# Intertype Declaration in AspectJ

Eduardo Figueiredo

http://www.dcc.ufmg.br/~figueiredo

# Intertype Declaration

- Pointcut and advice affect the dynamics (behaviour) of the system
- AspectJ also has options to manipulate the static structure
- Types of intertype declaration
  - To introduce new members: methods, attributes or constructors
  - To change the inheritance relationships

# Examples: Adding Members

```
private float Account.extraLimit;
```

**It includes a new float attribute, called *extraLimit*, into the *Account* class**

```
public static void Account.main(String[] args) {
   ...
}
```

**It includes a new *main* method into the *Account* class**

# Examples: Modifying Inheritance

```
declare parents: Account extends AbstractAccount;
```

**It makes *Account* to extend `AbstractAccount`**

```
declare parents: Account implements Serializable;
```

**It makes *Account* to implement `Serializable`**

```
declare parents: bank.* implements Serializable;
```

**It makes all classes and interfaces of the *bank* package to implement or extent `Serializable`**

# Aspect Inheritance

# Aspect

- Modular unit similar to a class in OOP
  - In addition to methods and attributes, it can also have pointcut, advice, and intertype declarations
- As aspect can intercept one or more classes of the system
- Its main goal is to implement a crosscutting concern
  - Classes implement the core concern

# Example of Logging (Java)

**The ATM class has crosscutting code to implement *Logging***

```
public class ATM {

   ...
   private int displayMainMenu() {
      screen.displayMessageLine( "\nMain menu:" );
      screen.displayMessageLine( "1 - View my balance" );
      screen.displayMessageLine( "2 - Withdraw cash" );
      screen.displayMessageLine( "3 - Deposit funds" );
      screen.displayMessageLine( "4 - Exit\n" );
      screen.displayMessage( "Enter a choice: " );
      int option = keypad.getInput();
      Logger.log("User option: " + option);
      return option;
   }
}
```

# *Logging* Aspect: Option 1

**Partial code of the *Logging* aspect**

```
public aspect Logging {

  ...

  public pointcut displayMainMenuPC() :
            call (private int ATM.displayMainMenu());

  after() returning (int option): displayMainMenuPC() {
    Logger.log("User option: " + option);
  }
}
```

**The *Logging aspect captures an int value returned by the displayMainMenu* method**

# The Logger Class

**Should we move/merge this class into an aspect?**

```
public class Logger {

  private static Vector<String> logs = new Vector<String>();

  public static void log(String text) {
    logs.add(text);
  }

  public static void printLog() {
    for (int i=0; i<logs.size(); i++) System.out.println( logs.get(i) );
  }
}
```

**Option 1: keep it as a separated class**

# *Logging* Aspect: Option 2 (merge)

***Merging the Logger class into the Logging aspect***

```
public aspect Logging {

  private Vector<String> logs = new Vector<String>();
  private static final int LOG = 0;

  public void log(String text) {
    logs.add(text);
  }
  ...
  public pointcut displayMainMenuPC() :
            call (private int ATM.displayMainMenu());

  after() returning (int option): displayMainMenuPC() {
    log("User option: " + option);
  }
}
```

**This code was merged from the *Logger* class**

# *Logging* Aspect: Option 3

**Partial code of the *Logging* aspect**

```
public aspect Logging extends Logger {

  ...

  public pointcut displayMainMenuPC() :
            call (private int ATM.displayMainMenu());

  after() returning (int option): displayMainMenuPC() {
    log("User option: " + option);
  }
}
```

**Aspects can extend either an aspect or a class. In this case, Logging extends the Logger class.**

# Bibliography


- R. LADDAD. **AspectJ in Action**, 2ª Ed. 2010.
  - Part 1  Understanding AOP and AspectJ