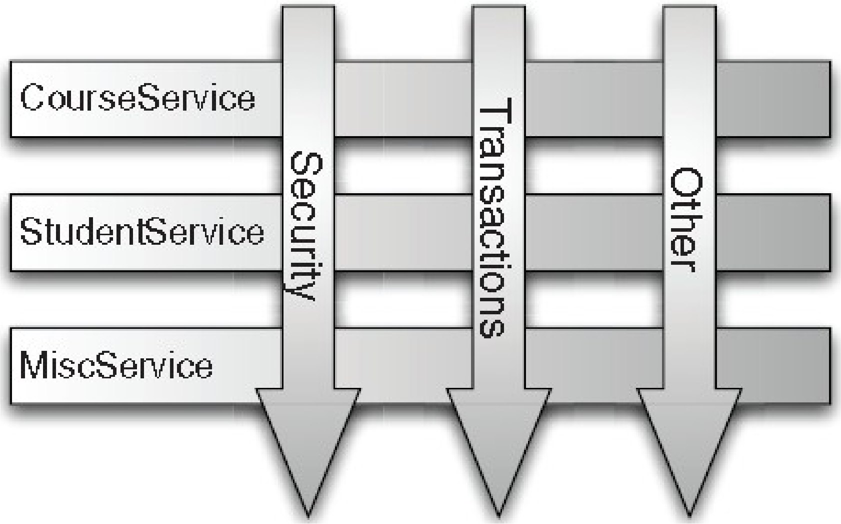
Spring AOP Basics

Topics

- Why AOP?
- AOP concepts
- Spring AOP

Why AOP?

- Aspect-oriented programming (AOP) provides for simplified application of cross-cutting concerns
- Examples of cross-cutting concerns
  - Logging
  - Transaction management
  - Security
  - Auditing
  - Locking
  - Event handling



AOP Concepts

AOP Concepts: Joinpoint

- Well-defined point during the execution of your application
- You can insert additional logic at Joinpoint's
- Examples of Jointpoint's
  - Method invocation
  - Class initialization
  - Object initialization

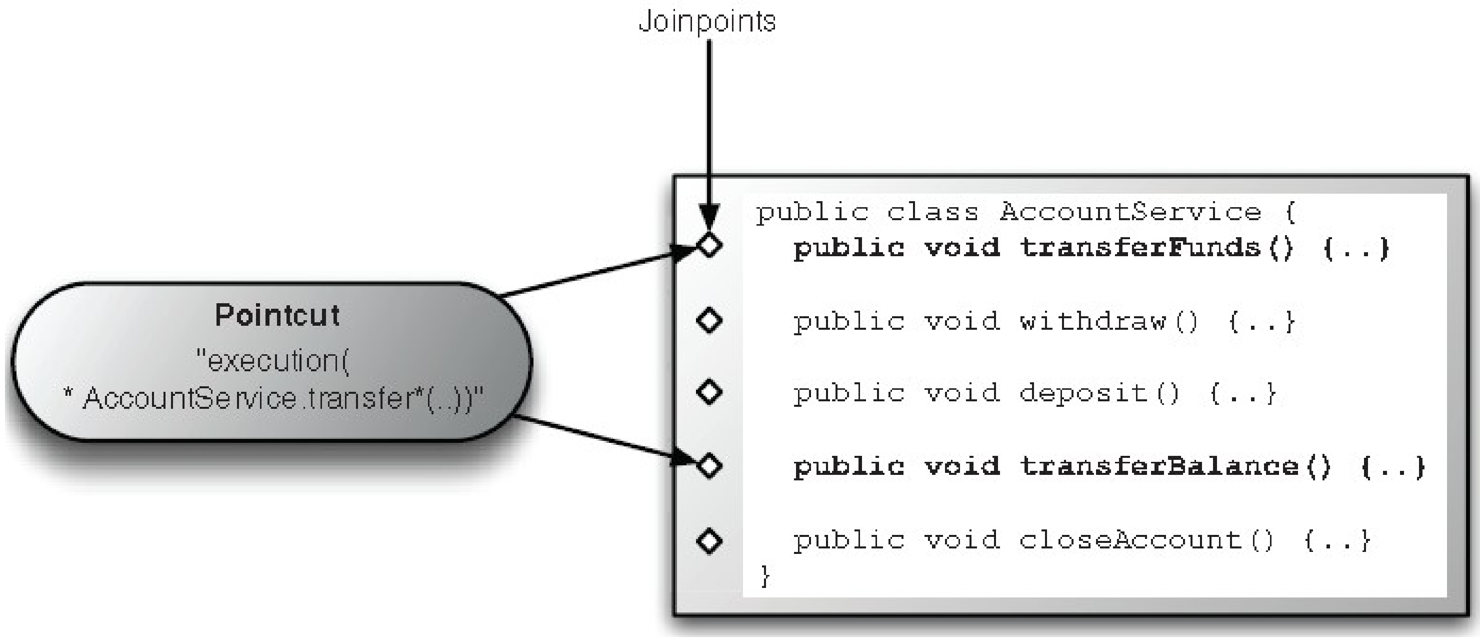
AOP Concepts: Advice

- The code that is executed at a particular joinpoint
- Types of Advice
  - before advice, which executes before joinpoint
  - after advice, which executes after joinpoint
  - around advice, which executes around joinpoint

AOP Concepts: Pointcuts

- A collection of joinpoints that you use to define when advice should be executed
- By creating pointcuts, you gain fine-grained control over how you apply advice to the components
- An expression that matches zero or more join points
- Example
  - A typical joinpoint is a method invocation.
  - A typical pointcut is a collection of all method invocations in a particular class
- Pointcuts can be composed in complex relationships to further constrain when advice is executed

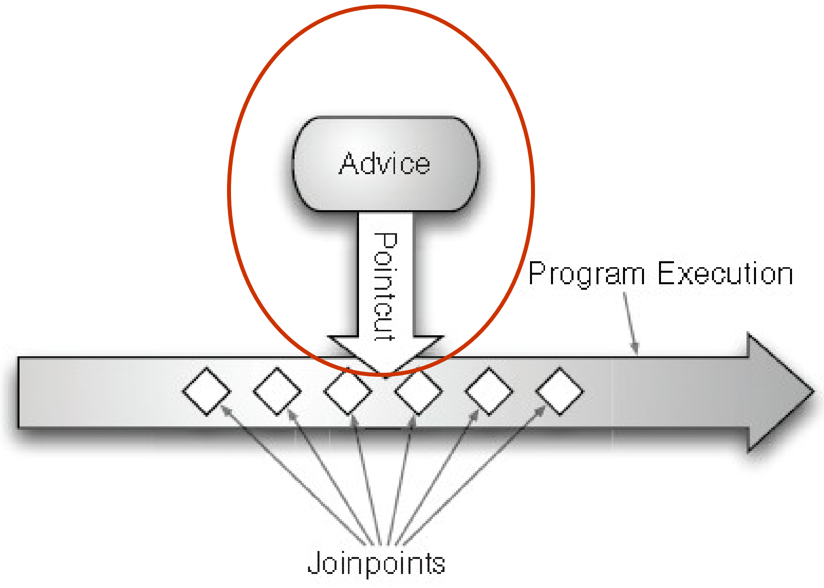
Defining Pointcuts

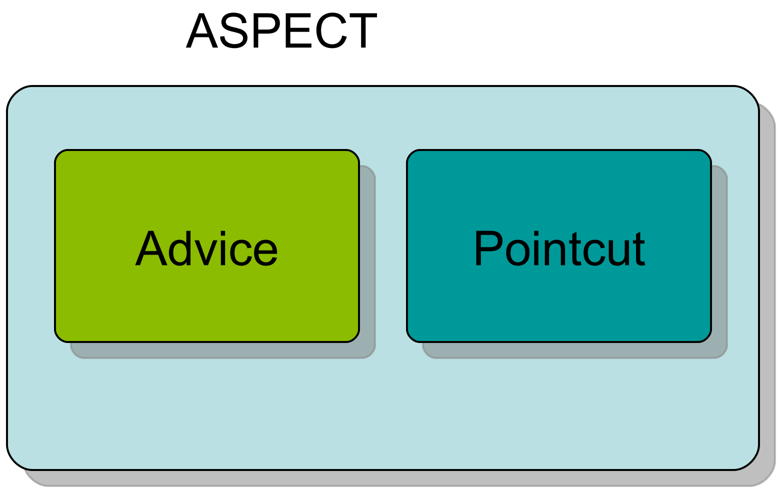


```
public class AccountService {
  public void transferFunds() {..}

  public void withdraw() {..}

  public void deposit() {..}

  public void transferBalance() {..}

  public void closeAccount() {..}
}
```

AOP Concepts: Aspects

- An aspect is the combination of advice and pointcuts





AOP Concepts: Weaving

- Process of actually inserting aspects into the application code at the appropriate point
- Types of Weaving
  - Compile time weaving
  - Runtime weaving

AOP Concepts: Target

- An object whose execution flow is modified by some AOP process
- They are sometimes called advised object

AOP Concepts: Introduction

- Process by which you can modify the structure of an object by introducing additional methods or fields to it
- You use the Introduction to make any object implement a specific interface without needing the object's class to implement that interface explicitly

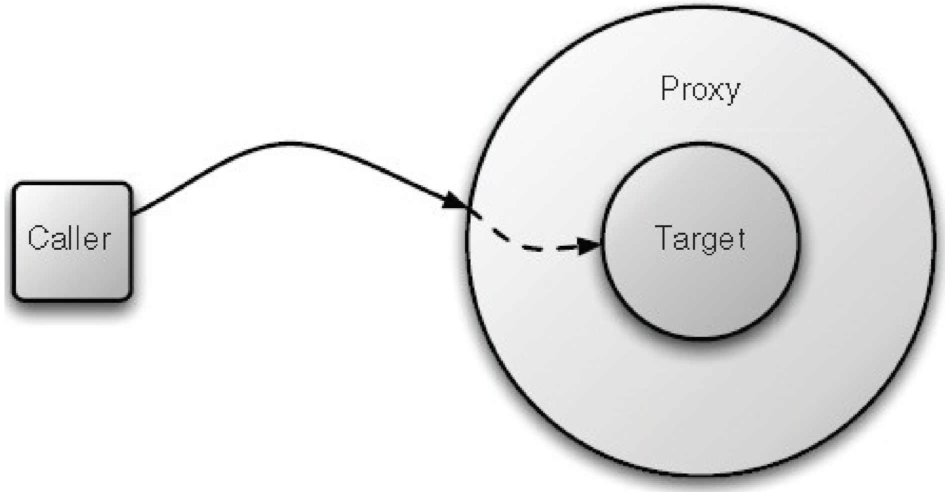
Types of AOP

Types of AOP

- Static AOP
  - The weaving process forms another step in the build process for an application
  - Example: In Java program, you can achieve the weaving process by modifying the actual bytecode of the application changing and modifying code as necessary
- Dynamic AOP
  - The weaving process is performed dynamically at runtime
  - Easy to change the weaving process without recompilation

Spring AOP

- In Spring, aspects are woven into Spring-managed beans at runtime by wrapping them with a proxy class
- The proxy class poses as the target bean, intercepting advised method calls and forwarding those calls to the target bean



Spring AOP

- Based on proxies
  - When you want to create an advised instance of a class, you must use the ***ProxyFactory*** class to create a proxy of an instance of that class, first providing the ***ProxyFactory*** with all the aspects that you want to be woven into the proxy
  - You typically use ***ProxyFactoryBean*** class to provide declarative proxy creation

HelloWorld Spring AOP

MessageWriter Class

- We want to display “Hello World !” through AOP

```
public class MessageWriter {
    public void writeMessage() {
            System.out.print("World");
    }
}
```

Target

- The joinpoint is the invocation of the *writeMessage*() method
- What we need is an “around advice”

```
public class MessageWriter {
        public void writeMessage() {
        System.out.print("World");
    }
}
```

Around Advice

- *MethodInterceptor* is AOP Alliance standard interface for around invoke
- *MethodInvocation* object represents the method invocation that is being advised

```
public class MessageDecorator implements
MethodInterceptor {
public Object invoke(MethodInvocation invocation)
throws Throwable {
System.out.print("Hello ");
Object retVal = invocation.proceed();
System.out.println("!");
return retVal;
}
```

Weaving MessageDecorator Advice

- Use *ProxyFactory* class to create the proxy of the target object

```
public static void main(String[] args) {
MessageWriter target = new MessageWriter();
// create the proxy
ProxyFactory pf = new ProxyFactory();
//Add the given AOP Alliance advice to the tail
//of the advice (interceptor) chain
pf.addAdvice(new MessageDecorator());
```

Weaving MessageDecorator Advice

```
//Set the given object as target
pf.setTarget(target);
//Create a new proxy according to the
//settings in this factory
MessageWriter proxy = (MessageWriter) pf.getProxy();
// write the messages
target.writeMessage();
System.out.println("");
// use the proxy
proxy.writeMessage();
}
}
```