

# AnnaBot

Annotations  
Assertions  
Testing  
Java

Ian Darwin

<https://darwinsys.com/javacook/>

# Annotations are sticky notes

@AnnotateThis

- Java Annotations are like Sticky Notes
  - Attach metadata to Java source code
  - Retained in compiled class file (usually)
  - Mechanism for runtime discovery
  - `someClass.getAnnotations()`, etc
- In Java language since 1.5 (2004)
- Used in Spring, Hibernate, Seam, JavaEE(EJB/JPA), JAX-WS, etc.



# Annotations In Action

Run-time (JAX-WS)

```
@WebService
```

```
public class Fred extends Caveman {
```

```
    @Override
```

Compile-time  
(not saved in .class file)

```
    public void callHome() {
```

```
        // call Wilma here
```

```
    }
```

```
}
```

# @Not Without Problems

- Extra annotations are ignored!
  - 100% open-ended namespace
  - A framework only looks for annotations it expects, where it expects them
  - If put annotation on wrong class or method or field, *framework won't see it*
  - No standard tools for checking
- To a tools developer, problem == opportunity



- The bot that
  - Not sm
- Basic mech
  - “Claim”  
Anno
  - Input =
  - Test ea



?  
ting”)  
ata  
n  
ch claim.

It really stands for  
Annotation Based  
Object Testing!

# Example Error

What's wrong with this picture?

```
@Entity public class Person {  
    @Id int id;  
    @Column(name="given_name")  
    public String getFirstName() {  
        return firstName;  
    }  
}
```



```
}
```



# AnnaBot code to catch it

```
claim JPAEntityClaim {  
  if (class.annotated(Entity)) {  
    atMostOne  
      method.annotated(javax.persistence.*),  
      field.annotated( javax.persistence.*)  
    {  
      error "Annotate JPA methods OR fields";  
    }  
  }  
}
```

# Inside AnnaBot

- “Claim” language will be compiled into Java bytecode using Antlr and Javassist
  - Antlr grammar is done; need to write actions
- For now write Claims as Java method calls
- Main program ties it together:
  - Find claim and target classes
  - Run each claim against each target



# “Run claim against target?”

- Uses Java Reflection API to check if method is annotated, etc.
- “Claim” class is Composite of Operators
- “Operator” interface: process(Class)
- “tree” package classes implement Operator
  - *Annotated*, *MethodAnnotated*, *FieldAnnotated*
    - *IsAnnotated* method – does “real” tests
  - Operators for *And*, *Or*, *AtLeastOne*, etc.

# Look under the hood

```
Field[] fields = c.getDeclaredFields();
boolean fieldHasJpaAnno = false;
for (Field field : fields) {
    Annotation[] ann = field.getDeclaredAnnotations();
    for (Annotation a : ann) {
        Package pkg = a.annotationType().getPackage();
        if (pkg != null && pkg.getName().startsWith("javax.persistence")) {
            fieldHasJpaAnno = true;
            break;
        }
    }
}
```



# Write Claims in Java

```
public class JPAEntityMethFieldClaim extends Claim {  
    public String getDescription() {  
        return "Annotate JPA methods OR fields";  
    }  
    public Operator[] getClassFilter() { // The "if" part  
        return new Operator[] {new ClassAnnotated("javax.persistence.Entity") };  
    }  
}
```

// Continued...

# Java Claim (continued)

```
public Operator[] getOperators() {  
    return new Operator[] {  
        new AtMostOne(  
            new FieldAnnotated("javax.persistence.*"),  
            new MethodAnnotated("javax.persistence.*"))  
        };  
    }  
}
```



# AnnaBot Performs

- Runs on 150 classes with a dozen claims in a few seconds
  - Too slow to run on every save in IDE
  - Too fast ***not*** to run it (hourly CI server)

# Recent Changes

- Regex matching for names, e.g.,  
`MethodAnnotated(javax.persistence.*, "set.*")`



# It will be even better

- “Claim Compiler” (AnnaBotC) will make it easier to add claims
  - Compile down to .class file
  - Can read claims from Jars
  - Will ship with jpa-claims.jar, spring-claims.jar, ...
- Claim language refinement
  - Annotation Attribute Assertions

...

# More Better

- Annotations can have *attributes*

```
@Column(name="my $*!# SQL column")
```

```
public String getDepartment() { ... }
```

- Results may be “undefined” - invalid table name
  - implementations, portability, ...
- Annotated subclasses need to do matching



# Cross-class Checking is Harder

- JPA requires that Entity modifier annotations apply to methods or fields
  - This is per “project” not per class!
- Solved by letting Java-based claim classes implement PrePostVerify interface
  - Place to check flags that you set when verifying individual classes

# The Source, Of Course

- AnnaBot is open-source, BSD-licensed

```
git clone \  
https://github.com/IanDarwin/annabot.  
git
```

- Please contribute back:
  - New claims!
  - Patches to code



# Takeaway - Usability

- Making a tool work may be relatively easy; making it really usable by others is harder
- Classpath Issues require care
  - Classes loaded by Reflection require all parent classes etc, to be on classpath
  - Need to hide this from user of tool
  - Hindsight: use 3d-party reflection API

# The Technical Paper

- Technical presentation (these slides) accepted for presentation at 'DEFECTS 2009' workshop at ISSTA 2009 (ACM SIGSoft / SIGPLAN yearly conference on software testing)
- Longer paper was published in *Advances in Software Engineering*, online at <http://hindawi.com/journals/ase/2010/540547.html>



# AnnaBot

## Q & A

Ian Darwin  
<https://annabot.org/>