Spring Framework
Module 3 – AOP

Evgeniy Krivosheev
Andrey Stukalenko
Vyacheslav Yakovenko
Last update: Feb, 2012

Contents

- Introduction
- AOP Activation in Spring container
- Pointcut language
- Advice Types
- Example AOP
- Additional Information
- Exercises

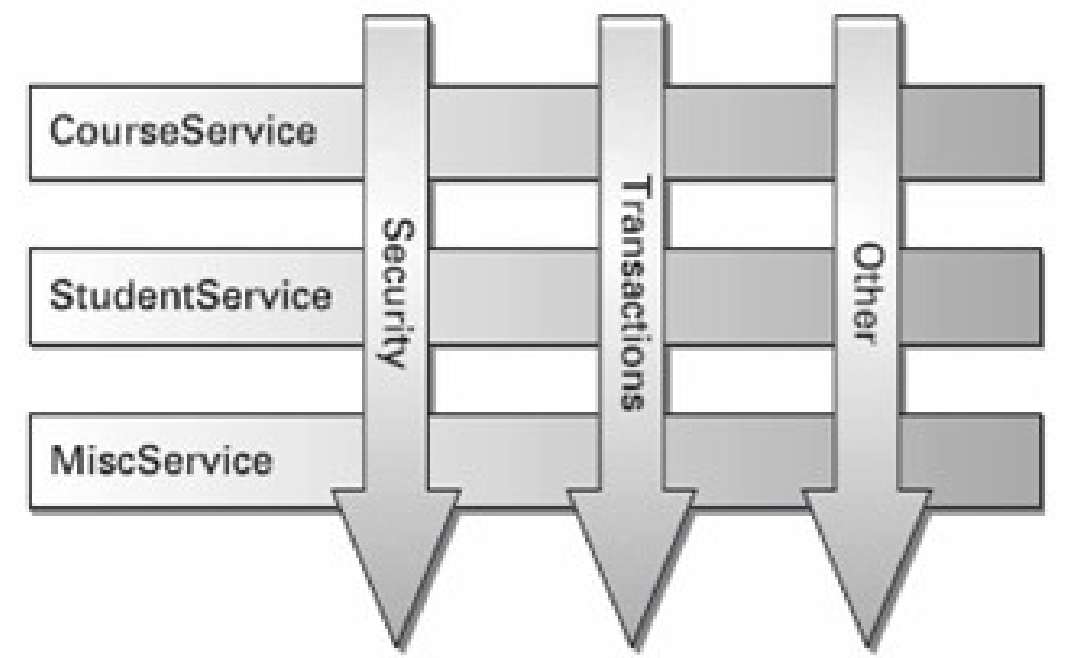
LUXOFT TRAINING

"Let us begin by defining some central AOP concepts and terminology. These terms are not Spring-specific… unfortunately, AOP terminology is not particularly intuitive; however, it would be even more confusing if Spring used its own terminology."

Spring Framework 3.1.0M1, 7.1.1. AOP Concepts

Spring Framework :: AOP :: Introduction

- Aspect Oriented Programming (AOP)
- AOP gives means for implementing orthogonal (crosscutting) functionality



Making it easier for you to grasp AOP, think about how this crosscutting business logic can be implemented on the basis of RDBMS.

LUXOFT TRAINING

Example of crosscutting logging based on RDBMS triggers:

```
/* Table level triggers */
 CREATE OR REPLACE TRIGGER DistrictUpdatedTrigger
 AFTER UPDATE ON district
 BEGIN
  INSERT INTO info VALUES ('table "district" has changed');
 END;
```

LUXOFT TRAINING

- AOP in Spring Framework is implemented by creating proxy object for the service you're interested in;
- Proxy object can be created using following libraries:
  - Using standard mechanism of creating dynamic proxies from J2SE;
  - Using CGLIB proxy.

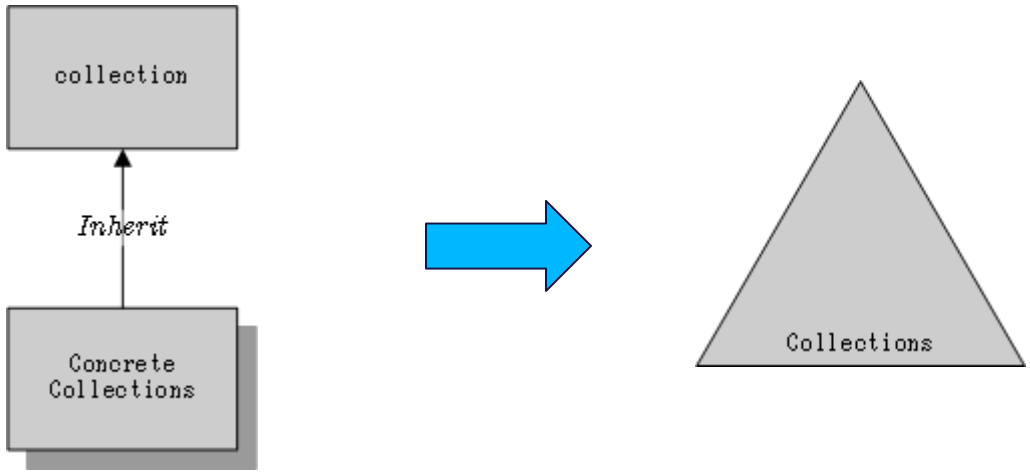
LUXOFT TRAINING

- The training will deal with standard J2SE dynamic proxies, so attendees have to understand this mechanism;
- You will find additional information on this subject in the following link: http://download.oracle.com/javase/1.3/docs/guide/refl (http://goo.gl/ucfM0)

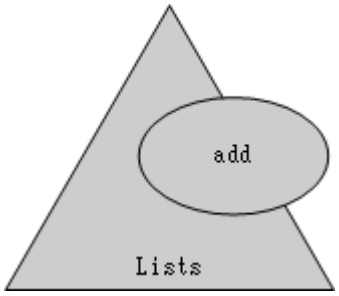
LUXOFT TRAINING

- What is crucial for understanding Spring AOP, is that dynamic proxy can only be created for class implementing interface;
- If interface is not implemented in class hierarchy, whatever the reason is, you have to use CGLIB proxies(not covered by this training).

From then onward we will draw hierarchy of objects that have interface in a vertex in the shape of triangle inherent to LePUS3 diagrams. At the same time, separate classes we will draw as usual UML rectangle:

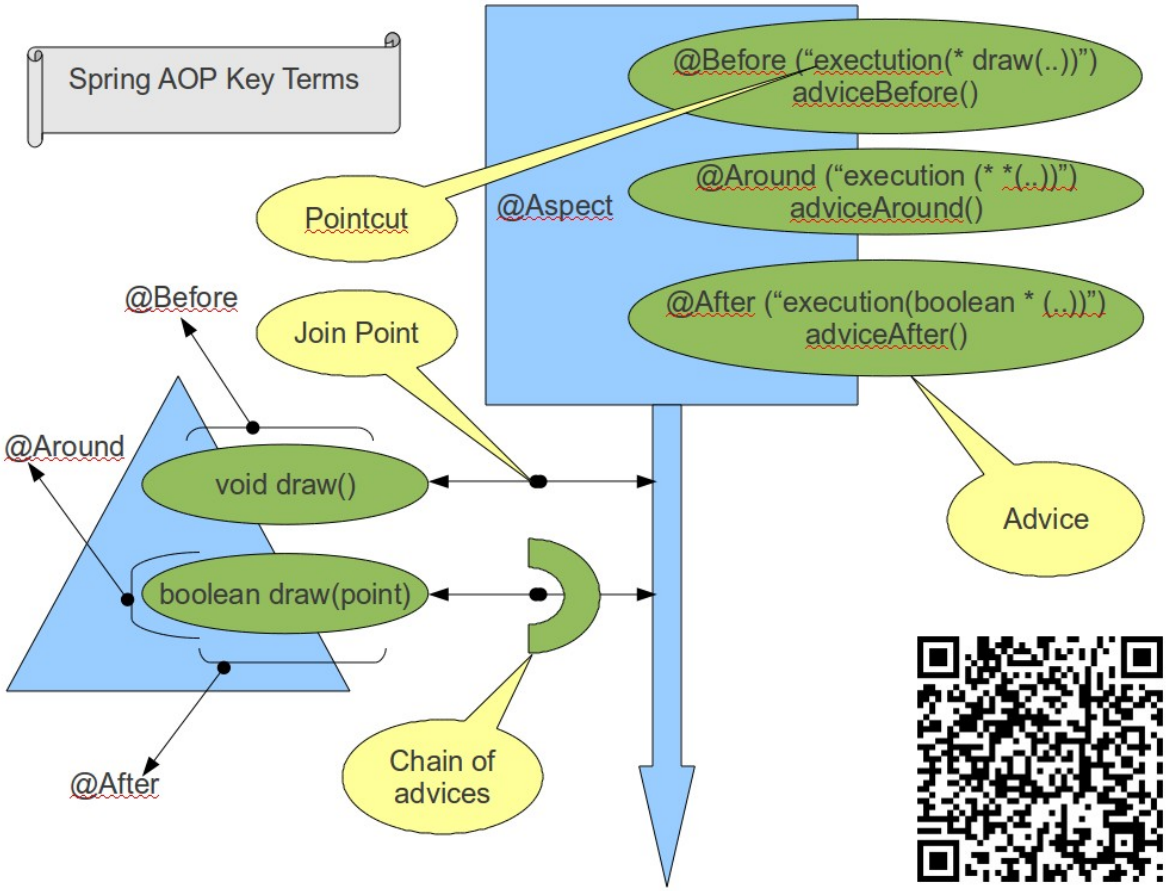


Those methods that are common for the whole class hierarchy (inherited from interface we're interested in) we will draw in the shape of an oval overlapping a rectangle or a triangle, which will be defined by the context:



Spring Framework :: AOP :: Introduction

There is a sketch below depicting key terms of AOP and their relationship:



LUXOFT TRAINING

- Aspect: a crosscutting functionality:
  - AOP in Spring is implemented in classes annotated with @Aspect (blue rectangle with dropdown arrow on the diagram);
- Examples:
  - Logging;
  - Data Access Restrictions;
  - Transactions, etc.

Join Point: a point during the execution of a program where an aspect can be implemented;

- A point where “regular” and crosscutting functionality intersect;
- Only method can be a join point in Spring Framework (methods of blue triangle in the class hierarchy on the diagram).

Advice: action taken by an Aspect at a particular join point. Provides application with new functionality.

In Spring AOP, Aspect method annotated as @Before, @Around, @After, etc., acts as Advice (see diagram).

Pointcut: a predicate defining at which join points an Advice shall be used.

Spring AOP uses the @AspectJ pointcut language by default.

The pointcut expression language will be examined further in detail.

Weaving: linking aspects with target objects for creating new proxy object;

Spring Framework uses two additional dependencies on the classpath in order to perform weaving at runtime:

- aspectjrt.jar
- aspectjweaver.jar

It is necessary to initiate creating dynamic proxies in a configuration file: <aop:aspectj-autoproxy />

Spring Framework :: AOP Activation

```
<?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: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/>
</beans>
```

Generally, in Spring AOP, a pointcut is expressed as :

execution(

modifiers-pattern?

ret-type-pattern

declaring-type-pattern?

name-pattern(param-pattern)

throws-pattern?)

? – optional parameters

LUXOFT TRAINING

Examples:

- execution (* *(..)) means the execution of any method regardless of return or parameter types;
- execution (int *(..)) means the execution of any method with no parameters that returns an int;
- execution (!static * *(..)) means the execution of any non-static method regardless of parameters;

Pointcut language

- bean: limits matching of join points to a particular Spring bean or a set of beans (when using wildcards)
- within: limits matching to any method within certain types
- this: limits matching to join points where AOP proxy bean is an instance of a given type
- target: limits matching to join points where the target object (i.e. application object being proxied) is an instance of a given type
- args: limits matching to join points where the arguments are instances of a given type
- @target: limits matching to join point where the class of the executing object has an annotation of a given type
- @args: limits matching to join points where the runtime types of actual arguments passed have the annotation of the given type
- @within: limits matching to join points within types that have given annotation
- @annotation: limits matching to join points where the subject of the join point (method being executed) has the given annotation

Pointcut language
Examples

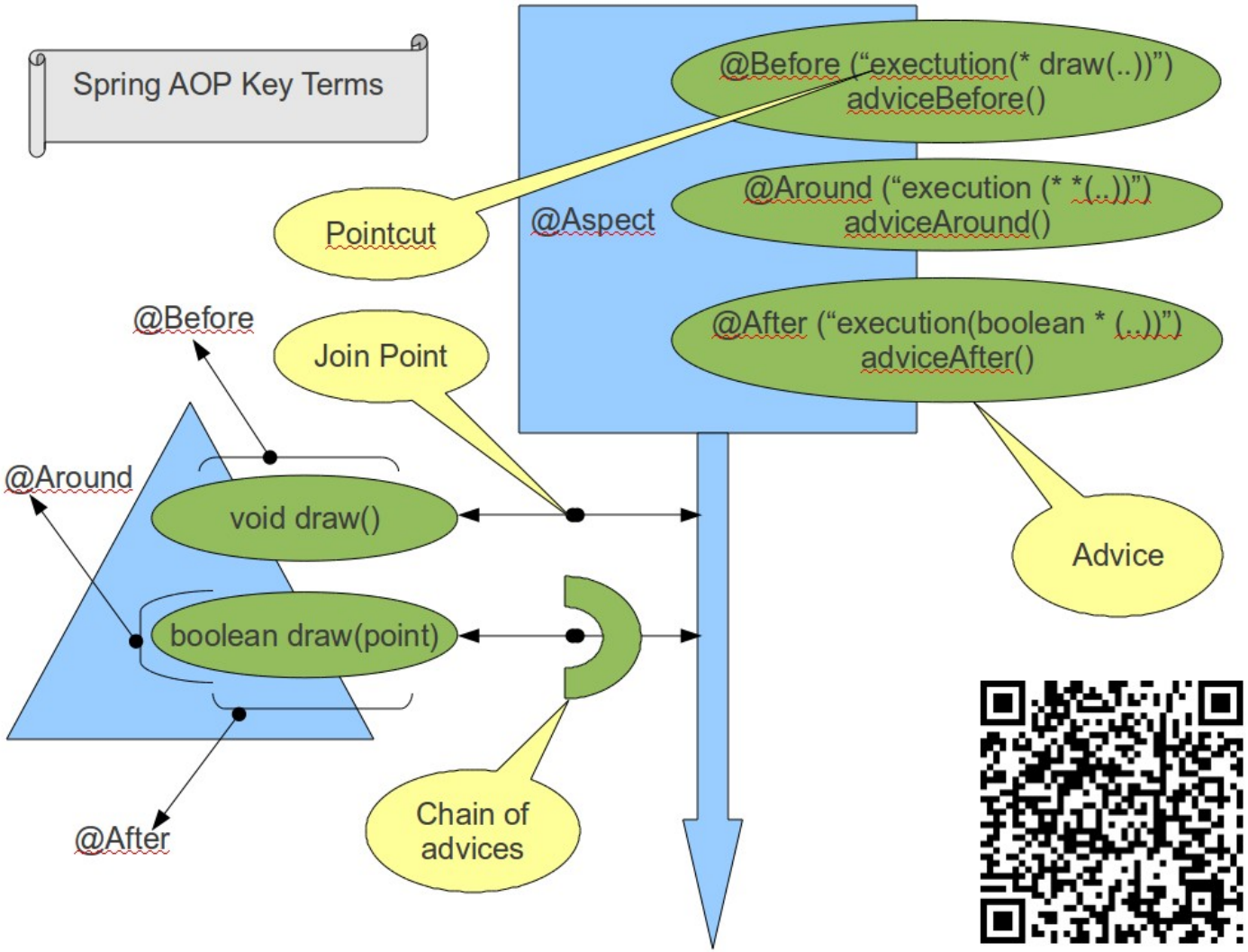
- execution(public * *(..))
- execution(* set*(..))
- execution(* com.xyz.service.AccountService.*(..))
- execution(* com.xyz.service.*.*(..))
- execution(* com.xyz.service..*.*(..))
- within(com.xyz.service.*)
- within(com.xyz.service..*)
- this(com.xyz.service.AccountService)
- target(com.xyz.service.AccountService)
- args(java.io.Serializable)
- @target(org.springframework.transaction.annotation.Transactional)
- @within(org.springframework.transaction.annotation.Transactional)
- @annotation(org.springframework.transaction.annotation.Transactional)
- @args(com.xyz.security.Classified)
- bean(tradeService)
- bean(*Service)

Spring :: AOP :: Pointcut language

Examples:

- execution (void Test.foo(..)): linking with foo method, class Test, regardless of parameters;
- execution (void Test.foo(int, String)): linking with foo method, class Test, with parameters int and String;
- execution (* foo.bar.*.dao.SomeClass.update*(..)): linking with any method beginning with “update” of dao sub-package ;
- Подробнее: http://www.eclipse.org/aspectj/doc/released/progguide/language-joinPoints.html (http://goo.gl/Vm79a)

Spring :: AOP :: Advice Types



Spring :: AOP :: Advice Types

LUXOFT TRAINING

@Before: executes before join point

The only way to cancel invocation of join point is to throw an exception

LUXOFT TRAINING

@AfterReturning: after a join point completes normally: for example, method invocation without throwing exceptions;

@AfterThrowing: is executed if a join point throws an exception;

@After (finally): in any case after join point invocation;

** A little bit more: Spring 3.1.RELEASE M1 Reference manual, 8.2.4 Declaring advice*

LUXOFT TRAINING

@Around: surrounds a joinpoint;

This is the most powerful kind of advice;

Chooses whether to proceed to the join point or to return its own value;

```
@Around("com.xyz.myapp.SystemArchitecture.businessService()")
public Object doBasicProfiling(ProceedingJoinPoint pjp) throws Throwable {
        // start stopwatch
        Object retVal = pjp.proceed();
        // stop stopwatch
        return retVal;
}
```

LUXOFT TRAINING

Algorithm:

- A customer enters a bar
- A bartender greets him
- The customer orders a drink
- If the customer has money
- He is being served
- The bartender asks customer what does he think
- If the customer is broke (isBroke)
- He is told something
- The bartender says good buy to customer

LUXOFT TRAINING

Quiz: Which algorithm steps belong to basic functionality and which to crosscutting functionality?

- A customer enters a bar
- A bartender greets him
- The customer orders a drink
- If the customer has money
- He is being served
- The bartender asks customer what does he think
- If the customer is broke (isBroke)
- He is told something
- The bartender says good buy to customer

LUXOFT TRAINING

- Basic function:
  - Ordering a drink;
- Auxiliary function:
  - greetings;
  - good buy;
  - questions.

LUXOFT TRAINING

Business objects:

```
public interface Bar {
    Squishee buySquishee(Customer customer);
}
public class ApuBar implements Bar {
    public Squishee buySquishee(Customer customer)  {
        if (customer.isBroke()){
            throw new CustomerBrokenException();
        }
        System.out.println("Here is your Squishee");
        return new Squishee("Usual Squishee");
    }
}
```

LUXOFT TRAINING

Auxiliary crosscutting functionality:

```
@Aspect
public class Politeness {

    @Before(value = "execution(* buySquishee(..))")
    public void sayHello() {
        System.out.println("Hello!");
    }
}
```

```
<bean id="politeness" class="aop.Politeness"/>
```

```
<bean id="bar" class="aop.ApuBar"/>
```

LUXOFT TRAINING

Auxiliary crosscutting functionality :

```
@AfterReturning(pointcut = "execution(* buySquishee(..))",
                returning = "retVal", argNames = "customer")
public void askOpinion(Object retVal) {
    System.out.println("Is " +
            ((Squishee)retVal).getName() + " Good Enough?");
}

@AfterThrowing("execution(* buySquishee(..))")
public void sayYouDontHaveEnoughMoney() {
    System.out.println("Hmmm...");
}

@After("execution(* buySquishee(..))")
public void sayGoodBye() {
    System.out.println("Good Bye!");
}
```

LUXOFT TRAINING

- Features of using @Around advices are described in much detail in sections 8.2.4.5, 6 of Spring 3.1.RELEASE Reference Manual;
- You must understand that when using dynamic proxies, aspect will only work in case if target bean is called through IoC container;

Exercises

- №5: Using Spring AOP. @AspectJ style. :
  - 30 - 45 min for practice;
  - 15 min for discussion;

Any questions!?