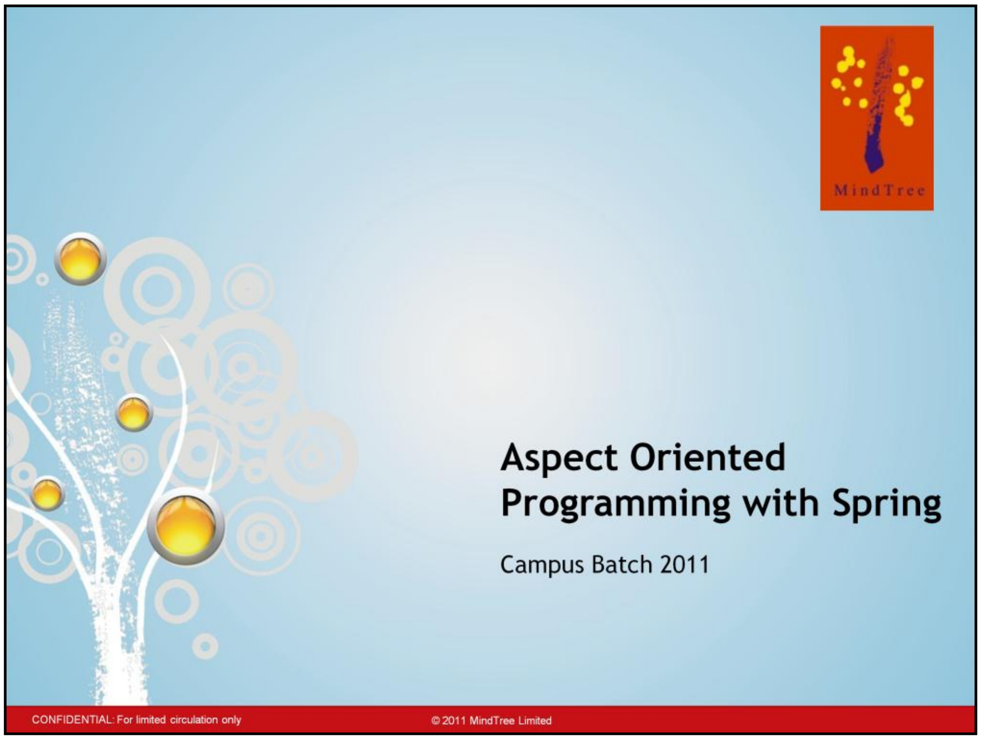
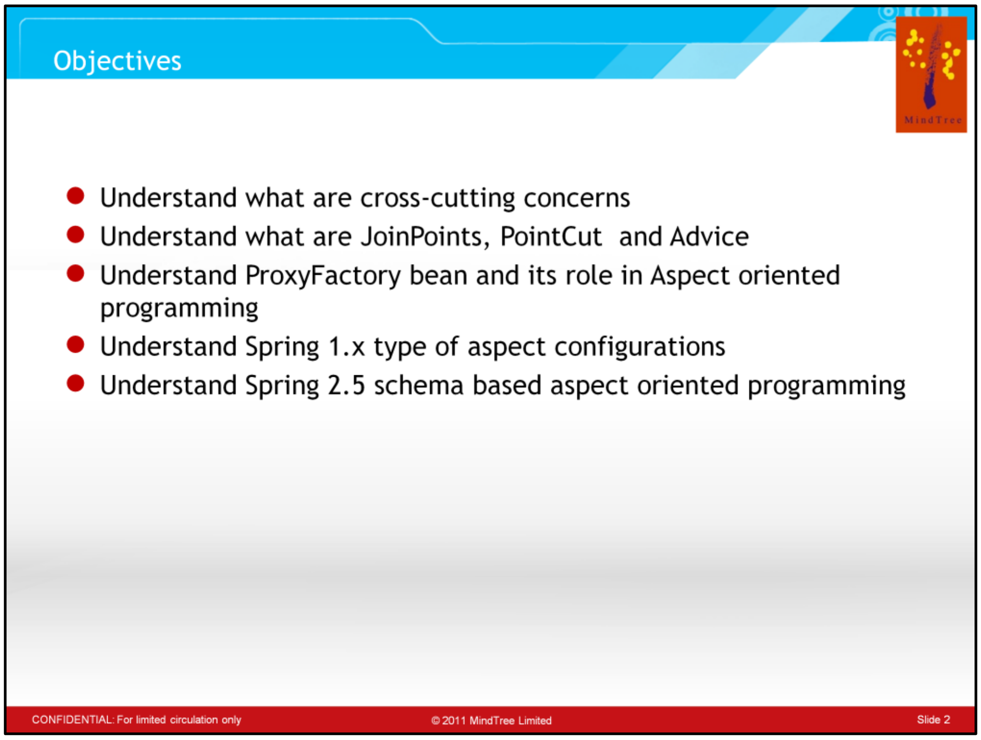


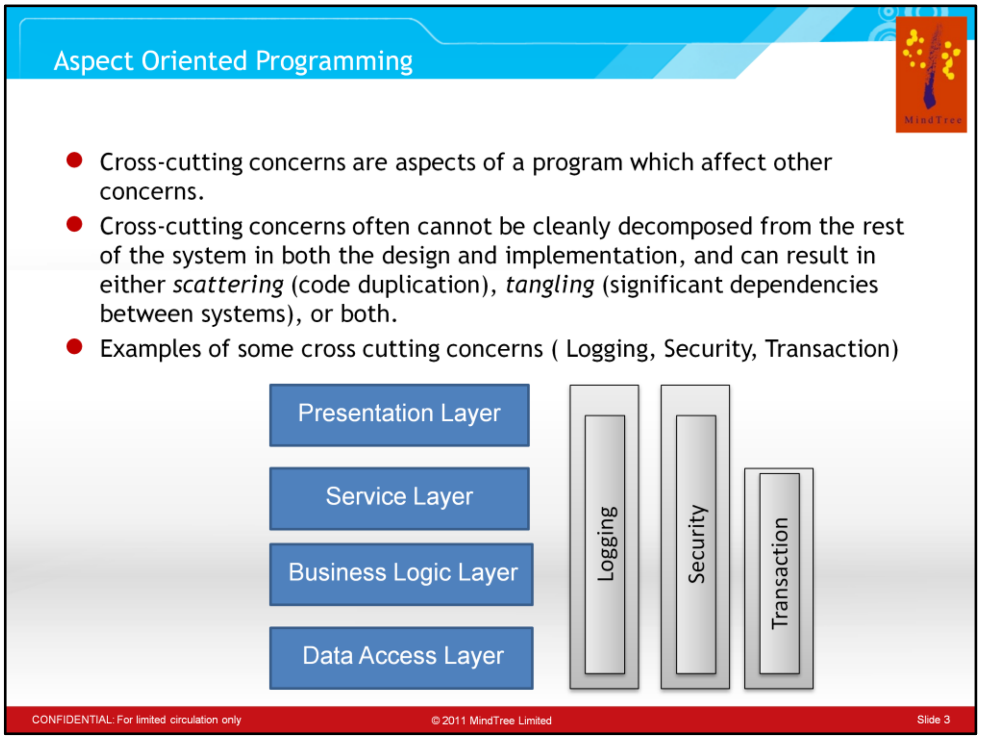
Aspect Oriented Programming with Spring

Campus Batch 2011



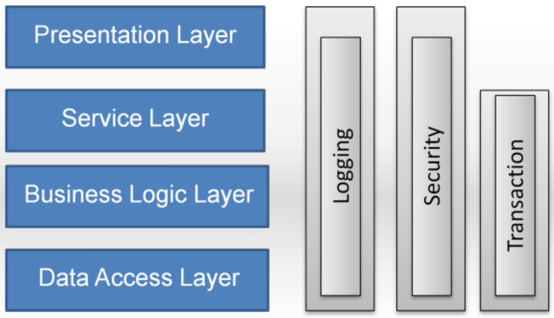
Objectives

- Understand what are cross-cutting concerns
- Understand what are JoinPoints, PointCut and Advice
- Understand ProxyFactory bean and its role in Aspect oriented programming
- Understand Spring 1.x type of aspect configurations
- Understand Spring 2.5 schema based aspect oriented programming



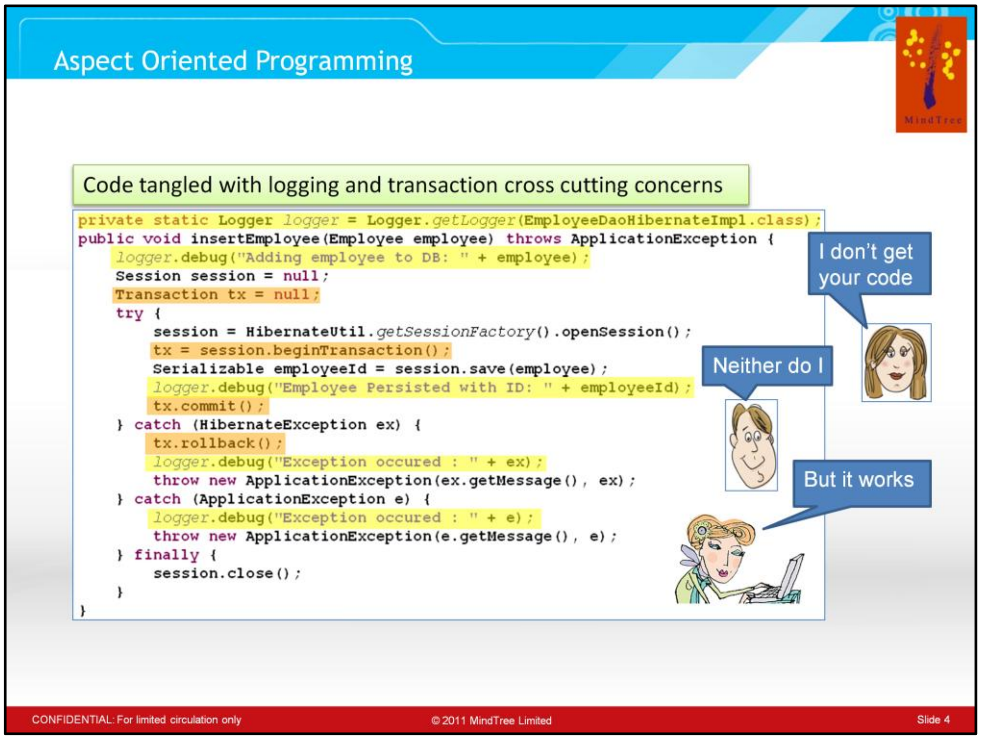
Aspect Oriented Programming

- Cross-cutting concerns are aspects of a program which affect other concerns.
- Cross-cutting concerns often cannot be cleanly decomposed from the rest of the system in both the design and implementation, and can result in either *scattering* (code duplication), *tangling* (significant dependencies between systems), or both.
- Examples of some cross cutting concerns (Logging, Security, Transaction)



CONFIDENTIAL: For limited circulation only © 2011 MindTree Limited Slide 3

Logging exemplifies a crosscutting concern because a logging strategy necessarily affects every logged part of the system. Logging thereby crosscuts all logged classes and methods. Similarly security is a cross-cutting concern in that many methods in an application can have security rules applied to them.



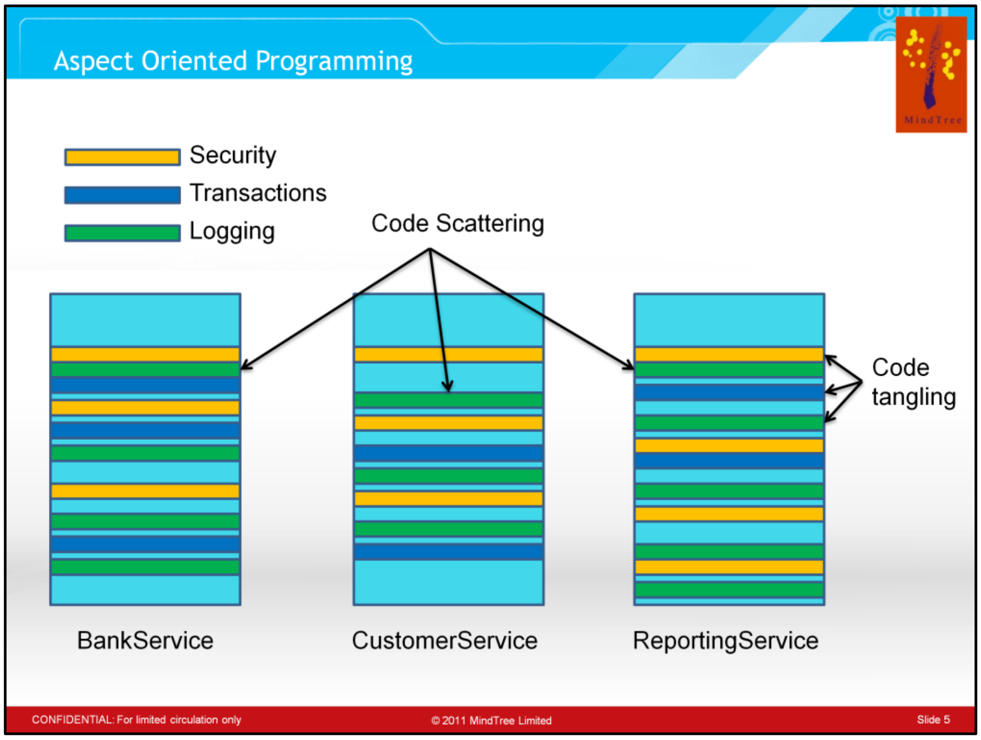
Aspect Oriented Programming

Code tangled with logging and transaction cross cutting concerns

```
private static Logger logger = Logger.getLogger(EmployeeDaoHibernateImpl.class);
public void insertEmployee(Employee employee) throws ApplicationException {
    logger.debug("Adding employee to DB: " + employee);
    Session session = null;
    Transaction tx = null;
    try {
        session = HibernateUtil.getSessionFactory().openSession();
        tx = session.beginTransaction();
        Serializable employeeId = session.save(employee);
        logger.debug("Employee Persisted with ID: " + employeeId);
        tx.commit();
    } catch (HibernateException ex) {
        tx.rollback();
        logger.debug("Exception occured : " + ex);
        throw new ApplicationException(ex.getMessage(), ex);
    } catch (ApplicationException e) {
        logger.debug("Exception occured : " + e);
        throw new ApplicationException(e.getMessage(), e);
    } finally {
        session.close();
    }
}
```

CONFIDENTIAL: For limited circulation only © 2011 MindTree Limited Slide 4

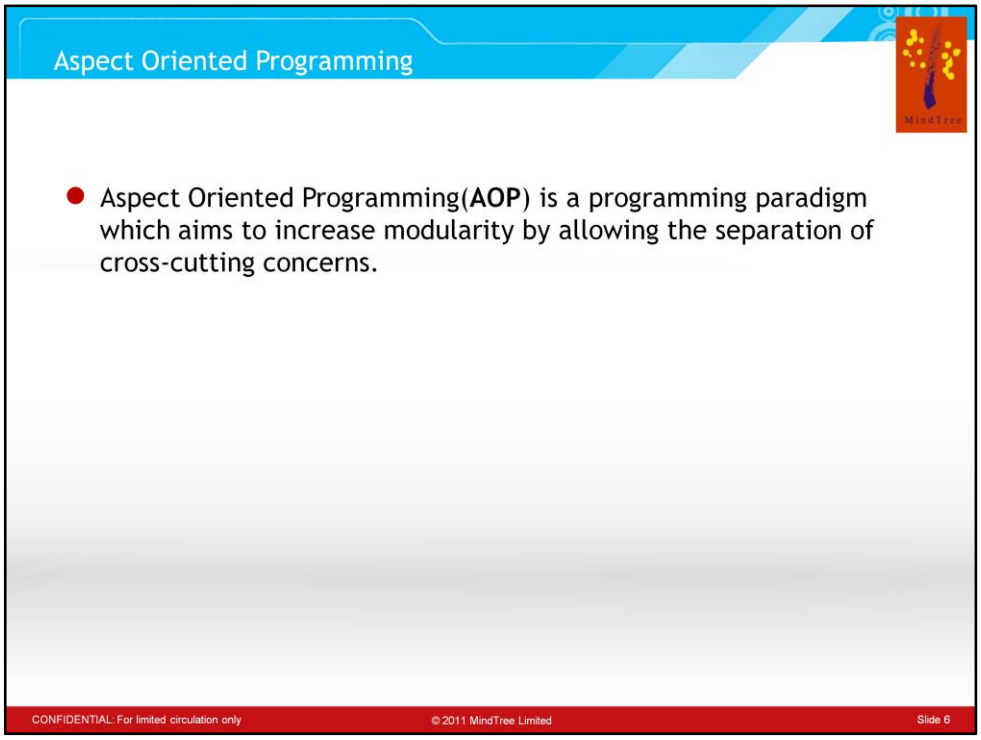
Perform Logging and apply transaction for different methods across layers itself is a sign that these are cross-cutting concerns.



Aspect Oriented Programming

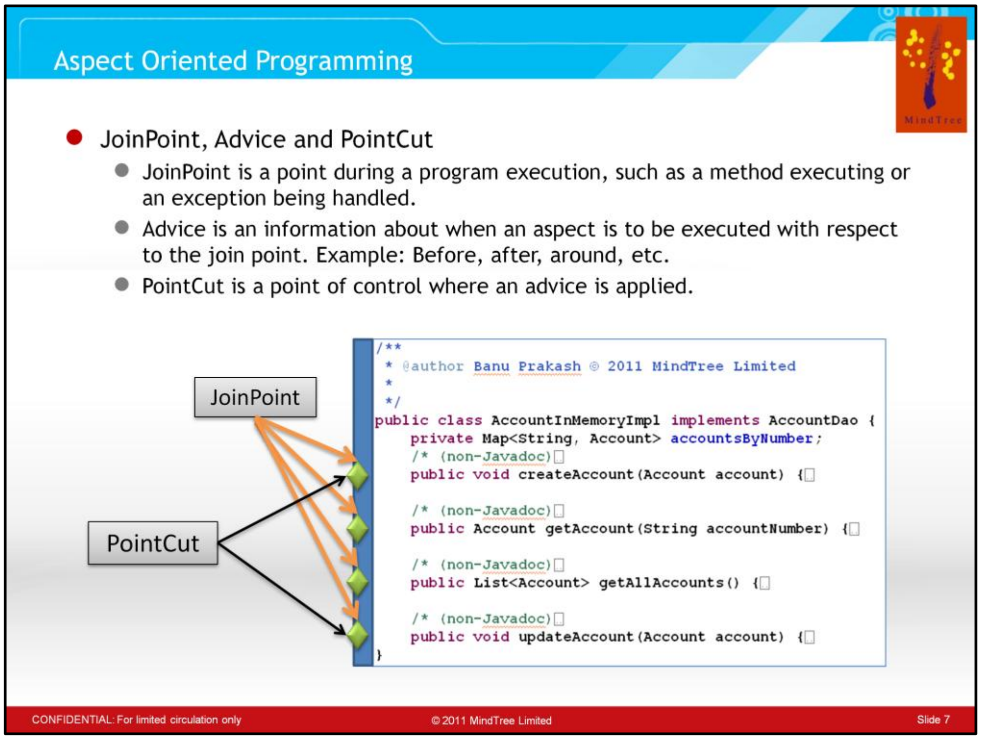
Code Scattering: The same concern spread across modules

Code Tangling: A coupling of concerns



Aspect Oriented Programming

- Aspect Oriented Programming(**AOP**) is a programming paradigm which aims to increase modularity by allowing the separation of cross-cutting concerns.



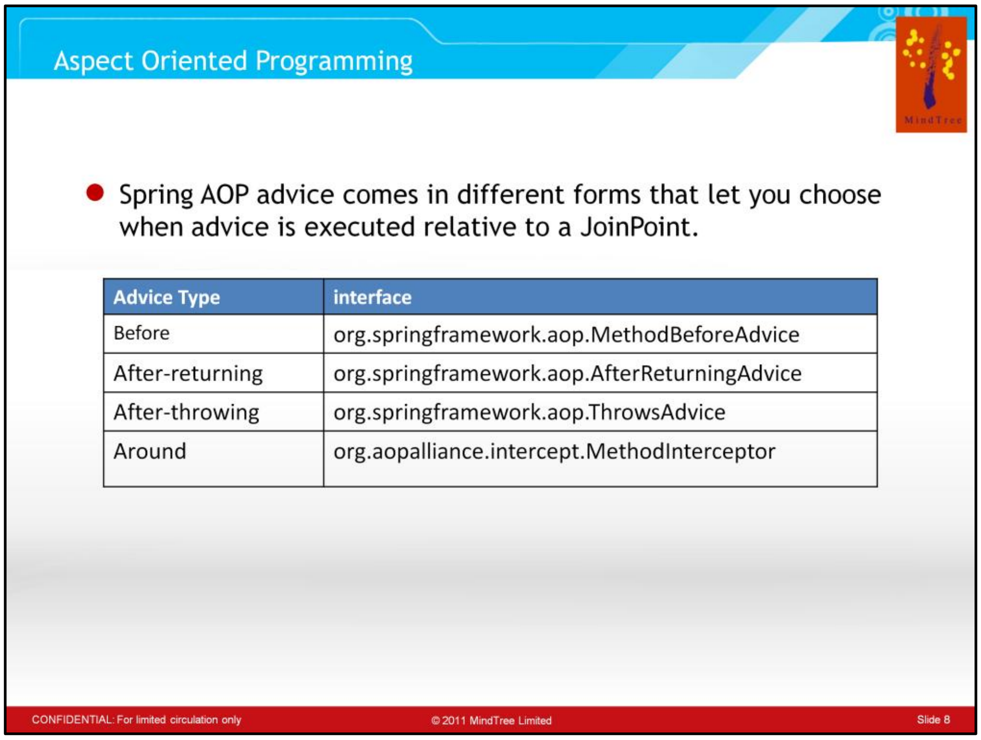
Aspect Oriented Programming

- JoinPoint, Advice and PointCut
  - JoinPoint is a point during a program execution, such as a method executing or an exception being handled.
  - Advice is an information about when an aspect is to be executed with respect to the join point. Example: Before, after, around, etc.
  - PointCut is a point of control where an advice is applied.

```
/**
 * @author Banu Prakash © 2011 MindTree Limited
 *
 */
public class AccountInMemoryImpl implements AccountDao {
    private Map<String, Account> accountsByNumber;
    /* (non-Javadoc)
    public void createAccount(Account account) {

    /* (non-Javadoc)
    public Account getAccount(String accountNumber) {

    /* (non-Javadoc)
    public List<Account> getAllAccounts() {

    /* (non-Javadoc)
    public void updateAccount(Account account) {
}
```

CONFIDENTIAL: For limited circulation only © 2011 MindTree Limited Slide 7

In this case we have selected only two JointPoints (updateAccount() and createAccount() methods) to apply an advice.



Aspect Oriented Programming

- Spring AOP advice comes in different forms that let you choose when advice is executed relative to a JoinPoint.

Advice Type	interface
Before	org.springframework.aop.MethodBeforeAdvice
After-returning	org.springframework.aop.AfterReturningAdvice
After-throwing	org.springframework.aop.ThrowsAdvice
Around	org.aopalliance.intercept.MethodInterceptor

CONFIDENTIAL: For limited circulation only © 2011 MindTree Limited Slide 8

Before advice : This interface requires that method public void before(Method method, Object[] args, Object target)

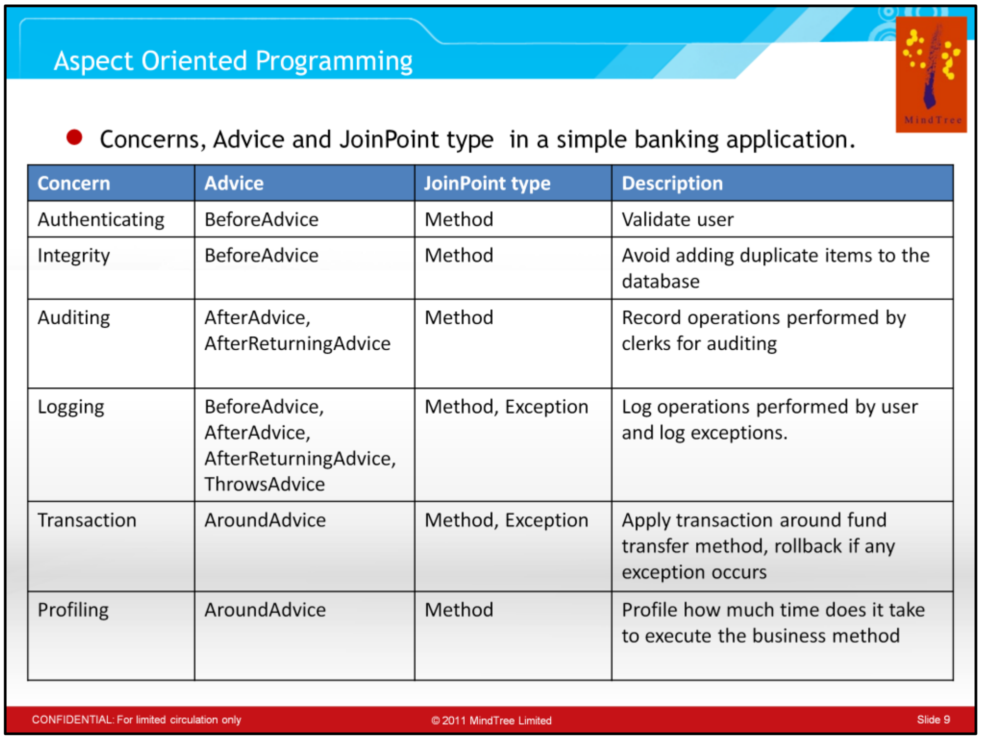
be implemented.

After returning advice : public void afterReturning(Object returnValue, Method method,

Object[] args, Object target) throws Throwable.

After throwing advice: public void afterThrowing(Method method, Object[] args, Object target, Throwable e).

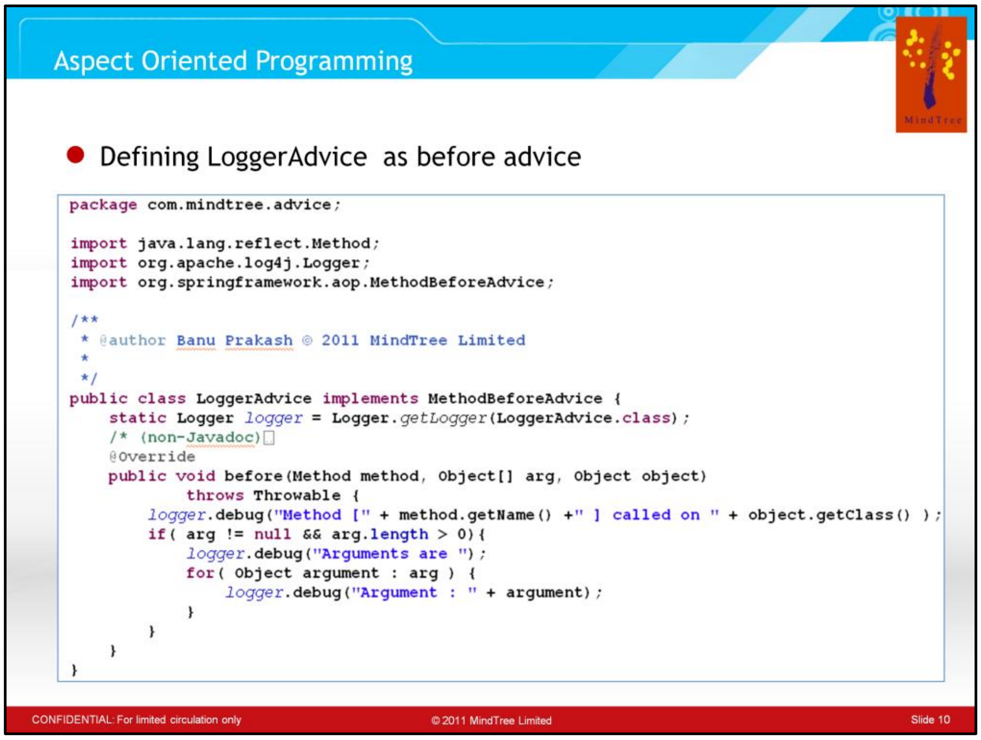
Around advice : public Object invoke(MethodInvocation invocation) throws Throwable.



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



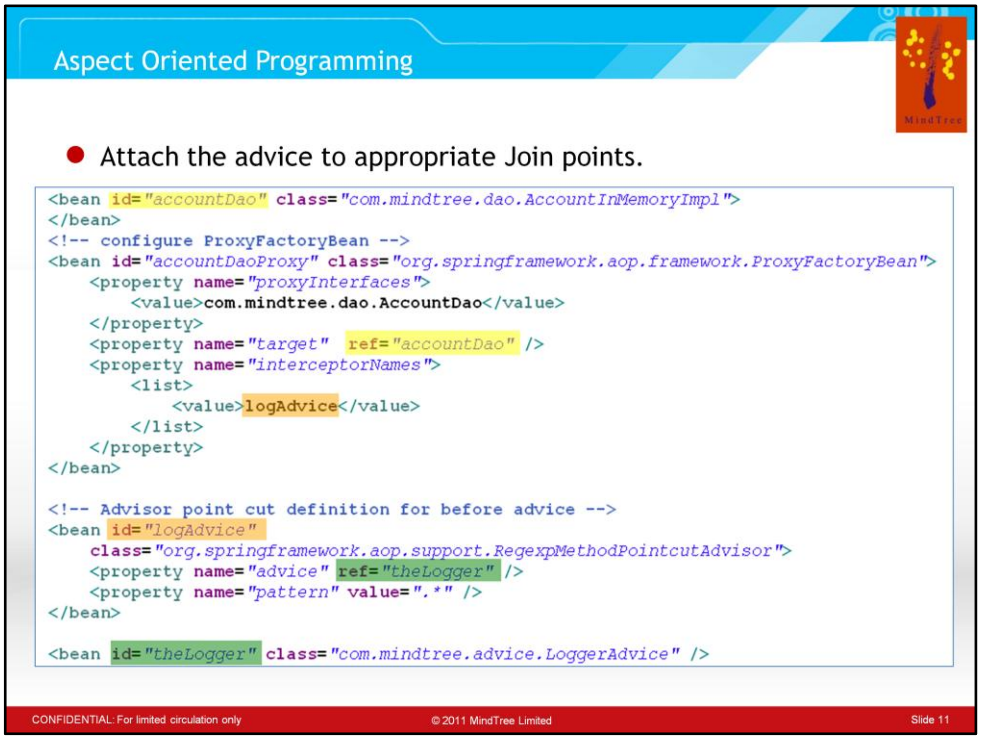
Aspect Oriented Programming

- Defining LoggerAdvice as before advice

```
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);
            }
        }
    }
}
```

CONFIDENTIAL: For limited circulation only © 2011 MindTree Limited Slide 10

LoggerAdvice aspect captures calls to methods being traced within a target application and displays this information using Logger.



Aspect Oriented Programming

- Attach the advice to appropriate Join points.

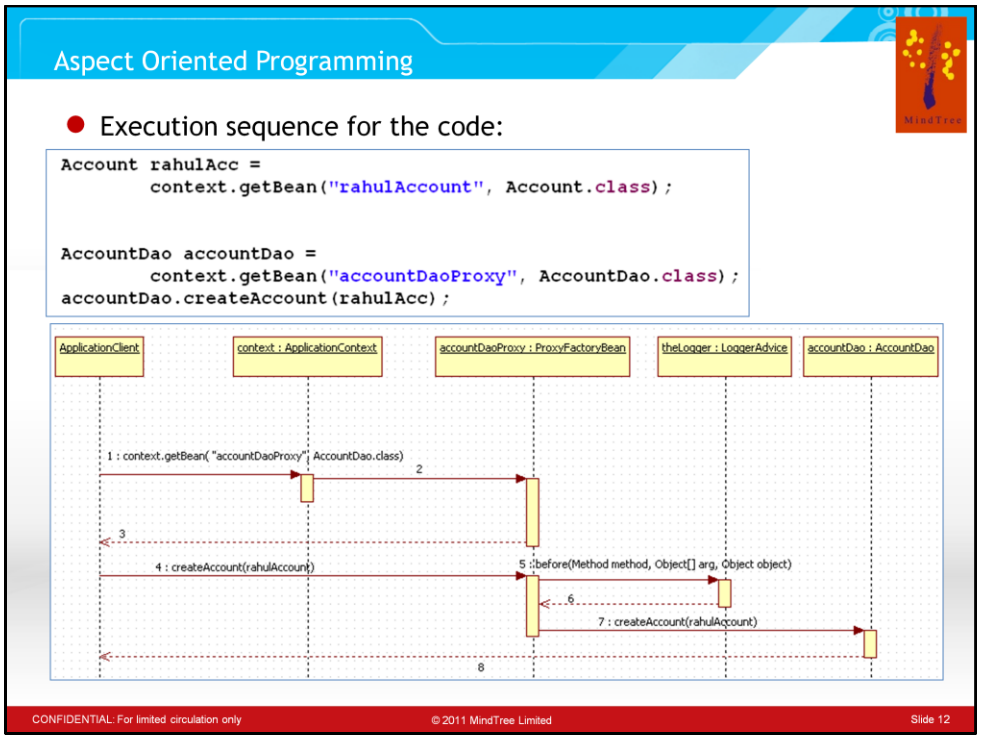
```
<bean id="accountDao" class="com.mindtree.dao.AccountInMemoryImpl">
</bean>
<!-- configure ProxyFactoryBean -->
<bean id="accountDaoProxy" class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="proxyInterfaces">
        <value>com.mindtree.dao.AccountDao</value>
    </property>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

CONFIDENTIAL: For limited circulation only © 2011 MindTree Limited Slide 11

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

The application code will call getBean() on bean with id=*"accountDaoProxy" and not id="accountDao".*

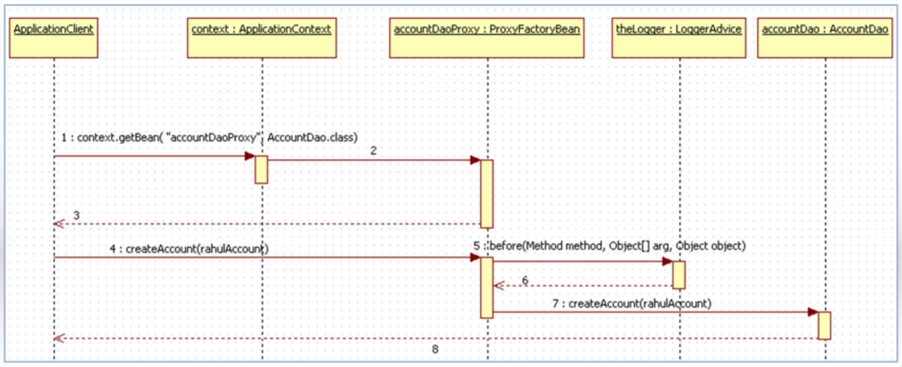
The proxy will use the pointcut to decide whether advice should be applied (or not), and then it invokes the advised bean itself. In our example when we invoke any methods of AccountDao interface using ProxyFactoryBean, it first invokes the before() method of LoggerAdvice before it invokes the actual method of AccountDaoInMemoryImpl.



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

Output is Listed for the following code:

```
 ApplicationContext context  =
  new ClassPathXmlApplicationContext("beans.xml");

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


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

 List<Account> accounts = accountDao.getAllAccounts();
  for(Account account : accounts) {
                System.out.println(account);
  }
```

Output :

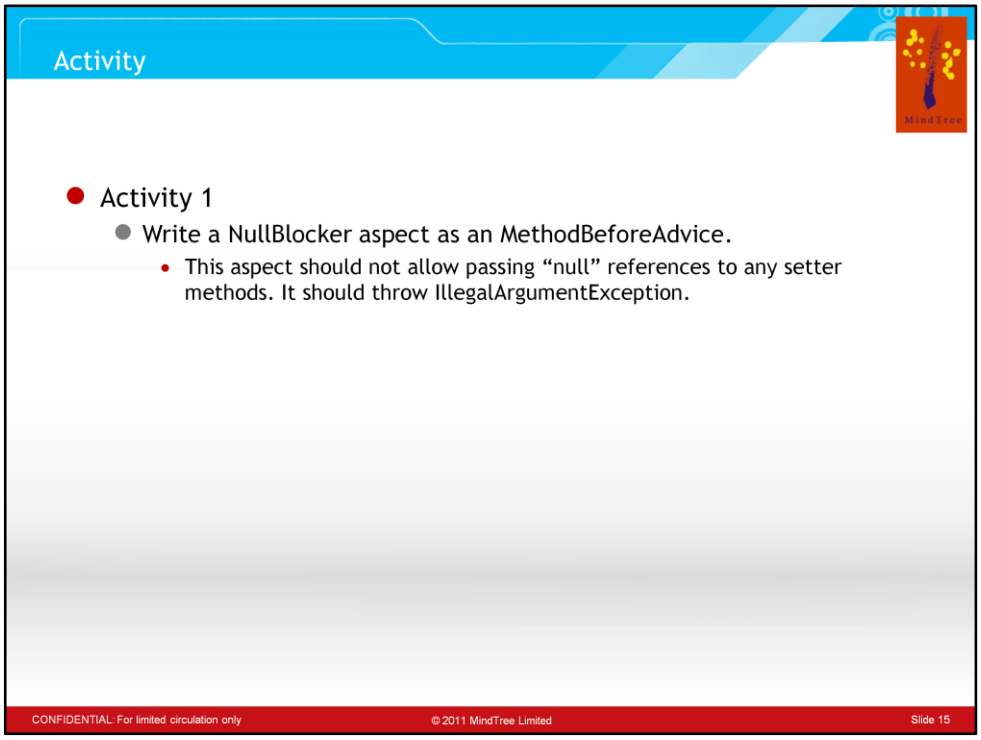
```
DEBUG [com.mindtree.advice.LoggerAdvice] - Method [createAccount ] called on class com.mindtree.dao.AccountInMemoryImpl
DEBUG [com.mindtree.advice.LoggerAdvice] - Arguments are
DEBUG [com.mindtree.advice.LoggerAdvice] - Argument : Account [SB500, Rahul B Prakash, 8590.5]
DEBUG [com.mindtree.advice.LoggerAdvice] - Method [getAllAccounts ] called on class com.mindtree.dao.AccountInMemoryImpl

Account [SB500, Rahul B Prakash, 8090.5]
```

Example

- Code Example:
  - Spring_AOP_1.zip
    - Illustrates Spring 1.x style of configuration of different types of advice.

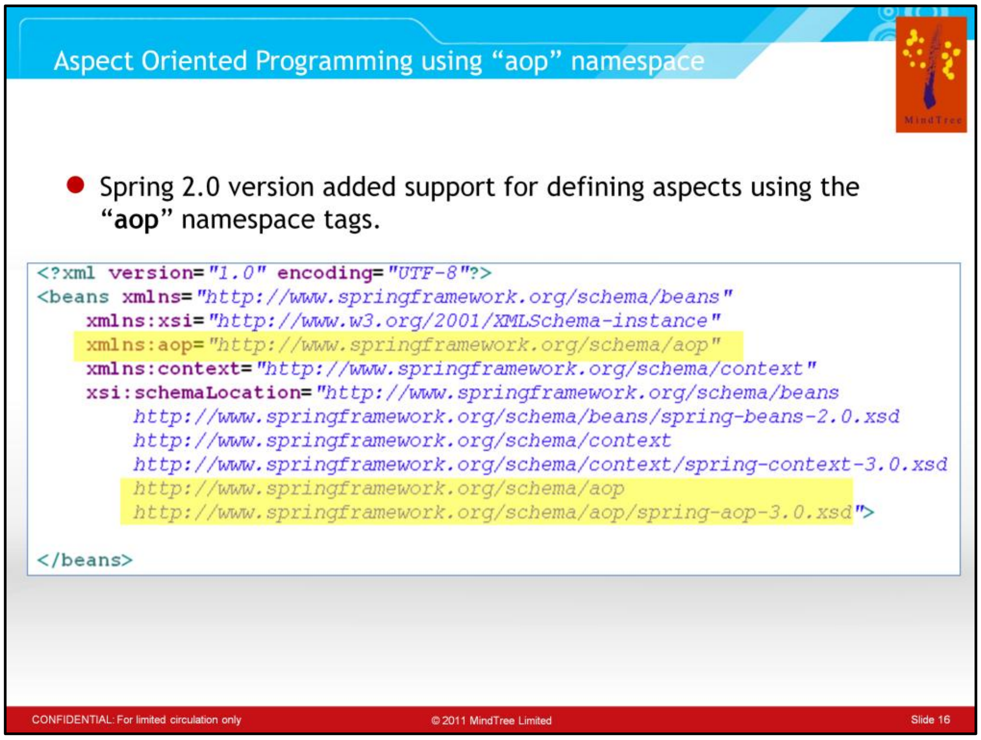
Activity

Things to do



Activity

- Activity 1
  - Write a NullBlocker aspect as an MethodBeforeAdvice.
    - This aspect should not allow passing “null” references to any setter methods. It should throw IllegalArgumentException.



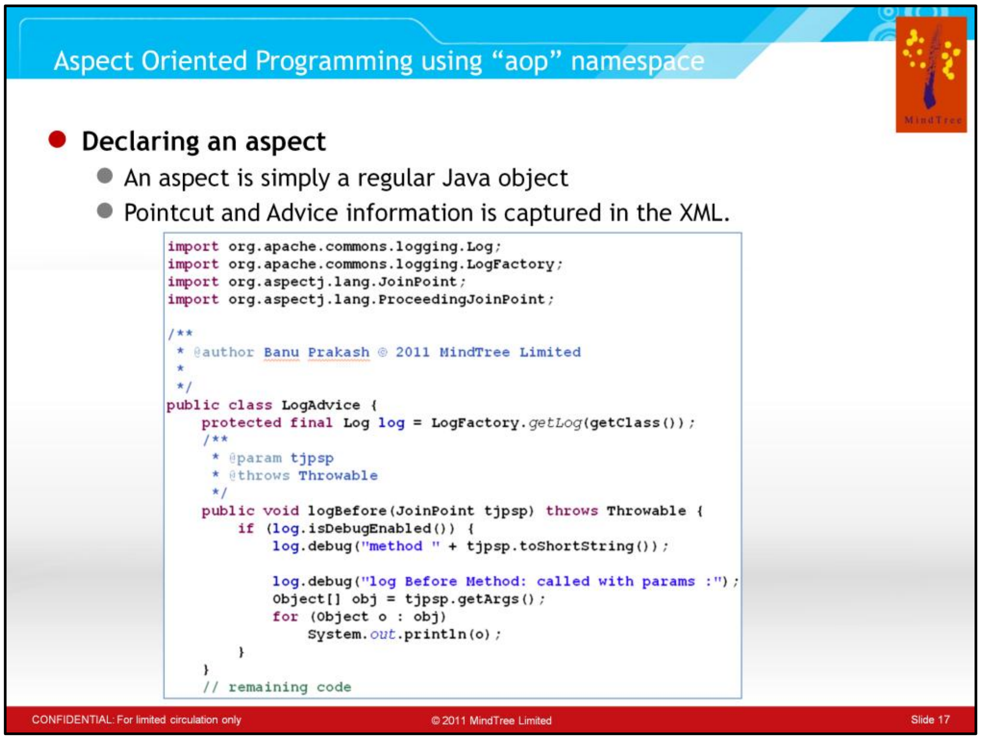
Aspect Oriented Programming using “aop” namespace

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

```
<?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>
```

CONFIDENTIAL: For limited circulation only © 2011 MindTree Limited Slide 16

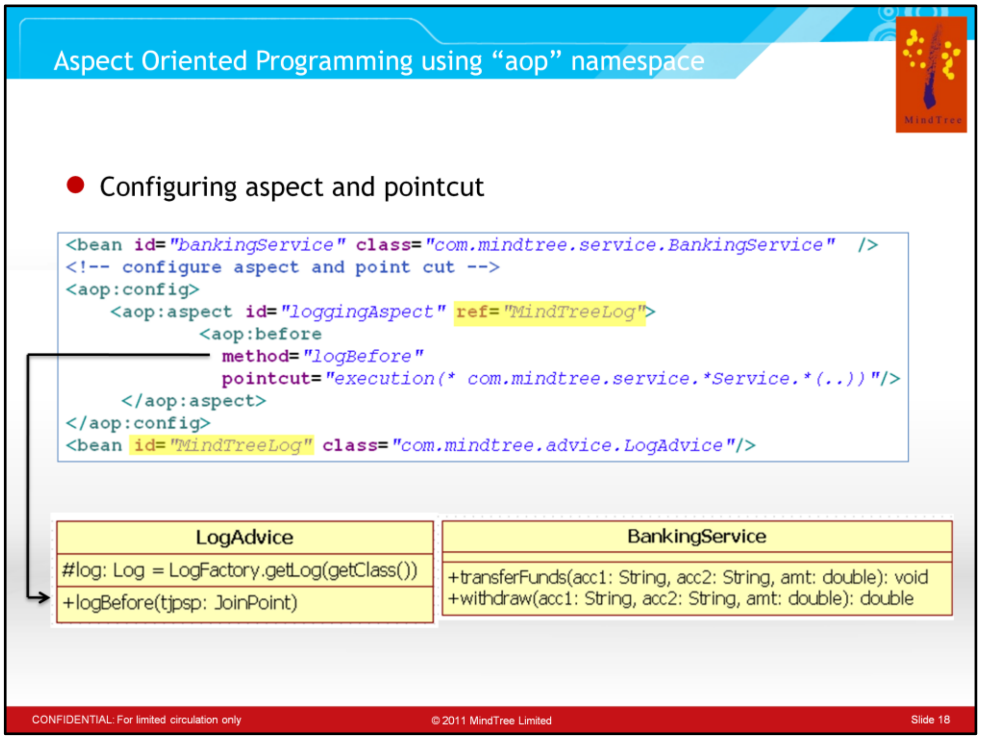
The Spring development team recognized that using ProxyFactoryBean is somewhat clumsy. So, they set out to provide a better way of declaring aspects in Spring. The outcome of this effort is found in the new XML configuration elements that come with Spring 2.0.



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
```



Aspect Oriented Programming using "aop" namespace

- Configuring aspect and pointcut

```
<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

Explanation:

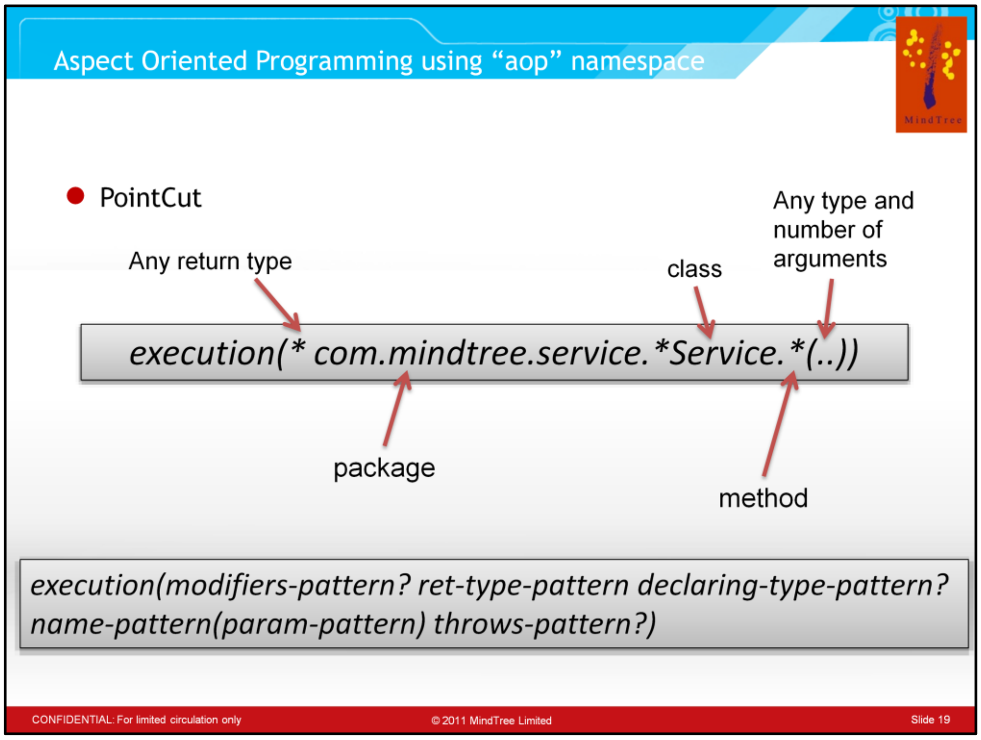
Before advice :

Before advice runs before a matched method execution. It is declared inside an <aop:aspect> using the <aop:before> element.

```
<aop:before
    method="logBefore"
    pointcut="execution(* com.mindtree.service.*Service.*(..))"/>
```

1) The method attribute identifies a method (logBefore) that provides the body of the advice.

2) pointcut="execution(* com.mindtree.service.*Service.*(..))" identifies that the logBefore advice is applicable to all methods of class ending with "Service" ,present in c"om.mindtree.service package" and taking any number and type of argument.



Aspect Oriented Programming using “aop” namespace

- PointCut

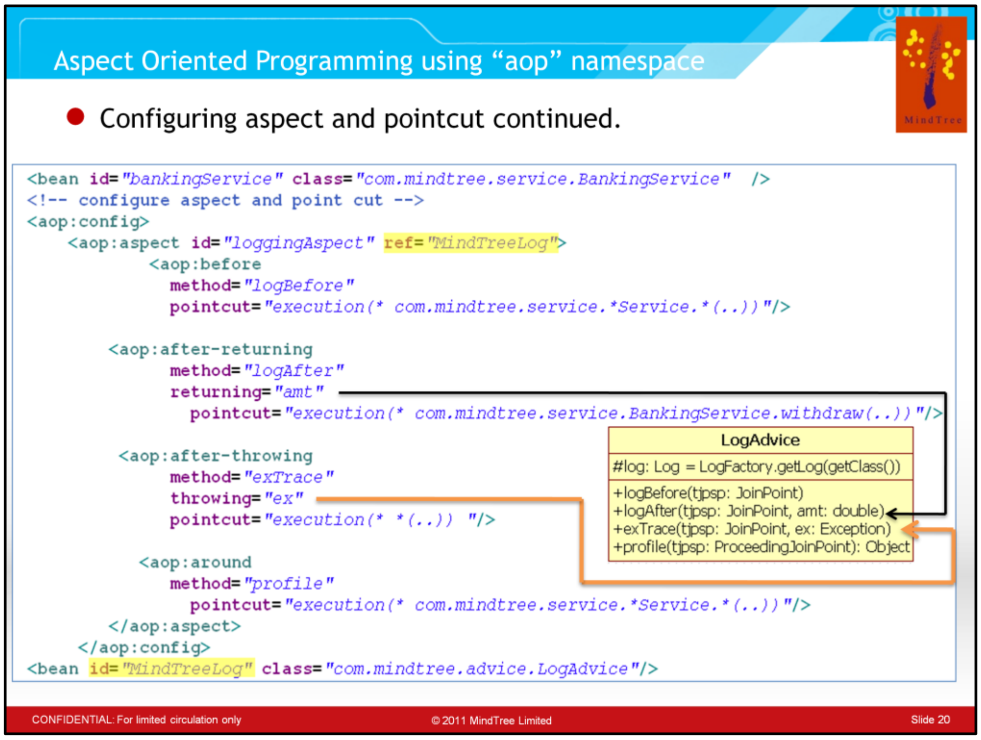
execution(com.mindtree.service.*Service.*(..))*

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

CONFIDENTIAL: For limited circulation only © 2011 MindTree Limited Slide 19

Some examples of common pointcut expressions:

1) the execution of any public method: execution(**public * *(..))**
2) the execution of any method with a name beginning with "set“: execution(* set*(..))
3) the execution of any method defined by the AccountService interface: execution(* com.mindtree.service.AccountService.*(..))
4) the execution of any method defined in the service package: execution(* com.mindtree.service.*.*(..))
5) the execution of any method defined in the service package or a sub-package: execution(* com.mindtree.service..*.*(..))



Aspect Oriented Programming using "aop" namespace

- Configuring aspect and pointcut continued.

```
<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())

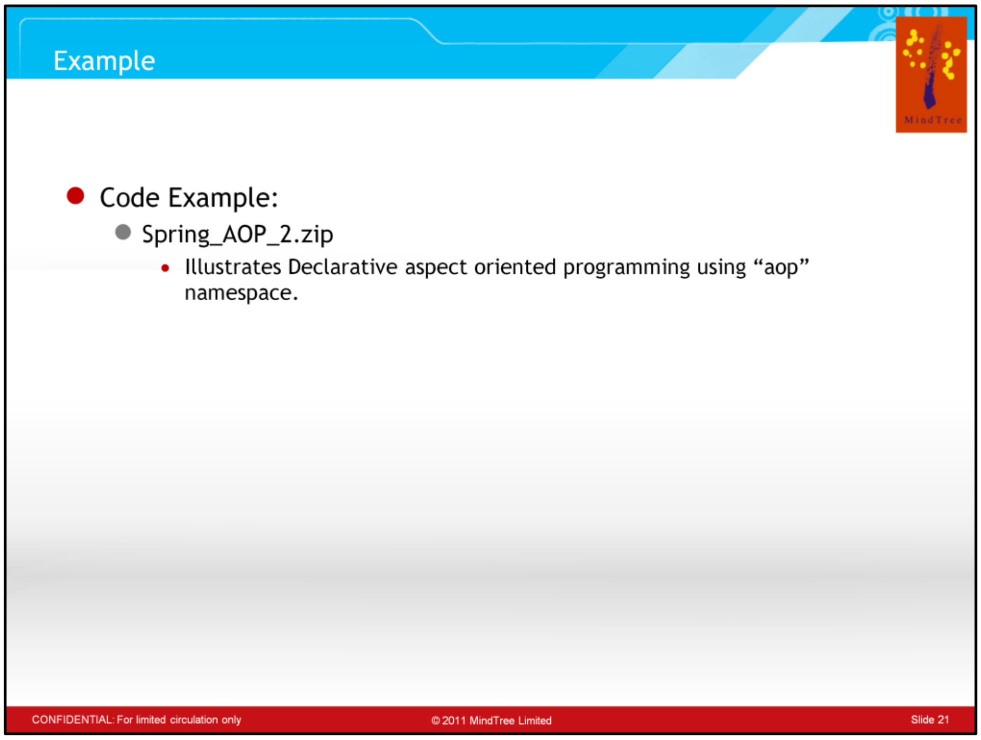
+logBefore(tjpsp: JoinPoint)
+logAfter(tjpsp: JoinPoint, amt: double)
+exTrace(tjpsp: JoinPoint, ex: Exception)
+profile(tjpsp: ProceedingJoinPoint): Object

CONFIDENTIAL: For limited circulation only © 2011 MindTree Limited Slide 20

Explanation:

After-returning advice : AfterReturning advice runs after returning from a matched method execution if no exception occurs. The return value (amt) from withdraw() method of BankingService class will be passed as an argument (amt) to logAfter() of advice (LogAdvice).

After-throwing advice: AfterThrowing advice runs if any exceptions are propagated from a matched method execution. The exception propagated from the method will be passed as an argument (ex) to exTrace() of advice LogAdvice.

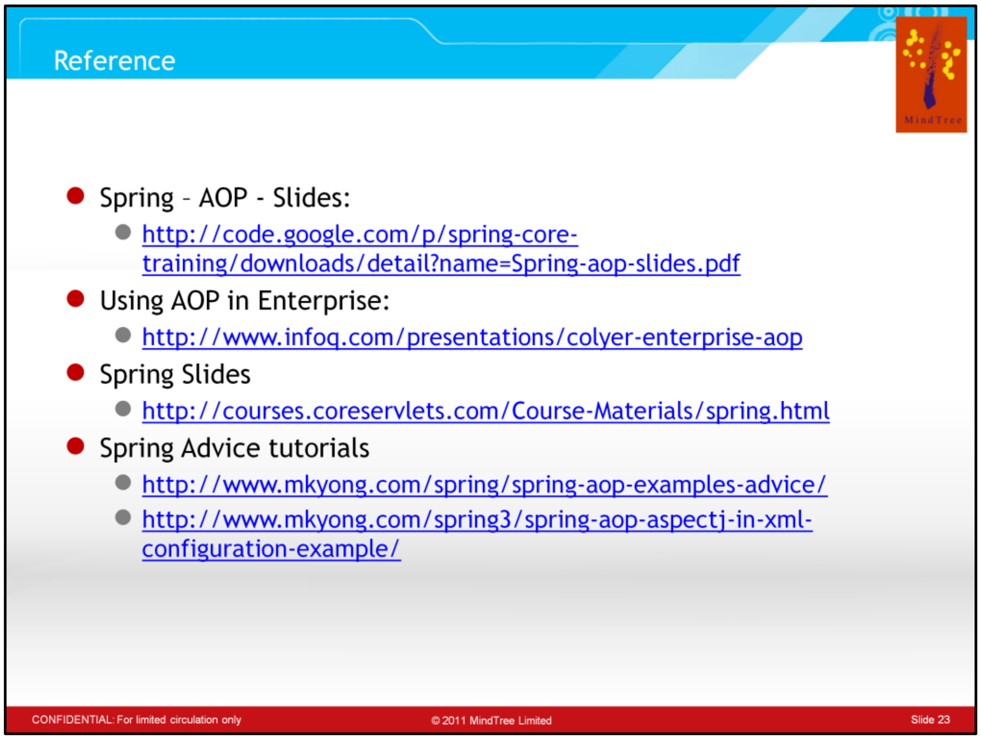


Example

- Code Example:
  - Spring_AOP_2.zip
    - Illustrates Declarative aspect oriented programming using “aop” namespace.

References

Contains the reference that will supplement the self learning and will be needed for completing the assignments & practice questions



Reference

- Spring – AOP - Slides:
  - http://code.google.com/p/spring-core-training/downloads/detail?name=Spring-aop-slides.pdf
- Using AOP in Enterprise:
  - http://www.infoq.com/presentations/colyer-enterprise-aop
- Spring Slides
  - http://courses.coreservlets.com/Course-Materials/spring.html
- Spring Advice tutorials
  - http://www.mkyong.com/spring/spring-aop-examples-advice/
  - http://www.mkyong.com/spring3/spring-aop-aspectj-in-xml-configuration-example/

Our Mission

Successful Customers

Happy People

Innovative Solutions

Contact Person

Contact_ contact@mindtree.com

+Country code-Phone

www.mindtree.com