aosd.05
International Conference on
Aspect-Oriented Software Development

What's new in AspectJ 5 ?

Adrian Colyer
AspectJ Lead
IBM STSM

Jonas Bonér
AspectJ Committer
BEA

aosd.05 International Conference on Aspect-Oriented Software Development

Agenda

- In the headlines: AspectJ and AspectWerkz
- Java 5 support in AspectJ
- Plain Java AOP with @AspectJ aspects
- Enhanced load-time weaving
- User experience with AspectJ 5 and AJDT

AspectJ 5: AspectJ and AspectWerkz join forces

- Announced January 2005
- Complementary skills and technology
- Growing AOP is more important than competing
  - Tools, Java 5, weaving, aspect libraries
- AspectJ 5 v1.5.0
  - Initial release 2Q05
  - Roadmap to bring more of the AW features into AJ5
- Backed by IBM and BEA, hosted on Eclipse

Agenda

- In the headlines: AspectJ and AspectWerkz
- **Java 5 support in AspectJ**
- Plain Java AOP with @AspectJ aspects
- Enhanced load time weaving
- User experience with AspectJ 5 and AJDT

Java 5

- Annotations
  - Metadata that can be attached to many of the Java constructs
- *Autoboxing*
  - *Automatic conversion between primitive types and their OO equivalents (e.g. int and Integer)*
- *Varargs*
  - *Support for methods that take variable numbers of arguments, remember printf() in C?*
- Covariance
  - When overriding methods, you can choose to narrow the return type
- Generics
  - Improves type checking, most useful for Collections
- *Enums*
  - *Allows for a fixed set of values to be defined for a type*

Annotations: simple matching

```
set(@SensitiveData * *)
get((@SensitiveData *) org.xyz..*.*)
execution(@Oneway * *.*(..))
within(@Secure *)
handler(!@Catastrophic *)
staticinitialization(@Persistent *)
call(* *.*(@Immutable *,..))
```

Annotations: runtime type, context exposure

- Variations on `this, target, args`

```
@this(Foo)
@target(Foo)
@args(Foo,*,Goo)
```

- Exposing annotations as context

```
@this, @target, @args, @within
@withincode, @annotation
```

```
pointcut withinCriticalMethod(Critical c) :
            @withincode(c);
```

AspectJ 5 M2 ✓

Annotations: declare annotation

```
declare @field: * *DAO+.*: @Persisted;

declare @method:
   public * BankAccount+.*(..) :
   @Secured(role=“supervisor”);

declare @type:
   org.xyz.model..* : @BusinessDomain;
```

Covariance

- How do covariant signatures affect join point matching ?
- The signatures of `B.whoAmI()` are:
  `B B.whoAmI()`
  `A A.whoAmI()`

`call(A whoAmI())`
  - matches

`call(B A.whoAmI())`
  - does NOT match

```
class A {
    A whoAmI() {
        return this;
    }
}

class B extends A {
    B whoAmI() {
        return this;
    }
}

B b = new B();
b.whoAmI();
```

Generics – the issues

- How to match generic signatures at join points
- Pattern wildcards vs generic wildcards (* == ?)
- How to expose generic types as context
- Generics and inter-type declarations
- Generic aspects ?

Matching generic signatures

- call, execution, get, set ***match based on signature***
- For each of these signatures, which pointcuts will match?

```
void foo(List<Number> ns) {...}
✓ execution(* foo(List<Number>))
✓ execution(* foo(List<*>))
✘ execution(* foo(List<?>))
✓ execution(* foo(List<Object+>))

void goo(List<? extends Number> ns) {...}
✘ call(* goo(List<?>))
✓ call(* goo(List<? extends Number>))
✘ call(* goo(List<Number+>))
```

Runtime types and generic signatures

- this, target, args match based on RTTI
  - Do not allow wildcards
  - BUT… erasure eliminates RTTI for generic types
- Rules in AspectJ 5:
- If we can determine that a pc will always match based on signature
  - Match
- If we can determine that a pc will never match based on signature
  - Do not match
- If we determine that a pc *could* match based on a runtime test
  - Match with an “Unchecked” warning

Example

```
Class X { void foo(List<? extends Number> {…} }

List<String> ls = …
List<Double> ld = …
List<? extends Number> ln = …
```

✘ x.foo(ls) -> does not match
✓ x.foo(ln) -> matches
✓ x.foo(ld) -> matches with unchecked warning

Agenda

- In the headlines: AspectJ and AspectWerkz
- Java 5 support in AspectJ
- **Plain Java AOP with @AspectJ aspects**
- Enhanced load-time weaving
- User experience with AspectJ 5 and AJDT

The @AspectJ aspects

- AspectJ has
  - **ONE** language
  - **ONE** semantics
  - **ONE** weaver
- With two different development styles
  - Code Style

    ```
    public aspect MyAspect { }
    ```
  - Annotation Style

    ```
    @Aspect public class MyAspect { }
    ```

The @AspectJ aspects

- Java 5 annotations enable compilation with a standard Java compiler

```
@Aspect public class MyAspect { }

org.aspectj.lang.annotation.*
    @Aspect
    @Pointcut
    @Before, @Around, @After, ...
    @DeclareParents, ...
```

- Design goals
    - Support compilation of the largest subset of AspectJ applications possible using a standard Java 5 compiler
    - Be able to mix styles in the same application

An @AspectJ aspect

Aspect is ***@Aspect*** class

@Pointcut defines pointcuts

@Before annotated methods are before advice

```
@Aspect // defaults to singleton
public class NoOpAspect {


    @Pointcut("execution(void Math.add(..))")
    void addMethods(){};



    @Before("addMethods()")
    public void noop() {
        System.out.print("in advice");
    }

}
```

thisJoinPoint & parameter binding

- With code style **`thisJoinPoint`** is implicitly available

```
before(Foo foo) : call(* dup(int)) && this(foo) {
    println("at " + thisJoinPoint);
}
```

- With annotation style, **`JoinPoint`** must appear in the advice signature

```
@Before("call(* dup(int)) && this(foo)")
public void callFromFoo(JoinPoint thisJoinPoint, Foo foo) {
    println("at " + thisJoinPoint);
}
```

aosd.05 International Conference on Aspect-Oriented Software Development

Inter-type declaration

- `declare parents ... implements` follows a **mixin** strategy

```
@Aspect public class MoodIndicator {

    public static interface Moody {
        Mood getMood();
    }

    @DeclareParents("org.xyz..*")
    static class MoodyImpl implements Moody {
        private Mood m_mood;
        public Mood getMood() { return m_mood; }
    }

    ...
}
```

Agenda

- In the headlines: AspectJ and AspectWerkz
- Java 5 support in AspectJ
- Plain Java AOP with @AspectJ aspects
- **Enhanced load-time weaving**
- User experience with AspectJ 5 and AJDT

Load-time weaving in AspectJ 5

- Weaving is **`ClassLoader`** aware
  - Eligible classes are advised by aspects they are visible to
  - One or more deployment descriptor(s)
- Enabled through
  - Java 5 agents (JVMTI), JRockit agents (Java 1.3)
  - Command line script
  - Specific integration
- We introduce a deployment descriptor

  `META-INF/aop.xml`

  `META-INF/aop.properties` (J2ME …)
- Similar to AspectWerkz schemes

Load-time weaving

- Controls
  - Aspects to use
  - Weaver configuration
  - Eligible classes

```
<aspectj>
    <aspects>
        <!-- <aspect name="com.ltw.MyDebugAspect"/> -->
        <aspect name="com.ltw.Aspect"/>
    </aspect>
    <weaver options="-XlazyTjp">
        <include within="com.webapp..*"/>
    </weaver>
</aspectj>
```

Deployment-time aspect definition

```
abstract aspect com.generic.AbstractLogging {
   abstract pointcut tracingScope();
   ...
}

<aspectj>
  <aspects>
    <concrete-aspect
        name="com.ltw.DeploymentTimeAspect"
        extends="com.generic.AbstractLogging">
      <pointcut name="tracingScope"
        expression="within(com.biz.*)"/>
    </concrete-aspect>
  </aspect>
  <weaver options="-XlazyTjp"/>
</aspectj>
```

Agenda

- In the headlines: AspectJ and AspectWerkz
- Java 5 support in AspectJ
- Plain Java AOP with @AspectJ aspects
- Enhanced load-time weaving
- **User experience with AspectJ 5 and AJDT**

AJDT

- Simply understands either style (code or annotation)
- Integrates the enhanced LTW support

- Plus other benefits unrelated to AspectJ 1.5.0
  - Visualizer enhancements
    - deow, general markers
  - Incremental compilation & structure model
  - Eager parsing & model update
  - Cross-reference view

AspectJ 5 Timeline

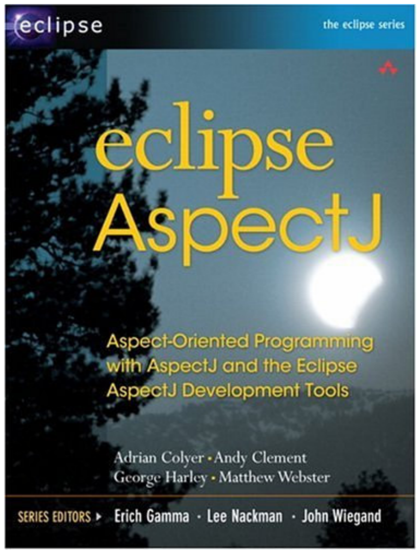
- `1.5.0M1` released December 10th
  - Included binary weaving of Java 5 compiled code
- Current dev stream
  - Compilation of Java 5 features and full support for annotations, autoboxing, varargs, covariance
    - For release as `1.5.0M2`
  - Work on enhanced LTW and annotation style going on in a branch
  - Generics work to be done, for release as `1.5.0M3`
- Possibly a `1.5.0M4` then release candidates and a final release
  - 2Q05
- AJDT support available for the new features shortly after each release

Summary

- AspectJ 5 integrates Java 5 features into the language
- Improved performance
- Annotation style development
- Enhanced Load Time Weaving support
  - Much more flexible deployment options
- AJDT will offer a consistent experience for both styles of development

Useful resources

- More info
  - http://eclipse.org/aspectj
  - http://aspectwerkz.codehaus.org
  - For new language features, see the **AspectJ developers notebook** linked from the AspectJ homepage
  - Buy the book ☺


eclipse
the eclipse series
eclipse AspectJ
Aspect-Oriented Programming with AspectJ and the Eclipse AspectJ Development Tools
Adrian Colyer · Andy Clement
George Harley · Matthew Webster
SERIES EDITORS ▸ Erich Gamma · Lee Nackman · John Wiegand

Adrian Colyer
adrian_colyer@uk.ibm.com

Jonas Bonér
jboner@bea.com

Around advice and custom proceed()

```
@Around("call(int Command.dup(int))
         && target(callee)
         && args(i)")
public int doNothing(MyJoinPoint jp, Command callee, int i) {
    return jp.proceed(callee, 2) + 3;
}


public static interface MyJoinPoint
extends ProceedingJoinPoint {

  public int proceed(Command callee, int arg);
}
```