Aspect Oriented Programming and AspectJ

Jens Gustavsson

Object Oriented Programming

- Objects represents things in the real world
- Data and operations combined
- Encapsulation
- Objects are self contained
- Separation of concerns

Outline

- Problems with OOP
- Introduction to AOP
- AspectJ

Example

```
class Account {
    private int balance = 0;

    public void deposit(int amount) {
        balance = balance + amount;
    }

    public void withdraw(int amount) {
        balance = balance - amount;
    }
}
```

Example

```
class Logger {
    private OutputStream stream;

    Logger() {
        // Create stream
    }

    void log(String message) {
        // Write message to stream
    }
}
```

Crosscutting

- Code in objects that does not relate to the functionality defined for those objects.
- Imagine adding:
  - User authentication
  - Persistence
  - Timing
  - …
- Mixing of concerns lead to:
  - Code scattering
  - Code tangling

Example

```
class Account {
    private int balance = 0;
    Logger logger = new Logger();

    public void deposit(int amount) {
        balance = balance + amount;
        logger.log("deposit amount: " + amount);
    }

    public void withdraw(int amount) {
        balance = balance - amount;
        logger.log("withdraw amount: " + amount);
    }
}
```

Mixing Concerns

- Correctness
  - Understandability
  - Testability
- Maintenance
  - Find code
  - Change it consistently
  - No help from OO tools
- Reuse

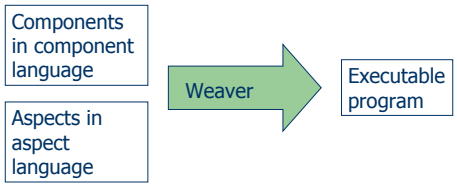
Aspect Oriented Programming

- Aspect = Concern that crosscuts other components
- Components written in *component language*
- Provide way to describe aspects in *aspect language*
- Not to replace OOP
- Does not have to be OO based

Weaving Time

- Preprocessor
- Compile time
- Link time
- Load time
- Run time

Aspect Weaving



Example

```
class Account {
    private int balance = 0;

    public void deposit(int amount) {
        balance = balance + amount;
    }

    public void withdraw(int amount) {
        balance = balance - amount;
    }
}
```

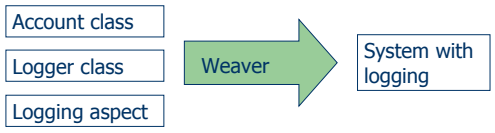
Example

```
define aspect Logging {

    Logger logger = new Logger();

    when calling any method(parameter "amount") {
        logger.log(methodname + " amount: " + amount);
    }
}
```

AOP Languages

- Join points
- Pointcuts
- Advice
- Aspects

Aspect Weaving



Join Point

A location within component code where the concern will crosscut it.

```
public void Account.deposit(int)
```

Pointcut

Specifies when a join point should be matched.

```
pointcut balanceAltered() :
  call(public void Account.deposit(int)) ||
  call(public void Account.withdraw(int));
```

Aspect

Similar to class. Groups join points, pointcuts and advice.

```
public aspect LoggingAspect {
  pointcut balanceAltered() :
    call(public void Account.deposit(int)) ||
    call(public void Account.withdraw(int));

  before(int i) : balanceAltered(i) {
    System.out.println("The balance changed");
  }
}
```

Advice

The code that shall be executed at a pointcut.

```
before(int i) : balanceAltered(i) {
  System.out.println("The balance changed");
}
```

AspectJ

- Xerox Palo Alto Research Center
- Gregor Kiczales
- Goal: Make AOP available to many developers
  - Open Source
  - Tool integration Eclipse
- Components in Java
- Java with extensions for describing aspects

AspectJ Demo

Patterns

- Match any type: *
- Match 0 or more characters: *
- `call(private void Person.set*(*)`
- `call(* * *.*(*)`
- `call(* * *.*(..)`
- All subclasses: `Person+`

Join Points

- Method call execution
- Constructor call execution
- Field get
- Field set
- Exception handler execution
- Class/object initialization

Logical Operators

- `call((Person+ && ! Person).new(..))`

Example

```
pointcut balanceAccess() :
  get(private int Account.balance);

before() : balanceAccess() {
  System.out.println("balance is
  accessed");
}
```

Advice

- Before
- After
  - Unqualified
  - After returning
  - After throwing
- Around

thisJoinPoint

- getTarget()
- getArgs()
- getSignature()
- getSourceLocation()

Example

```
pointcut withdrawal() :
  call(public void Account.withdraw(int));

before() : withdrawal() {
   // advice code here
}
```

Example

```
pointcut withdrawal() :
  call(public void Account.withdraw(int));

after() : withdrawal() {
   // advice code here
}
```

Example

```
pointcut withdrawal() :
  call(public void Account.withdraw(int));

after() throwing(Exception e) :
  withdrawal() {
   // advice code here
}
```

Example

```
pointcut withdrawal() :
  call(public void Account.withdraw(int));

after() returning : withdrawal() {
   // advice code here
}
```

Example

```
pointcut withdrawal() :
  call(public void Account.withdraw(int));

around() : withdrawal() {
  // do something
  proceed();
  // do something
}
```

Inter-type Declarations

- Add members
  - methods
  - constructors
  - fields
- Add concrete implementations to interfaces
- Declare that types extend new types
- Declare that types implement new interfaces

Aspect Instantiation

- Aspects are converted to classes by AspectJ compiler
- Types of instantiation:
  - Singleton
  - Per-object
  - Per-control-flow
- Aspects can contain fields (and methods)

Inter-type Declarations Demo

AOP Brainstorming Examples

- Resource pooling connections
- Caching
- Authentication
- Design by contract
- Wait cursor for slow operations
- Inversion of control
- Runtime evolution

Other AOP languages

- AspectWerkz
- JAC
- JBoss-AOP
- Aspect#
- LOOM.NET
- AspectR
- AspectS
- AspectC
- AspectC++
- Pythius