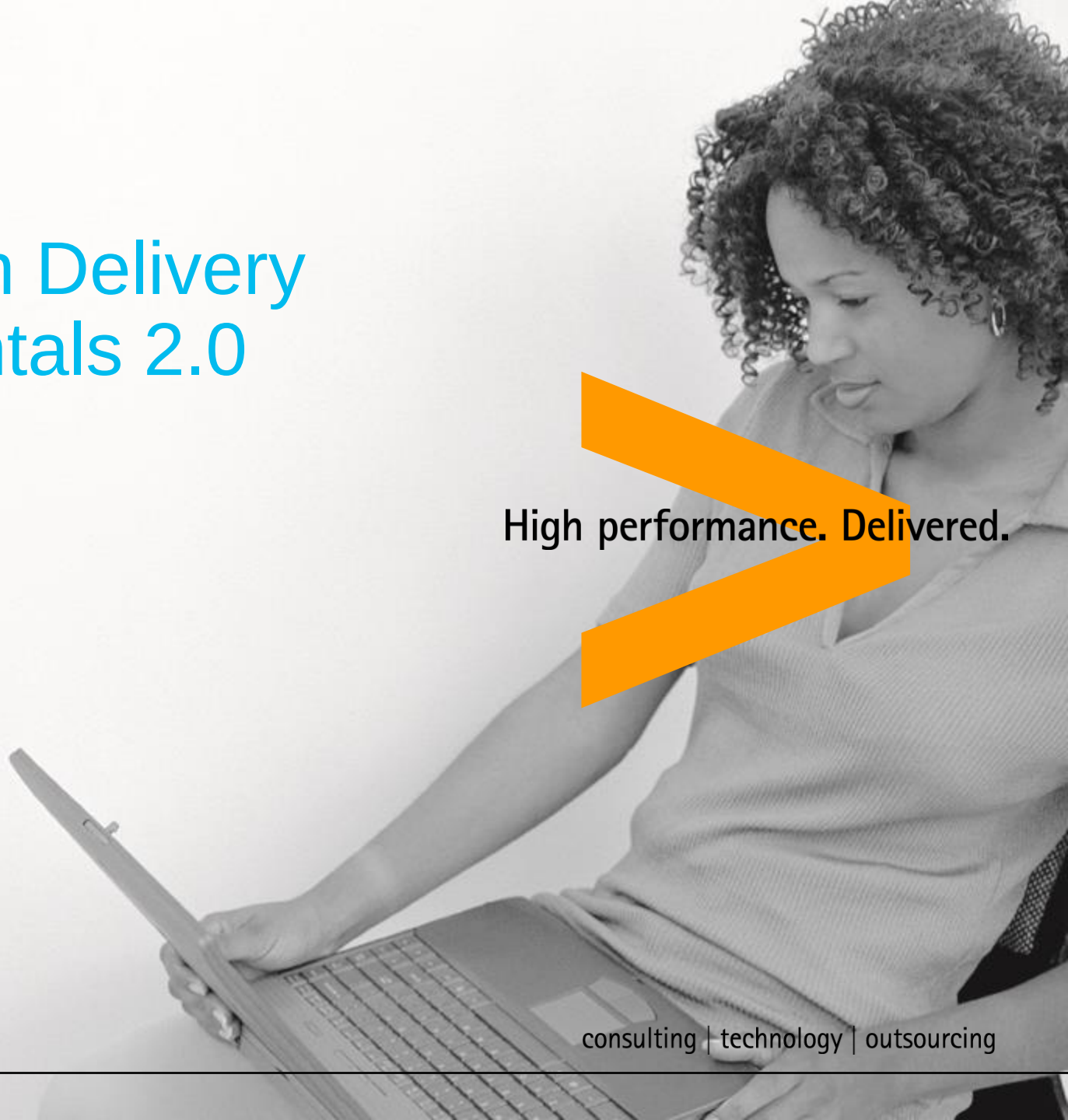
Application Delivery Fundamentals 2.0

Spring AOP



Module Objectives

- At the end of this module, participants will be able to:
  - Describe fundamentals of AOP and AOP concepts.
  - Describe how various cross-cutting concerns can be separated using Spring with AOP.
  - Define how to implement AOP concepts in Spring Framework.

AOP Fundamentals – Overview

- Aspect-oriented programming:
  - Programming methodology that facilitates modularization of cross-cutting concerns
  - Based on identifying and creating aspects
    - Building block of AOP is "*Aspect*"
  - Complements object-oriented programming
    - Building block of OOP is "*Object*"

AOP Fundamentals – Frameworks

- Aspect-oriented programming frameworks in Java:
  - AspectJ
  - Spring
  - JBossAOP
  - AspectWerkz
  - Java Aspect Components (JAC)

AOP Fundamentals (1 of 7)

- Core concern:
  - Distinct feature or behavior of a program which is modular/independent of other features
  - Core Business Logic

- Cross-cutting concerns:
  - Concerns that cannot be cleanly decomposed from the other concerns in the object-oriented programs
  - A concern that spans across multiple modules
  - Not core business logic/concern but something that is needed for other purposes

AOP Fundamentals (2 of 7)

- Common cross-cutting concerns:
  - Logging
  - Security
  - Caching
  - Transaction management
  - Error handling
  - Performance monitoring

AOP Fundamentals (3 of 7)

- Problem area: interspersed code
  - Code scattering and code duplication
  - Same concern used in many places of the application
  - Example: Let's consider a requirement for logging the entrance and exit of a method.
    - The same logger statements are repeated at the start and end of each method.
    - The same logger class and statements are used in all the methods across all modules in the application.

AOP Fundamentals (4 of 7)

- Sample for demonstrating interspersed code

```
public class CourseRegistrationService
{
// properties, constructors and other methods…..

Logger myLogger;
public int registerStudentIntoCourse(long
                        studentId, String courseId) {

myLogger.log("Before registering "+ studentId
          +"into Course: " + courseId);

// do some core business logic here…..

 myLogger.log("After registering "+ studentId
                                  +"into Course: " + courseId);

                        return 0;
}
```

Logger object declaration

Multiple logger statements – code duplication and scattering

AOP Fundamentals (5 of 7)

- Sample for demonstrating tangled code

```
public class CourseRegistrationService
{
    // properties, constructors and other methods…..
    protected SecurityService authServ = new SecurityService();
    public void registerStudentIntoCourse(long studentId, String courseId)
    {
        try{
            if(authServ.verifyAccess(studentId)){
                // do some core business logic here…..
            }
            else{
                throw new ServiceDeniedException();
            }
        }
        catch(ServiceDeniedException sde)
        {
            //do something…
        }
    }
}
```

SecurityService class provides methods for verifying access.

Encapsulates the “security” concern logic and shields the complexity of security implementation from the rest of the application code

Mixing of multiple concerns:
i. Core business logic
ii. Security
iii. Exception handling

ServiceDeniedException class provides exception handling functions.

AOP Fundamentals (6 of 7)

- Issues with these problem areas:
  - Intermixed cross-cutting concerns code with application logic code is:
    - Hard to maintain and debug
    - Prone to more errors
    - Increased dependency between modules – highly/tightly coupled
    - Changes in the interface or implementation of one of these concerns can lead to rewriting or changing code in multiple source files

AOP Fundamentals (7 of 7)

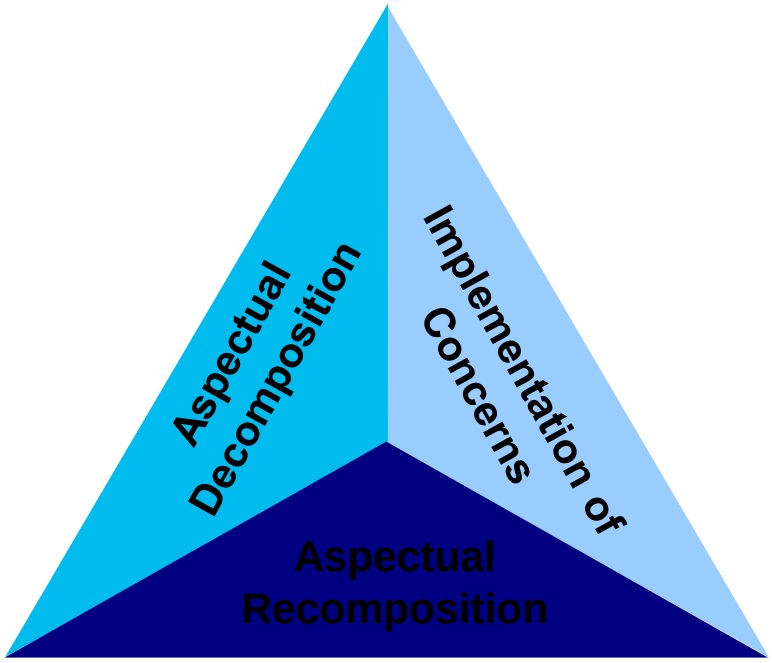
- Solution: Aspect-Oriented Programming
  - Aims to separate cross-cutting concerns
  - Reduces:
    - Duplicated code
    - Tangled code
    - Scattered code
    - Dependency between the concerns
  - Modularizes common code dealing with each of the concern which is applicable across multiple modules

AOP Fundamentals – Phases of AOP

1. Aspectual decomposition
   - Identifying the core concerns and cross-cutting concerns

2. Implementation of concerns
   - Implementing the identified core and cross-cutting concerns independently

3. Aspectual recomposition
   - Weaving the individual concerns into the core application code based on weaving rules



Checkpoint Question 1

Which of the following does AOP enable? (Select all that apply.)

A. Separation of concerns
B. Interception
C. Addition of aspects to existing software
D. Dependency Injection

Checkpoint Answer 1

Which of the following does AOP enable? (Select all that apply.)

A. **Separation of concerns**
B. **Interception**
C. **Addition of aspects to existing software**
D. Dependency Injection

AOP Concepts (1 of 2)

- Let's say we have a class with a few methods
  - it is needed to log a message mentioning which parameter is passed
    - Should be done before a method is executed
  - also it is required to log the exit from the method
    - Should be done after a method is executed

- Logging can either be added within the business service class or can be written separately in another class and then can be weaved using AOP
- Using Spring AOP, we can create aspects (separate class for logging)
- To weave it with business service class, we need to annotate methods using @Before and @After

AOP Concepts (2 of 2)

- Spring aspect can be like the one below

```
public class SimpleAspect {
    public void afterMethod() throws Throwable {
            System.out.println("After the method call");
    }
    public void beforeMethod(String myName){
            System.out.println("My name is "+myName);
    }
}
```

- The afterMethod() will log message after a method is called and beforeMethod() will log message before a method is called.

AOP Terminology

- Spring AOP framework uses the existing common AOP terminology:
  - Aspect
  - Join Point
  - Pointcut
  - Advice
  - Introduction
  - Weaving
  - Target object
  - Proxy object

AOP Terminology – Aspect

- Well-modularized cross-cutting concern
- A programming construct which enables cross-cutting concerns
  to be captured in modular units
- Aspect can:
  - Have usage constraint specifications
  - Extend another aspect, class or implement an interface
  - Have abstract fields and methods; i.e., it can be abstract
  - Have Java methods, fields, etc.

AOP Terminology – Join Point (1 of 3)

- Fundamental concept of AOP
- A well-defined location or a point in the execution of a program
- A point where one or more aspects can be applied
- Typically, Join Points can be of the following kinds:
  - Method and constructor call
  - Method and constructor execution
  - Exception handler execution
  - Field assignment: set and get
  - Class or object initialization: static and dynamic
- In Spring AOP, a Join Point is always a method execution.

AOP Terminology – Join Point (2 of 3)

```
public class CourseRegistrationService
{
    public CourseDao courseDao;
    public UserDao userDao;
    public void setCourseDao(CourseDao course) {
        this.courseDao = course;
    }
    public void setUserDao(UserDao user) {
        this.userDao = user;
    }
    public int registerStudentIntoCourse(long studentId, String courseId,
            String term) throws Exception
    {
        try {
            Student student = userDao.findByAutogenId(studentId);

            Course course = courseDao.findByCourseId(courseId);
            List<String> courseIdList = userDao.
                findRegisteredCourseListByStudentId(student.getAutoGenUserId());
            // Check if course is already registered for this user
            validateAlreadyCourseRegistered(courseIdList, course, student,courseId);
        } catch (Exception ex) {
            ex.printStackTrace();
            throw new Exception("Exception in registerStudentIntoCourse:  "
                    + ex.getMessage());
        }
        return 0;
    }
    private void validateAlreadyCourseRegistered(List<String> courseIdList,
            Course course, Student student, String courseId) throws Exception {
//do something;
    }
}
```

Field assignment Join Point

Method call Join Points

Exception handler execution Join Point

Method execution Join Point

AOP Terminology – Join Point (3 of 3)

- In Spring AOP framework, Join Point context information can be accessed by an advice during method execution:

Context	Join Point Method used to Obtain the Context	Description
this	getThis()	Returns the currently executing object that has been intercepted (AOP proxy)
target	getTarget()	Returns the target object of the execution (advised object)
args	getArgs()	Returns the method arguments passed at the Join Point
signature	getSignature()	Returns the method signature at the Join Point

AOP Terminology – Pointcut

- Selection or collection of Join Points defined to specify when/where an advice should be executed
- Selects Join Points and matches values at those points (which can be an advice)
- Can also collect context at the selected Join Points

AOP Terminology – Advice (1 of 2)

- The actual code to be executed when the pointcut is triggered
- Consists of the action to be taken at Join Points in a pointcut
- Can be marked before, after or around the Join Points in the pointcut
- Can be modeled as an interceptor or chain of interceptors around the Join Point

AOP Terminology – Advice (2 of 2)

- Types of advice:

Type	Description
Before	Invoked and executed before a Join Point
After returning	Invoked and executed after Join Point is completed, without throwing any exceptions
After throwing	Invoked and executed after Join Point exits by throwing an exception
After (finally)	• Invoked and executed after Join Point exits, no matter what the outcome is (either exception or completes successful) • Typically used for releasing resources
Around	• Surrounds a Join Point • Can perform custom functionality/logic before and/or after the Join Point • Decides whether to invoke a Join Point

AOP Component Comparison

Component	Description	Semantics
Aspect	A programming construct which enables cross-cutting concerns to be captured in modular units	Combines what, when and where
Join Point	A well-defined location or a point in the execution of a program (method call, method execution, exception handler, etc.)	Where
Advice	Has a body (method definition)	What
	Has a type (before, after, around, etc.)	When
Pointcut	Specifies a collection of Join Points	Where


Join point
Weaving
Advice
AOP
Proxy
Pointcut
Aspect
Target Object

Checkpoint Question 2

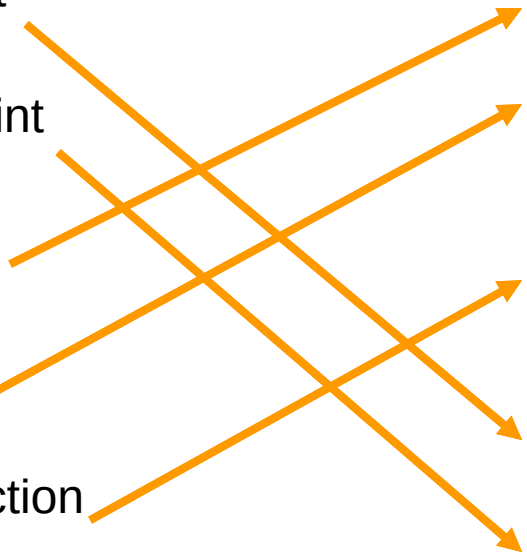
Match the terms with their explanations.

Terms	Explanations
1. Pointcut	A. Action to be taken at Join Points
2. Join Point	B. Well-modularized cross-cutting concern
3. Advice	C. Adding methods or fields to an advised class
4. Aspect	D. Set of Join Points
5. Introduction	E. Well-defined point of execution in a program

Checkpoint Answer 2

Match the terms with their explanations.

Terms	Explanations
1. Pointcut	A. Action to be taken at Join Points
2. Join Point	B. Well-modularized cross-cutting concern
3. Advice	C. Adding methods or fields to an advised class
4. Aspect	D. Set of Join Points
5. Introduction	E. Well-defined point of execution in a program



AOP Terminology – Weaving

- Links aspects/concerns into the application with other aspects/concerns
- Assembles modules into final form
- Can configure rules to combine different modules together
- Weaving can occur during:
  - Compile time
  - Load time
  - Runtime
- Spring AOP performs weaving at runtime.

AOP Terminology – Target Object or Advised Object

- The object on which an aspect or set of aspects can be applied
- Object being advised
- One or more aspects can advise the target object
- In Spring Framework, this object will always be a proxied object.

Checkpoint Question 3

AOP provided by Spring is a type of which of the following? (Select all that apply.)

A. Static AOP
B. Simple AOP
C. Dynamic AOP
D. Compile-time AOP

Checkpoint Answer 3

AOP provided by Spring is a type of which of the following? (Select all that apply.)

A. Static AOP
B. Simple AOP
C. Dynamic AOP
D. Compile-time AOP

Questions

Pointcut Additional Information

- * and .. are used as wildcard tokens in defining pointcut expressions.
- Pointcut expressions can be combined to create composite pointcut expressions:
  - For @AspectJ approach, use &&, || and ! Operators.
  - For schema-based approach, use and, or, and not keywords, respectively.
- Spring AOP allows creating user-defined pointcut expressions which can be referenced by the pointcut expression name – called as named pointcuts.
- Common pointcut expressions can be placed in a single aspect class and can be referred by multiple aspects.

Checkpoint Question 4

Common pointcut expressions can be placed in a single aspect class

and can be referred by multiple aspects.

A. True
B. False

An aspect class can have one advice only.

A. True
B. False

Checkpoint Answer 4

Common pointcut expressions can be placed in a single aspect class

and can be referred by multiple aspects.

A. True

B. False

B. False : class can have one advice only.

A. True

B. False

Checkpoint Question 5

Spring AOP supports AspectJ pointcut designators (set of operations)

for use in pointcut expressions.

A. True
B. False

Spring automatically identifies a class as an AspectJ aspect when it is annotated with @Annotation.

A. True
B. False

Checkpoint Answer 5

Spring AOP supports AspectJ pointcut designators (set of operations)

for use in pointcut expressions.

A. True

B. False

Spring automatically identifies a class as an AspectJ

B. False ıen it is annotated with @Annotation.

A. True

B. False

Questions

Spring AOP : See-It

Demonstration : Faculty will demonstrate how to implement AOP using Spring

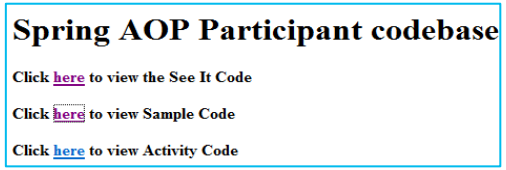
Environment: Eclipse

Duration: 20 min

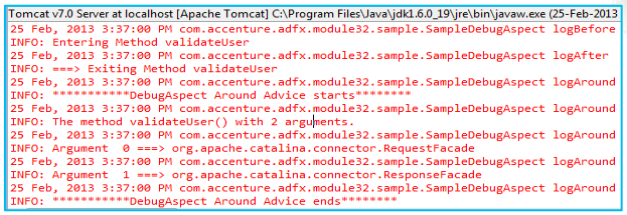
Steps:

1. Open the project ADFExtensionCodebaseM6AOP_participant in Eclipse
2. Run it on Tomcat server. Click the button
3. You should be able to see a click link for Home. Click on it. Refer the log
4. You should be able to see a click link for Sample. Click on it & refer the log
5. You should see a screen like the one below (see 2a below)
6. Click the link to view sample code
7. Click on Login button without any value in user name & password (see 2b below)
8. Try various combinations, refer the log and see how it behaves.

2a



2b



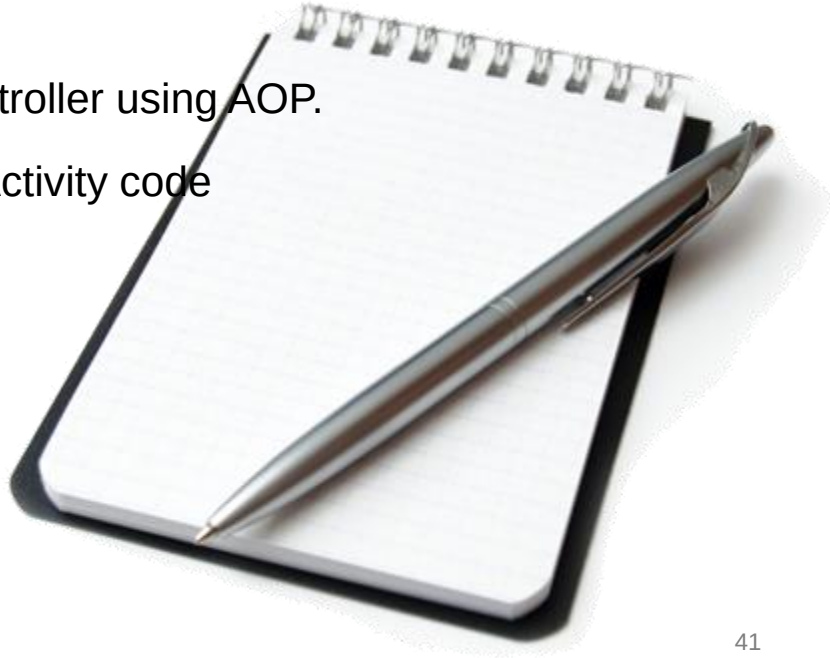
Spring AOP : Try It

Time Allocated : 30 minutes

Environment or File : ActivityController.java (complete) and ActivityDebugAspect.java (incomplete)

Steps :

1. Open the project ADFExtensionCodebaseM6AOP_participant in Eclipse
2. Open com.accenture.adfX.module5.activity.ActivityDebugAspect.java
3. Complete the **TODO 1-TODO 12** mentioned in ActivityDebugAspect.java to enable the log in Controller using AOP.
4. Save and run the program & Click the link to run Activity code
5. Click Add Museum and refer the log.
6. Click on each link and verify the log.



Questions

Module 6 Summary (1 of 4)

- AOP modularizes common code dealing with each of the concern which is applicable across multiple modules.
- AOP reduces object-oriented programming problems like duplicated, tangled and scattered code.
- The different cross-cutting concerns in software development which AOP addresses include logging, security, caching, profiling, transaction management, error handling, performance monitoring, etc.
- The three phases of AOP are aspectual decomposition, implementation of concerns and aspectual recomposition.

Module 6 Summary (2 of 4)

- Spring AOP framework uses the common AOP terminology such as aspect, Join Point, pointcut, advice, etc.
- Spring AOP supports only method execution Join Points.
- Spring AOP provide clean decoupling of “what” action to be performed from “when/where” that action should take place.
- Spring AOP performs weaving at runtime.

Module 6 Summary (3 of 4)

- Aspects in Spring can be implemented in two declarative ways: regular classes with either annotated AspectJ style or with XML schema-based approach.

- Spring AOP is implemented in pure Java, eliminating the need for special compilation process and making it suitable for use in any Java EE web container or application server.

- All the AOP-related configuration can be done in the Spring configuration file itself without the need for another AOP-specific configuration file.

Module 6 Summary (4 of 4)

- Spring AOP supports AspectJ pointcut designators (set of operations) for use in pointcut expressions.

- Pointcut expressions can be regular expressions, method pattern match, composite pointcuts, or user-defined pointcut expressions (which can be referenced by the pointcut expression name – also called named pointcuts).

- Spring AOP's @AspectJ and schema-based approaches use the same pointcut expressions and advice types.

Questions