   public void setName(String str);
}
```

Business interface and its implementing class

```
public class SampleBean implements SampleInterface {
   private String name;
   private int age;
   //getter/setter methods for these properties

   public void process() {
         System.out.println("checking with the process() method-1");
    }
}
```

Business method

# Eg : Integrating Log4j framework into AOP

The interceptor

```
public class LoggingInterceptor {
    Logger myLog;
    public Object logs(ProceedingJoinPoint call) throws Throwable {
        Object point = null;
        myLog = Logger.getLogger(LoggingInterceptor.class);
        PropertyConfigurator.configure("log4j.properties");
        try {
            System.out.println("from logging aspect: entering method "
                                                + call.getSignature().getName());
            myLog.info("Hello : It is " + new java.util.Date().toString());
            point = call.proceed();
            System.out.println("from logging aspect: exiting method ");
        } catch (Exception e) {
            System.out.println("Logging the exception with date " + new Date());
      }
     return point;
  }
```

# Eg : Integrating Log4j framework into AOP

The configuration file

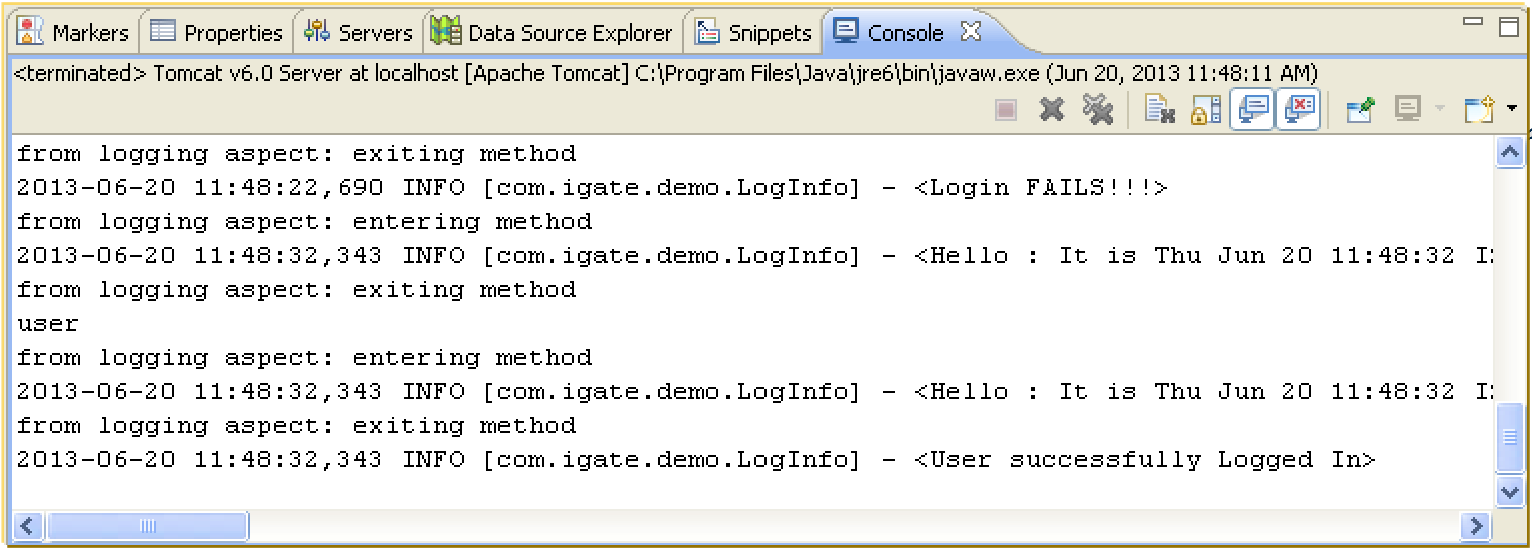
```
<beans  …..>
    <bean id="sampleBean" class="training.spring.aop.logger.SampleBean"/>
    <bean id="loggingInterceptor"
                class="training.spring.aop.logger.LoggingInterceptor" />
    <aop:config>
        <aop:aspect ref="loggingInterceptor">
            <aop:pointcut id="myCutLogging" expression="execution(* *.p*(..))"/>
               <!-- - when you want to do? before method ,after method,..... -->
             <aop:around pointcut-ref="myCutLogging" method="logs" />
         </aop:aspect>
     </aop:config>
</beans>
```

log4j.properties

```
log4j.rootLogger=debug, myAppender
log4j.appender.myAppender=org.apache.log4j.ConsoleAppender
log4j.appender.myAppender.layout=org.apache.log4j.SimpleLayout
```

# Demo: DemoMVC_AOP

- This demo shows how to integrate the Log4j logging framework with AOP using MVC based an application



```
<terminated> Tomcat v6.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre6\bin\javaw.exe (Jun 20, 2013 11:48:11 AM)
from logging aspect: exiting method
2013-06-20 11:48:22,690 INFO [com.igate.demo.LogInfo] - <Login FAILS!!!>
from logging aspect: entering method
2013-06-20 11:48:32,343 INFO [com.igate.demo.LogInfo] - <Hello : It is Thu Jun 20 11:48:32 I
from logging aspect: exiting method
user
from logging aspect: entering method
2013-06-20 11:48:32,343 INFO [com.igate.demo.LogInfo] - <Hello : It is Thu Jun 20 11:48:32 I
from logging aspect: exiting method
2013-06-20 11:48:32,343 INFO [com.igate.demo.LogInfo] - <User successfully Logged In>
```

output

# Lab

- Lab-3 from the lab guide

# Summary

- We have so far seen:
  - AOP basics and terminologies
  - Key AOP terminologies
  - The different ways that Spring supports AOP.

# Review Questions

- Question 1: In Spring’s aop configuration namespace, how is an aspect defined?
  - Option 1 : <aop:advisor>
  - Option 2 : <aop:aspect>
  - Option 3 : <aop:declare-aspect>
  - Option 4 : <aop:config>

- Question 2 : In addition to method join points, Spring also supports field and constructor joinpoints.
  - Option 1 : True
  - Option 1 : False

# Review Questions

- Question 3 : ProceedingJoinPoint’s _______ method must be used to provide access to the advised method, so that it can execute.
  - Option 1 : invoke()
  - Option 2 : continue()
  - Option 3 : proceed()
  - Option 4 : next()

- Question 4 : <aop:aspectj-autoproxy/> automatically proxies beans whose methods match the pointcuts defined with @Pointcut annotations in @Aspect-annotated beans.
  - Option 1 : True
  - Option 1 : False

This image cannot currently be displayed.

# Review Questions

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| S | T | A | R | G | E | T | P |
| J | P | D | B | C | A | A | R |
| N | R | V | I | U | T | S | O |
| P | O | I | N | T | C | U | T |
| R | C | C | V | W | E | Y | E |
| O | E | E | O | Q | P | F | E |
| X | E | T | K | G | S | M | G |
| Y | D | E | E | G | A | H | E |

This image cannot currently be displayed.