Aspect Oriented Programming with AspectJ

Ted Leung

Sauria Associates, LLC

twl@sauria.com

Overview

- Why do we need AOP?
- What is AOP
- AspectJ

Why do we need AOP?

- Modular designs are not cut and dried
- Responsibilities can be assigned to one or more classes
- Examples:
  - Every servlet for the administrative part of the site must check for a logged in administrator user
  - Site navigation : changing country in the UI (via multiple means) must update a bunch of data structures
  - Maintaining both ends of a two way relationship
  - Logging and tracing

What is AOP?

- Introduces the notion of crosscutting concerns
  - Concerns that you want to modularize
  - Concerns whose implementation is all over
- Introduces language mechanisms for identifying and capturing crosscutting concerns

Why bother with AOP?

- Capture the crosscutting concern explicitly
  - Both the behavior of the concern
  - The specification of its applicability
- Change is easier
  - Change the aspect – no grepping
- Aspects can be plugged in or out

AspectJ and AOP

- AspectJ is an aspect-oriented extension to Java
- One Concept : Join Points
- Four constructs
  - Pointcut Designators (Pointcuts)
  - Advice
  - Introduction
  - Aspects
- Aspects are composed of advice and introductions, and attached to pointcuts

Hello World AOP Style

```
public class HelloWorld {
  public static void main(String args[]) {
  }
}
```

```
public aspect HelloAspect {
  pointcut entry() :
   execution(public static void main(String[]));

  after() : entry() {
    System.out.println("Hello World");
  }
}
```

Join Points

- A well defined point in the program flow
- Created each time:
  - A method is called
  - A method executes
  - A field is get/set
  - An Exception handler executes
  - A dynamic initializer (constructor) executes
  - A static initializer executes

Pointcuts : dynamic crosscuts

- A declarative specification of a set of join points
- Easy to change versus copying calls or code to where they belong
- Examples:
  - Calls to method X from within the control flow of method Y
  - Calls to public methods

Pointcut designators

- field get and set
  - `set(int MyClass.field)`
- Method call
  - `call(int add(int, int))`
- Method execution
  - `execution(int add(int,int)`
- Exception Handling
  - `handler(IOException)`
- Constructor Execution
  - `initialization(class)`
- Lexical control flow
  - `within(class), withincode(method)`
- Dynamic control flow
  - `cflow(pointcut), cflowbelow(pointcut)`

Composing pointcuts

- Pointcut designators can be composed
- &&

  -
    ```
    target(MyClass) && call(void draw())
    ```
- ||

  -
    ```
    call(void draw) &&
    target(MyClass1) || target(package.*))
    ```
- !

  -
    ```
    call(void draw()) && !target(MyClass)
    ```

Signatures

- Basic signature:
  - `float compute(float, float)`
- On a specific class
  - `float Account.compute(float, float)`
- Any 0-ary method on a specific class
  - `Account.*()`
  - `Account.*(int)`
- Any public method returning an int
  - `public int *.*(..)`
- Any method throwing IOException
  - `public *.*(..) throws IOException`

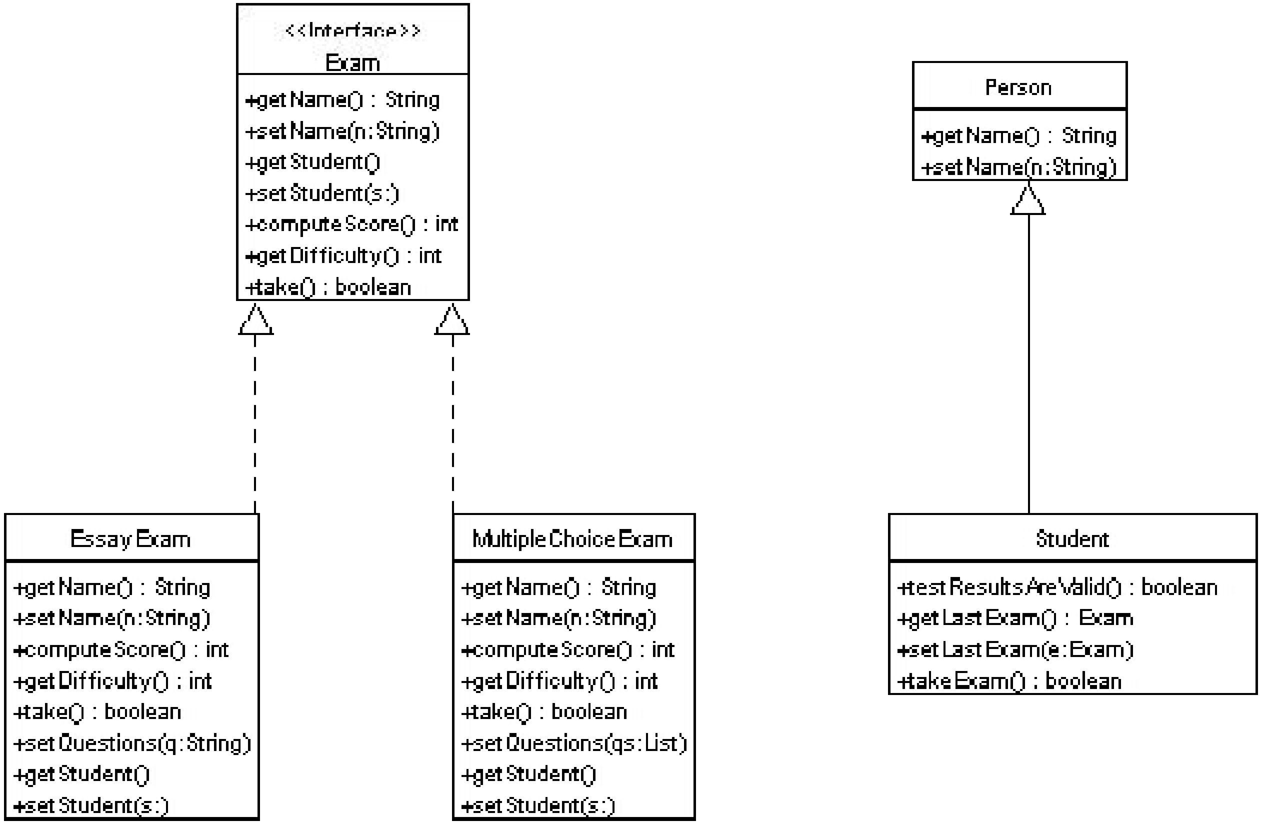
Type Patterns

- A type name
  - `vector`
- Wildcards
  - `java.util.*List`
  - `org.apache.xerces.xni.*`
  - `org.w3c..*`
- Subtypes
  - `java.util.AbstractList+`
- Arrays
  - `java.util.String[]`
- Composition
  - `java.io.* || java.nio.*`

Advice

- Code that runs at each join point selected by a pointcut
- Kinds of advice
  - before
  - after
    - after returning
    - after exception
  - around

Example Object Model



<<Interface>>
Exam

+get Name() : String
+set Name(n:String)
+get Student()
+set Student(s:)
+compute Score() : int
+get Difficulty() : int
+take() : boolean

Person

+get Name() : String
+set Name(n:String)

Essay Exam

+get Name() : String
+set Name(n:String)
+compute Score() : int
+get Difficulty() : int
+take() : boolean
+set Questions(q:String)
+get Student()
+set Student(s:)

Multiple Choice Exam

+get Name() : String
+set Name(n:String)
+compute Score() : int
+get Difficulty() : int
+take() : boolean
+set Questions(qs:List)
+get Student()
+set Student(s:)

Student

+test Results Are Valid() : boolean
+get Last Exam() : Exam
+set Last Exam(e:Exam)
+take Exam() : boolean

Examples

- Fields
- Wildcards
- ExceptionFlow

Accessing pointcut context

- We want to be able to access program values at a join point
- Pointcuts can take parameters
  - pointcut *name*(Type1 arg1, Type2 arg2) : args(arg1, arg2) && *pointcut*
- Advice can use those parameters
  - Around(Type1 arg1, Type2 arg2) : *name*(arg1, arg2) {

      ... use arg1 & arg2 in advice code

    }

Example

- Fields1
- Servlet Based

Advice precedence

- What happens when lots of advice matches?
- Dominates keyword
  - `aspect A dominates` *`TypePattern`* `{ }`
- Subaspect advice takes precedence
- Otherwise undetermined

Introduction: static crosscuts

- Aspect can introduce:
  - introduce fields
    - *Modifiers Type TypePattern.Id* { = *Expr* };
  - introduce methods
    - *Modifiers TypePattern*.new(*Formals*){ *Body* }
    - *Modifiers TypePattern.Id*(Formals) { Body }
  - Implement an interface
    - declare parents : *TypePattern* implements *TypeList*;
  - Extend a class
    - declare parents : *TypePattern* extends *TypeList*;
- Can introduce on many classes at once

Example

- SerialNumber

Aspect Extension

- Aspects can extend classes
- Aspects can implement interfaces
- Aspects can extend abstract aspects
  - The sub aspect inherits pointcuts

Example

- Abstract tracing Aspect

Associated aspects

- How many aspects are instantiated?
- singleton
  - By default
- aspect *Id* perthis(*Pointcut*)
  - 1 per currently executing object
- aspect *Id* pertarget(*Pointcut*)
  - 1 per target object
- aspect *Id* percflow(*Pointcut*)
  - 1 per control flow entrance
- aspect *Id* percflowbelow(*Pointcut*)
  - 1 per cflowbelow entrance

Privileged Aspects

- Aspects normally obey Java access control rules
- Aspects that can break encapsulation
  - `privileged aspect` *`Id`* `{`
    `}`

Example

- UpdateAspect

Tool Support

- Ant
- IDE Support
  - Emacs
  - Forte
  - JBuilder
- AJDoc
- AJBrowser
- Debugger

- Uses .lst files to do aspect weaving

AspectJ Status

- 1.0
  - Released 11/30/2001
- 1.1
  - Faster increment compilation
- 2.0
  - Dynamic crosscuts
  - Work on bytecode files

Musings

- AspectJSP
- Eclipse
- No need for .lst’s

Development uses

- Logging
- Tracing
- Timing
- Exception handling / logging
- Various kinds of invasive/non-invasive instrumentation

- Flexibility of pointcut designators

Application uses

- Consistency maintenance / checking
  - Keeping both sides of a bi-directional relationship up to date
- Policies
  - Security
  - Session Handling
  - Failure / Retry
  - Synchronization
- Context Passing
  - Avoids huge argument lists or carrier objects
- Multiple Views

To Learn More

- www.aspectj.org
- www.aosd.net
- CACM 10/2001
- These slides at:
  - www.sauria.com