Mindtree

Welcome to possible

Aspect Oriented Programming

Objectives

- Define proxy factory bean class
- Define AOP namespace Creating Aspect, choosing pointcuts, declarative aop using aop namespace

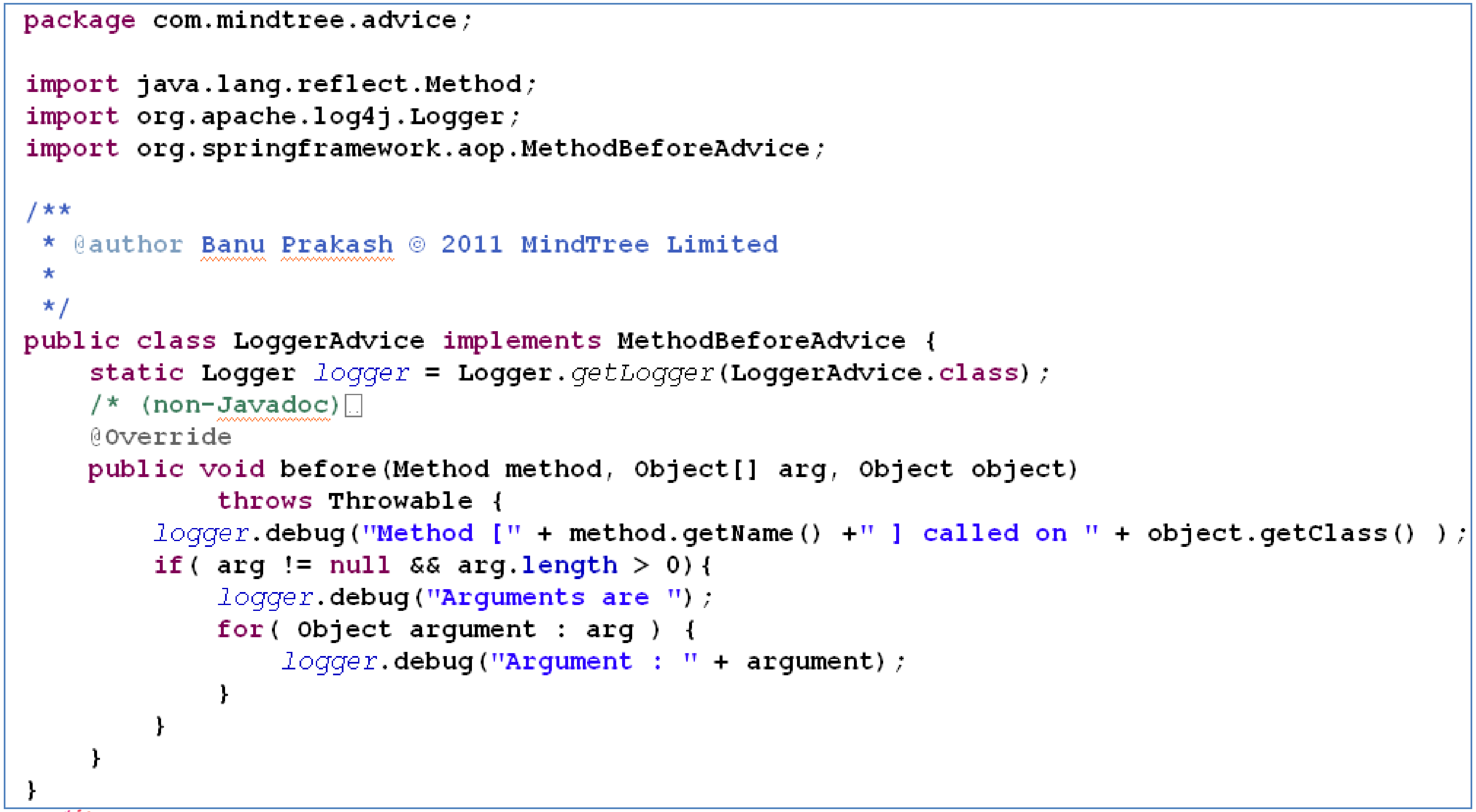
Aspect Oriented Programming

- Concerns, Advice and JoinPoint type in a simple banking application.

Concern	Advice	JoinPoint type	Description
Authenticating	BeforeAdvice	Method	Validate user
Integrity	BeforeAdvice	Method	Avoid adding duplicate items to the database
Auditing	AfterAdvice, AfterReturningAdvice	Method	Record operations performed by clerks for auditing
Logging	BeforeAdvice, AfterAdvice, AfterReturningAdvice, ThrowsAdvice	Method, Exception	Log operations performed by user and log exceptions.
Transaction	AroundAdvice	Method, Exception	Apply transaction around fund transfer method, rollback if any exception occurs
Profiling	AroundAdvice	Method	Profile how much time does it take to execute the business method

Aspect Oriented Programming

- Defining LoggerAdvice as before advice

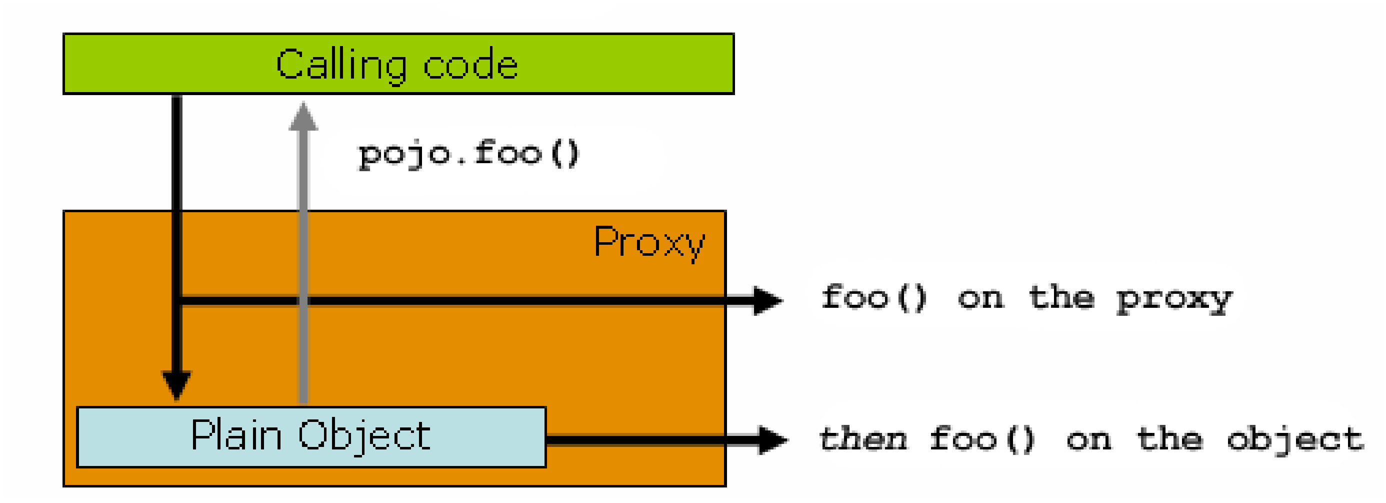


```java
package com.mindtree.advice;

import java.lang.reflect.Method;
import org.apache.log4j.Logger;
import org.springframework.aop.MethodBeforeAdvice;

/**
 * @author Banu Prakash © 2011 MindTree Limited
 *
 */
public class LoggerAdvice implements MethodBeforeAdvice {
    static Logger logger = Logger.getLogger(LoggerAdvice.class);
    /* (non-Javadoc)
    @Override
    public void before(Method method, Object[] arg, Object object)
            throws Throwable {
        logger.debug("Method [" + method.getName() +" ] called on " + object.getClass() );
        if( arg != null && arg.length > 0){
            logger.debug("Arguments are ");
            for( Object argument : arg ) {
                logger.debug("Argument : " + argument);
            }
        }
    }
}
```

Mindtree

ProxyFactoryBean

- ProxyFactoryBean is a factory bean that produces a proxy that applies one or more interceptors to a bean.



Aspect Oriented Programming

- Attach the advice to appropriate Join points.

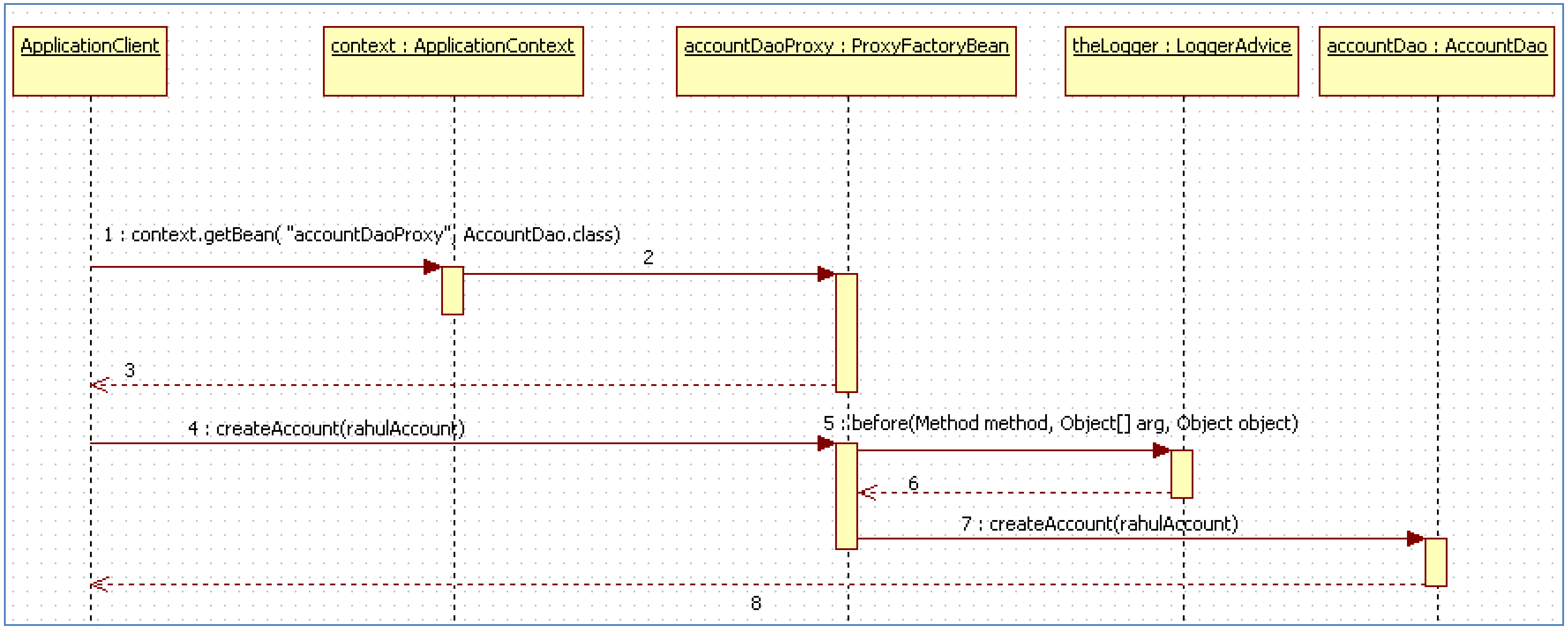
```xml
<bean id="accountDao" class="com.mindtree.dao.AccountInMemoryImpl">
</bean>
<!-- configure ProxyFactoryBean -->
<bean id="accountDaoProxy" class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="proxyInterfaces">
        <value>com.mindtree.dao.AccountDao</value>
    </property>
    <property name="target"  ref="accountDao" />
    <property name="interceptorNames">
        <list>
            <value>logAdvice</value>
        </list>
    </property>
</bean>

<!-- Advisor point cut definition for before advice -->
<bean id="logAdvice"
    class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
    <property name="advice" ref="theLogger" />
    <property name="pattern" value=".*" />
</bean>

<bean id="theLogger" class="com.mindtree.advice.LoggerAdvice" />
```

Mindtree

Aspect Oriented Programming

- Execution sequence for the code:

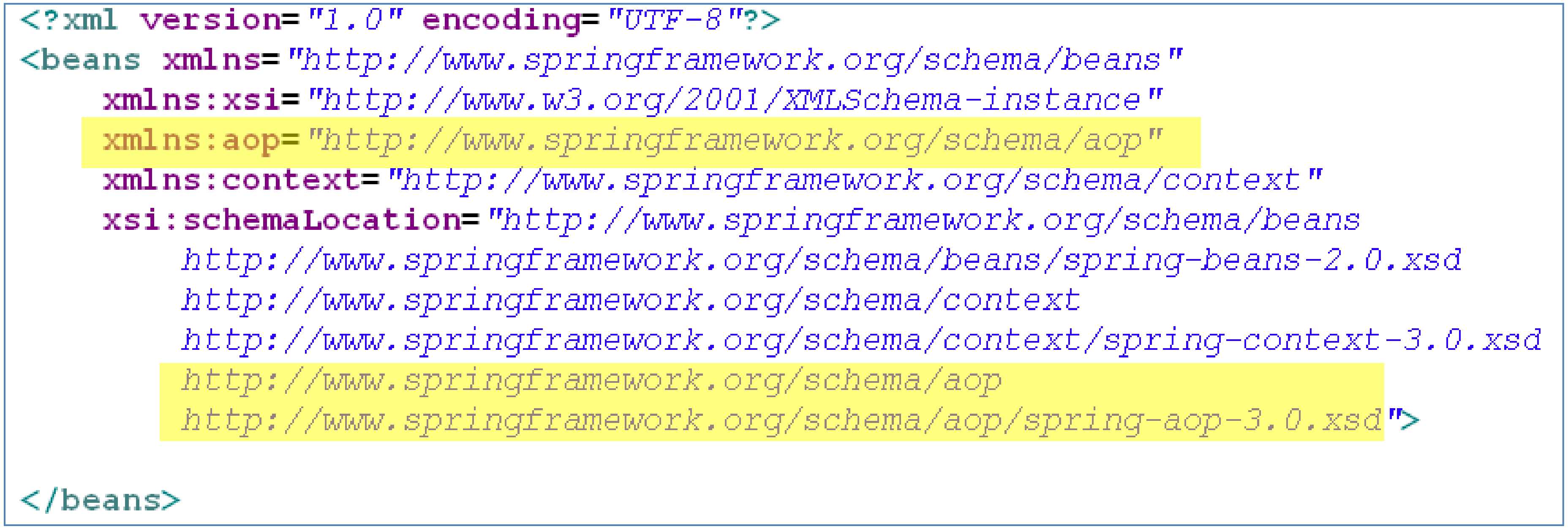
```
Account rahulAcc =
        context.getBean("rahulAccount", Account.class);


AccountDao accountDao =
        context.getBean("accountDaoProxy", AccountDao.class);
accountDao.createAccount(rahulAcc);
```


ApplicationClient
context : ApplicationContext
accountDaoProxy : ProxyFactoryBean
theLogger : LoggerAdvice
accountDao : AccountDao
1 : context.getBean("accountDaoProxy", AccountDao.class)
2
3
4 : createAccount(rahulAccount)
5 : before(Method method, Object[] arg, Object object)
6
7 : createAccount(rahulAccount)
8

Mindtree

Aspect Oriented Programming using “aop” namespace

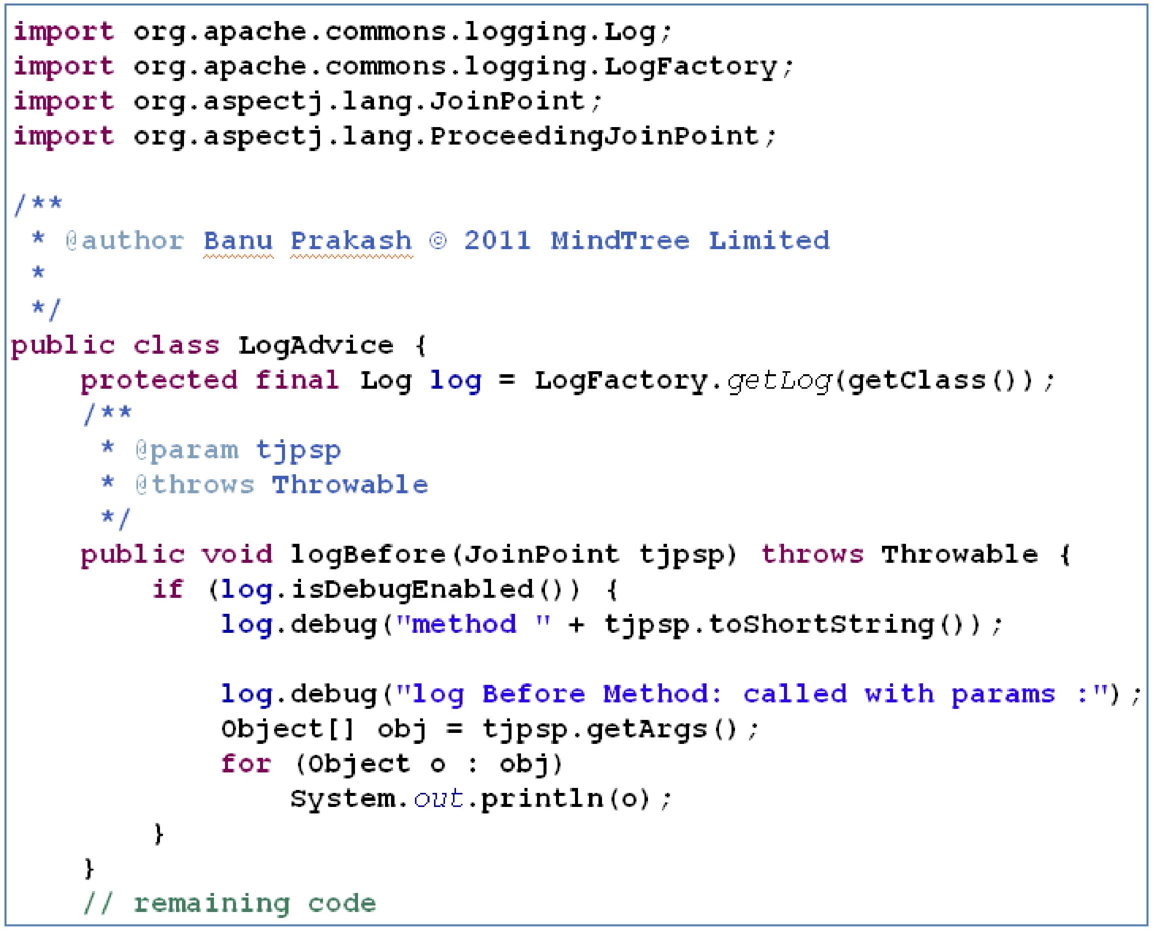
- Spring 2.0 version added support for defining aspects using the “**aop**” namespace tags.



```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"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.0.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">

</beans>
```

Aspect Oriented Programming using “aop” namespace

- **Declaring an aspect**
  - An aspect is simply a regular Java object
  - Pointcut and Advice information is captured in the XML.

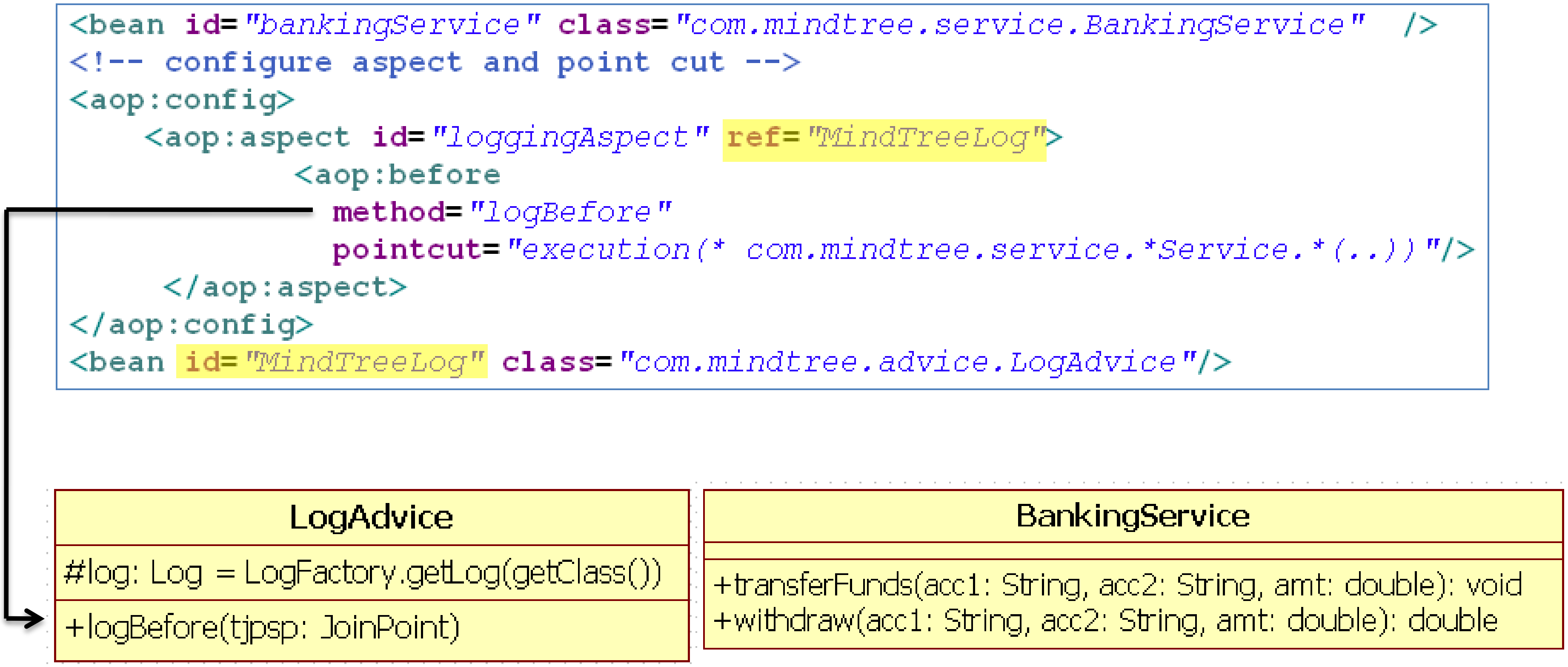


```java
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.ProceedingJoinPoint;

/**
 * @author Banu Prakash © 2011 MindTree Limited
 *
 */
public class LogAdvice {
    protected final Log log = LogFactory.getLog(getClass());
    /**
     * @param tjpsp
     * @throws Throwable
     */
    public void logBefore(JoinPoint tjpsp) throws Throwable {
        if (log.isDebugEnabled()) {
            log.debug("method " + tjpsp.toShortString());

            log.debug("log Before Method: called with params :");
            Object[] obj = tjpsp.getArgs();
            for (Object o : obj)
                System.out.println(o);
        }
    }
    // remaining code
```

Mindtree

Aspect Oriented Programming using “aop” namespace

- Configuring aspect and pointcut

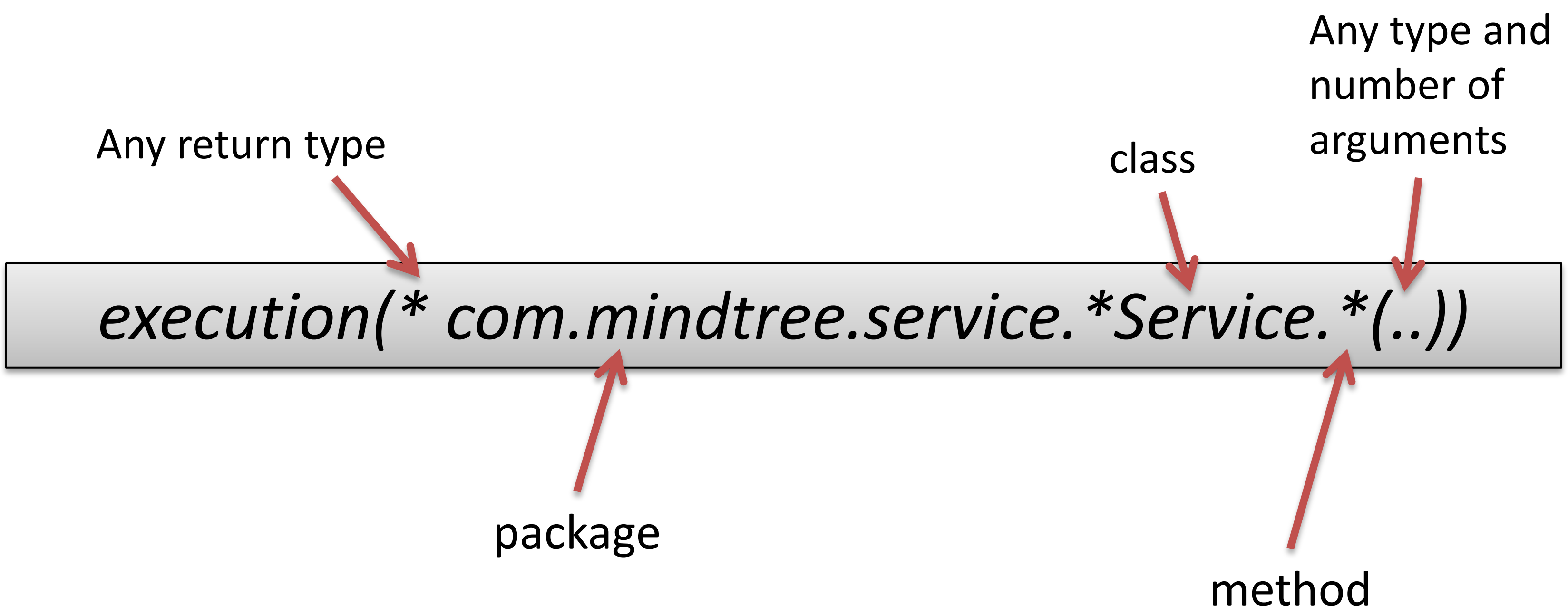


```xml
<bean id="bankingService" class="com.mindtree.service.BankingService"  />
<!-- configure aspect and point cut -->
<aop:config>
    <aop:aspect id="loggingAspect" ref="MindTreeLog">
            <aop:before
              method="logBefore"
              pointcut="execution(* com.mindtree.service.*Service.*(..))"/>
     </aop:aspect>
</aop:config>
<bean id="MindTreeLog" class="com.mindtree.advice.LogAdvice"/>
```

LogAdvice
#log: Log = LogFactory.getLog(getClass())
+logBefore(tjpsp: JoinPoint)
BankingService
+transferFunds(acc1: String, acc2: String, amt: double): void
+withdraw(acc1: String, acc2: String, amt: double): double

Aspect Oriented Programming using “aop” namespace

- PointCut

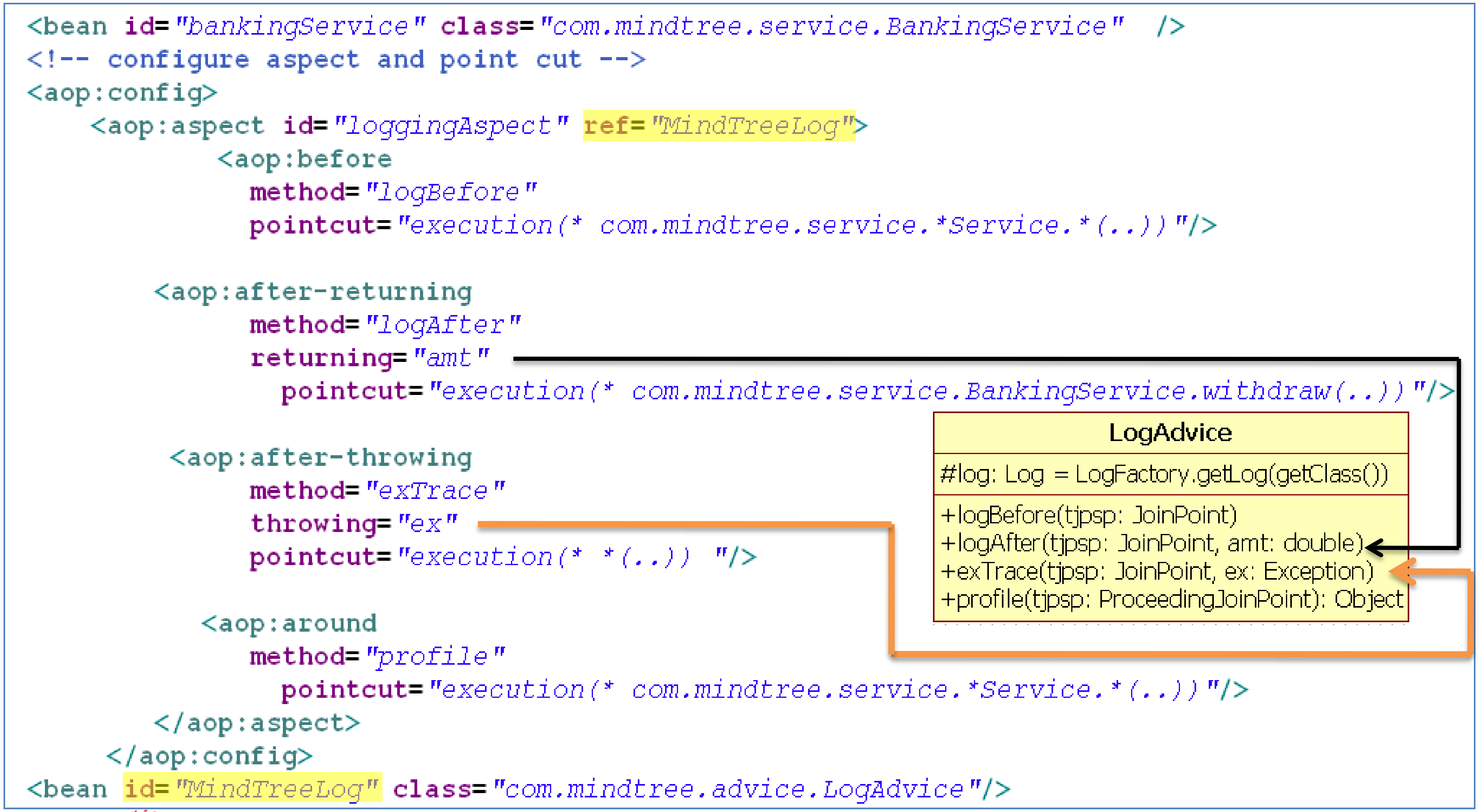


```
execution(* com.mindtree.service.*Service.*(..))
```

execution(modifiers-pattern? ret-type-pattern declaring-type-pattern? name-pattern(param-pattern) throws-pattern?)

Aspect Oriented Programming using “aop” namespace

- Configuring aspect and pointcut continued.



```xml
<bean id="bankingService" class="com.mindtree.service.BankingService"  />
<!-- configure aspect and point cut -->
<aop:config>
    <aop:aspect id="loggingAspect" ref="MindTreeLog">
            <aop:before
              method="logBefore"
              pointcut="execution(* com.mindtree.service.*Service.*(..))"/>

        <aop:after-returning
              method="logAfter"
              returning="amt"
                pointcut="execution(* com.mindtree.service.BankingService.withdraw(..))"/>

         <aop:after-throwing
              method="exTrace"
              throwing="ex"
              pointcut="execution(*  *(..))  "/>

           <aop:around
              method="profile"
                pointcut="execution(* com.mindtree.service.*Service.*(..))"/>
        </aop:aspect>
     </aop:config>
<bean id="MindTreeLog" class="com.mindtree.advice.LogAdvice"/>
```

LogAdvice
#log: Log = LogFactory.getLog(getClass())

+logBefore(tjpsp: JoinPoint)
+logAfter(tjpsp: JoinPoint, amt: double)
+exTrace(tjpsp: JoinPoint, ex: Exception)
+profile(tjpsp: ProceedingJoinPoint): Object

Welcome to possible

India | USA | UK | Germany | Sweden | Belgium | France | Switzerland | UAE | Singapore | Australia | Japan | China