

# *Eliminating Cross-Cutting Concerns with Aspect-Oriented Programming*

*Capital District Java Developers Network  
November 15, 2007*

**Michael P. Redlich**  
**(908) 730-3416**  
**[michael.p.redlich@exxonmobil.com](mailto:michael.p.redlich@exxonmobil.com)**

# My Background (1)

---



## 🏠 Degree

- B.S. in Computer Science
- Rutgers University (go **Scarlet Knights!**)

## 🏠 “*Petrochemical Research Organization*” in New Jersey

- Senior Research Technician (1988-1998, 2004-present)
- Systems Analyst (1998-2002)

## 🏠 Ai-Logix, Inc.

- Technical Support Engineer (2003-2004)

## 🏠 Amateur Computer Group of New Jersey (ACGNJ)

- Java Users Group Leader (2001-present)
- President (2007-present)
- Secretary (2006)



# My Background (2)



## 🦋 Publications (co-authored with Barry Burd)

- [James: The Java Apache Mail Enterprise Server](#)
- [Avoid Excessive Subclassing with the Decorator Design Pattern](#)
- [Keeping Your Java Objects Informed with the Observer Design Pattern](#)
- [Manufacturing Java Objects with the Factory Method Design Pattern](#)
- [Resistance is Futile - How to Make Your Java Objects Conform with the Adapter Pattern](#)
- [Get to Know Your Java Object's State of Mind with the State Pattern](#)
- [Encapsulating Algorithms with the Template Method Design Pattern](#)



# Objectives

---



- △ Cross-Cutting Concerns
- △ Introduce Aspect-Oriented Programming (AOP)
- △ Example Application



# Software Concerns

---



## △ Primary concerns

- Core application functionality

## △ Secondary concerns

- System-wide objects that can be used in any primary concern



# What are Cross-Cutting Concerns?



- △ Secondary, system-wide concerns that can be found in multiple primary concerns
  - Logging
  - Authentication
  - Authorization
  - Persistence
- △ Requires certain behavior to occur at one or more points in the control flow of a program for its implementation to be correct

```
// cross-cutting concerns?
```

**Is this method a cross-cutting concern?**

```
public void addClaim(Claim claim) {  
    if(claim == null)  
        throw new IllegalArgumentException("null claim");  
    this.claims.add(claim);  
}
```

**Is this method a cross-cutting concern?**

```
protected void notifyListeners() {  
    for(Iterator iterator = listeners.iterator();  
        iterator.hasNext();) {  
        PolicyListener listener = iterator.next();  
        listener.policyUpdated(this);  
    }  
}
```

**This method needs to be invoked at the appropriate points in the control flow of the application**



```
// cross-cutting concerns?
```

```
public void addClaim(Claim claim) {  
    if(claim == null)  
        throw new IllegalArgumentException("null claim");  
    this.claims.add(claim);  
    notifyListeners(); ← Simply call notifyListeners() here...  
}
```

```
protected void notifyListeners() {  
    for(Iterator iterator = listeners.iterator();  
        iterator.hasNext();) {  
        PolicyListener listener = iterator.next();  
        listener.policyUpdated(this);  
    }  
}
```





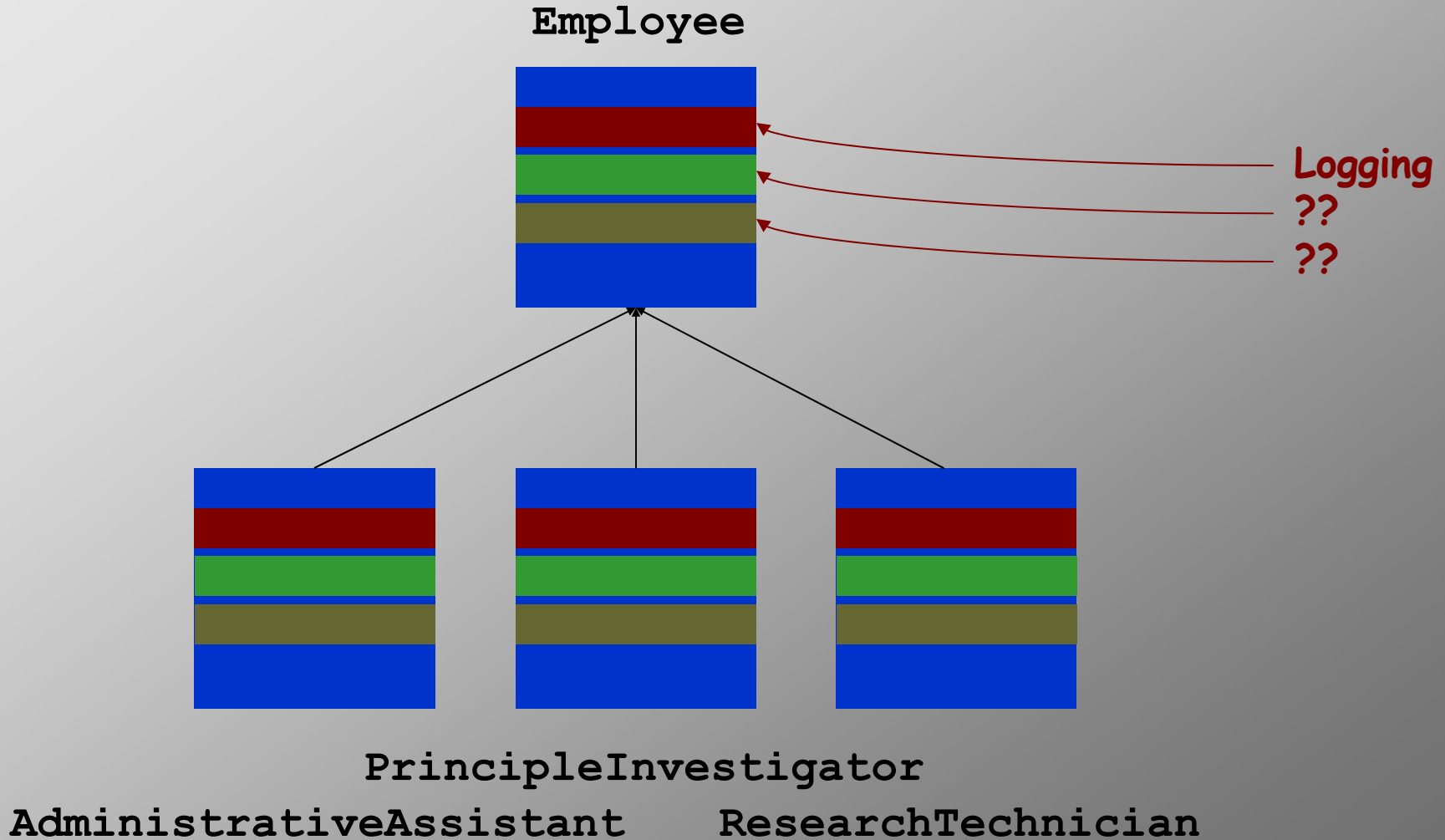
# *So, Are You Ready...*

---



☹ ...to review an initial Laboratory application?

# Cross-Cutting Concerns



# What is Aspect-Oriented Programming?



- ▲ A programming paradigm
- ▲ Separates cross-cutting concerns from the core functionality of the application
- ▲ Implementations:
  - Spring AOP
  - AspectJ
  - JBoss AOP
  - AspectWerkz \*

# What About Object-Oriented Programming?



▲ Object-Oriented Programming is excellent...

- ❑ ...for modeling real-world objects
- ❑ ...for separation of implementation from interface
- ❑ ...for creating interfaces that allow for loose object coupling

▲ However, secondary concerns must still be referenced in each of the primary concerns

- ❑ creating cross-cutting concerns

▲ Even Design Patterns can suffer from cross-cutting concerns!

# Core Concepts

---



☛ Join Points

☛ Pointcuts

☛ Advice

☛ Aspect



# Join Points (1)

---



## ♣ Identifiable points within the execution of a program

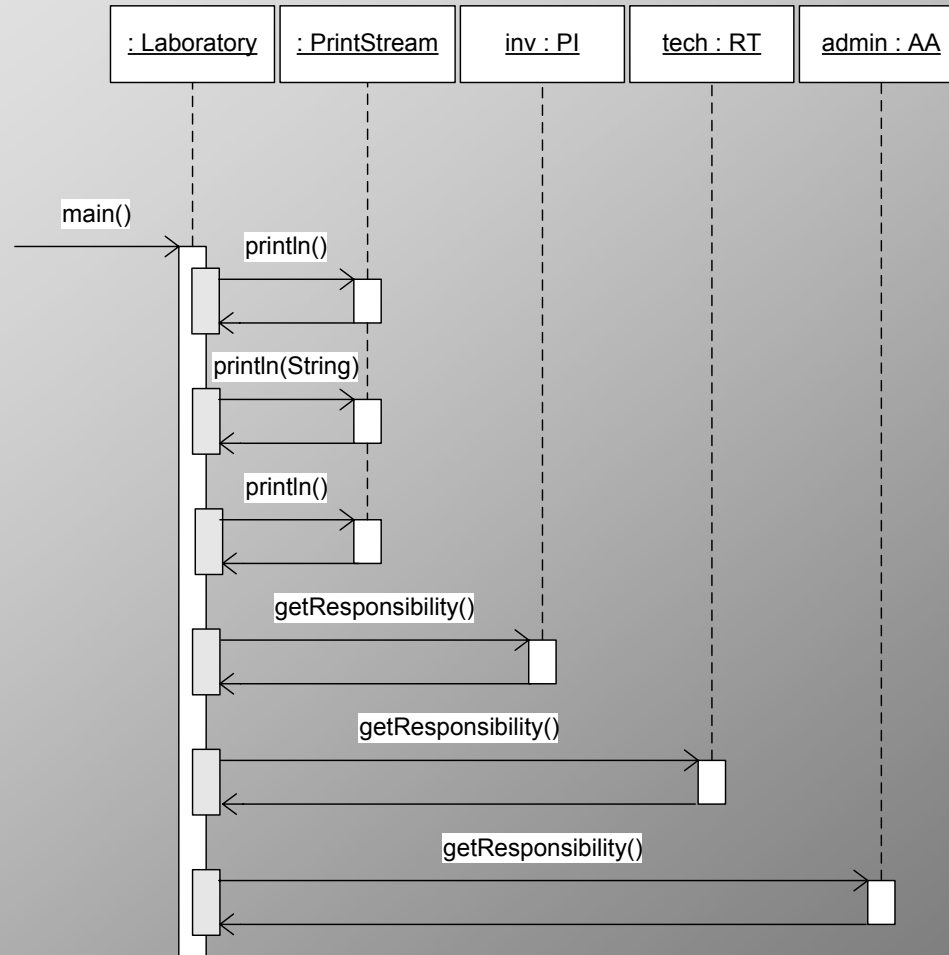
- Calling methods
- Initializing objects (constructor calls)
- Accessing/updating data members

## ♣ Place into which aspects are woven

## ♣ Join Point Model

- Defines a set of events visible to an aspect during program execution
- Join Points
- Pointcuts

# Join Points (2)



# Pointcuts



▲ Filters to match join points that meet a specification

▲ Three (3) types:

Kind

Scope

Context

▲ Prototype:

```
[visibility-modifier] pointcut name (ParameterList) :  
PointcutExpression ;
```





# Pointcuts (2)

---



## ⚠ Pointcut Expression

- ❑ Combination of pointcut designators and operators (&&), (||), and (!) as necessary

# Pointcut Designators



## 🔥 Kind designators

- Match certain “kinds” of join point events
- `call (methodSignature)`
- `execution (methodSignature)`

## 🔥 Context designators

- Match join points based on join point context
- `target (Type)`
- `args (Type)`

## 🔥 Scope designators

- Match join points within a certain scope

# Advice

---



- ⚠ Specifies what to do at the join points of interest
- ⚠ Code that is woven into a pointcut
- ⚠ Three (3) types:
  - Before
  - After
  - Around

# Before Advice



⚠ Executes before a matched join point

⚠ Prototype:

```
before(ParameterList) : pointcutName(ParameterList) {  
    // body of advice...  
}
```

# *After Advice (1)*

---



⚠ Executes after a matched join point

⚠ Three (3) basic forms:

- Successful return from a matched join point
- Returning from a matched join point upon some exception condition
- Returning from a matched join point either normally or upon an exception condition

# After Advice (2)



## 🔺 Prototypes:

```
after(ParameterList) returning(returnValue) :
pointcutName(ParameterList) {
    // body of advice...
}

after(ParameterList) throwing(ExceptionType) :
pointcutName(ParameterList) {
    // body of advice...
}

after(parameterList) : pointcutName(ParameterList) {
    // body of advice...
}
```



# Around Advice



⚠ Executes before and after a matched join point

⚠ Can determine:

- Continuation of program execution into matched join point
- Return type

⚠ Prototype:

```
ReturnType around(ParameterList) :  
pointcutName(ParameterList) {  
    // body of advice  
    if(// some desired condition)  
        proceed(ParameterList)  
}
```

# Aspect



⚠ A unit of modularity, encapsulation, and abstraction

☐ Sound familiar??

⚠ Aspect = Pointcut + Advice

⚠ Prototype:

```
[visibility-modifier] aspect {  
    // pointcut definition(s) ...  
    // advice definitions(s) ...  
    // other methods ...  
}
```



# *So, Are You Ready...*

---



☛ ...to review the refactored Laboratory application?

# Resources (1)

---



## 🐼 Spring Framework

☐ <http://www.springframework.org/>

## 🐼 AspectJ

☐ <http://www.eclipse.org/aspectj/>

## 🐼 JBoss AOP

☐ <http://labs.jboss.com/jbossaop/>

## 🐼 AspectWerkz

☐ <http://aspectwerkz.codehaus.org/>



# Resources (2)

---



## ♣ Capitol District Java Developers Network

☐ <http://www.cdjdn.com/>

## ♣ ACGNJ Java Users Group

☐ <http://www.javasig.org/>

## ♣ Princeton Java Users Group

☐ <http://www.myflex.org/princetonjug/>

## ♣ NYJavaSIG

☐ <http://www.javasig.com/>

## ♣ Philly Spring Users Group

☐ <http://phillyspring.org/>



# Further Reading (1)

---



## 🔺 Professional Java Development with the Spring Framework

- Rod Johnson, et. al
- ISBN 0-76457-483-3

## 🔺 Pro Spring

- Rob Harrop and Jan Machacek
- ISBN 1-59059-461-4

## 🔺 Spring in Action

- Craig Walls and Ryan Breidenbach
- ISBN 1-93239-435-4



# Further Reading (2)

---



## 🔥 Eclipse AspectJ

- ❑ Adrian Colyer, et. al
- ❑ ISBN 0-321-24587-3

## 🔥 AspectJ in Action

- ❑ Ramnivas Laddad
- ❑ ISBN 1-930-110-93-6