Hacking Java

Enhancing Code at build or runtime

Sean Patrick Floyd

zalando

zalando

Sean Patrick Floyd

- Search Engineer @ Zalando
- ~20 years experience
- Java, Maven, Spring, Scala, Groovy
- Twitter: @oldJavaGuy
- StackOverflow: bit.ly/spfOnSO

Scope of this talk

- Overview of non-standard techniques
- Grouped by use case
- Some techniques are more mature than others
- Code samples and unit tests

Use Cases

- Value Objects
- Third party library patching
- Compile-time code defect analysis

zalando

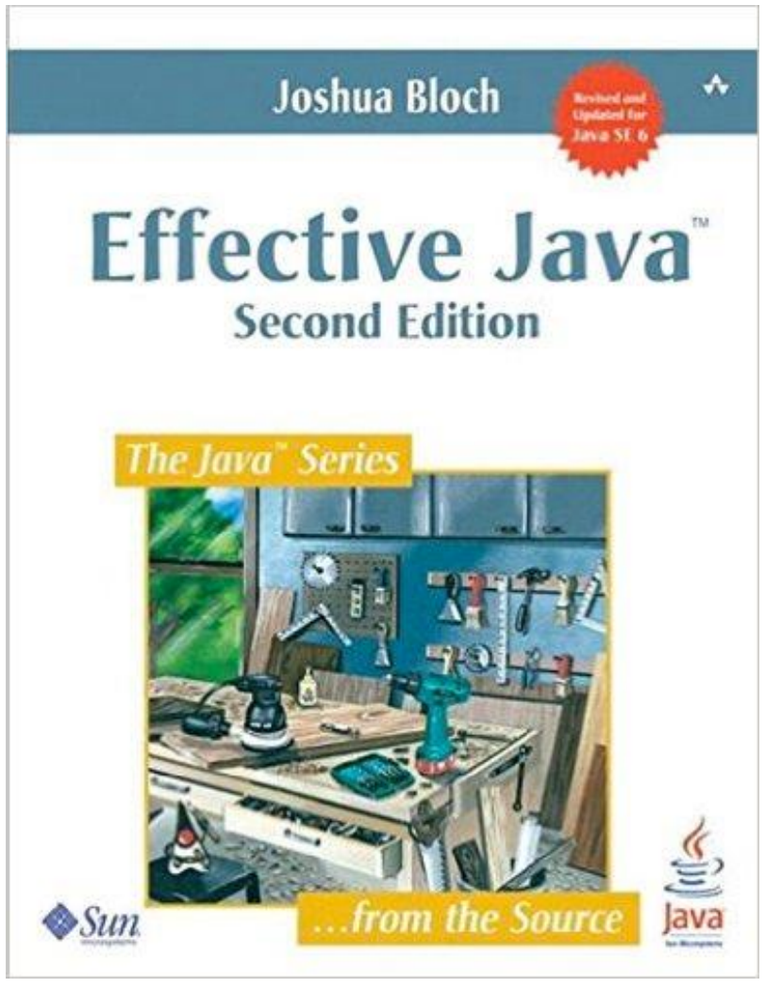
Github Project

- github.com/mostlymagic/hacking-java
  bit.ly/hackingJava

- Organized as Multi-Module Maven project
- Grouped by use case
- Sample code and unit tests for every technique

Use Case: Value Objects

- Standard, well defined behavior (equals(), hashCode(), toString()) according to Effective Java
- Mutable or immutable, with constructors, factory methods or builders
- Java Boilerplate very verbose and error-prone



zalando

The Pain

```
public class MyValueObject {

    private final String property1; private final boolean property2;
    public MyValueObject(String property1, boolean property2) {
        this.property1 = property1; this.property2 = property2;
    }

    public String getProperty1() { return property1; }
    public boolean isProperty2() { return property2; }

    @Override public int hashCode() { return Objects.hash(property1, property2); }

    @Override public boolean equals(Object obj) {
        if (this == obj) { return true; }
        else if (obj instanceof MyValueObject) {
            MyValueObject other = (MyValueObject) obj;
            return Objects.equals(this.property1, other.property1)
                && Objects.equals(this.property2, other.property2);
        } else { return false; } }

    @Override public String toString() {
        return MoreObjects.toStringHelper(this)
                .add("property1", property1).add("property2", property2).toString();
    }
}
```

For comparison: Scala

```
case class MyValueObject(property1: String, property2: Boolean) {}
```

Source: http://onlyinamericablogging.blogspot.com/2014/11/move-along-now-nothing-to-see-here.html

zalando

Boilerplate

- equals() / hashCode() / toString() / getters / setters / constructors / compareTo()
- IDEs offer generation, but in a static way
- potential bugs: adding or removing fields
- Boilerplate (n.): newspaper [and IT] slang for "unit of writing that can be used over and over without change," 1893, from a literal meaning (1840) "metal rolled in large, flat plates for use in making steam boilers." Source: http://www.etymonline.com/

en.wikipedia.org/wiki/Boilerplate_(robot)

zalando

Plan

- Let’s look at technologies that let us define value objects in a less-verbose way
- But with the full functionality (a test suite will monitor correct working of getters, equals, hashCode and toString)
- Different approaches: build-time, compile-time, run-time

Testing

- Expected Data structure (mutable or immutable):

User
- firstName (String)
- lastName (String)
- birthDate (LocalDate)
- addresses (List[Address])

Address
- street   (String)
- zipCode   (int)
- city   (String)

zalando

Test suite for value objects

```
public abstract class BaseUserTest {
    protected static final String FIRST_NAME = "Fred";  // etc.

    @Test public void equalsAndHashCodeAreSymmetrical() {
        Object user1 = createUser(); Object user2 = createUser();
        assertThat(user1, is(equalTo(user2))); assertThat(user2, is(equalTo(user1)));
        assertThat(user1.hashCode(), is(equalTo(user2.hashCode()))); }

    @Test public void toStringIsConsistent() {
        assertThat(createUser().toString(), is(equalTo(createUser().toString())));
        String s = createUser().toString();
        assertThat(s, containsString(FIRST_NAME)); /* etc. */ }

    @Test public void compareToIsSymmetrical() {
        Object l = createUser(), r = createUser();
        assertThat(l, instanceOf(Comparable.class));
        assertThat(r, instanceOf(Comparable.class));
        assertThat(((Comparable) l).compareTo(r),
          equalTo(((Comparable) r).compareTo(l))); }
}
```

zalando

Test suite (continued)

```
@Test public void propertyMapHasCorrectValues() {
    Object instance = createUser();
    Map<String, Object> map = getPropertyMap(instance);
    assertThat(map, hasEntry("firstName", FIRST_NAME)); // etc.
}

private static Map<String, Object> getPropertyMap(Object obj) {
    Map<String, Object> map = new TreeMap<>();
    try { Arrays.stream(Introspector.getBeanInfo(obj.getClass(), Object.class)
        .getPropertyDescriptors()).filter((it) -> it.getReadMethod() != null)
        .forEach((pD) -> { Method m = propertyDescriptor.getReadMethod();
            try { Object o = m.invoke(instance); map.put(pD.getName(), o);
            } catch (IllegalAccessException | ... e) { throw new ISE(e); }});
    } catch (IntrospectionException e) { throw new IllegalStateException(e); }
    return propertyMap;
}

protected abstract Object createUser();
```

Alternative Test Suite

- Guava's AbstractPackageSanityTests (http://bit.ly/AbstractPackageSanityTests)

- Automatically runs sanity checks against top level classes in the same package of the test that extends AbstractPackageSanityTests. Currently sanity checks include NullPointerTester, EqualsTester and SerializableTester.

- Nice, but not a perfect match for this use case

Annotation Processing

- JSR 269, pluggable annotation processing: Separate compiler lifecycle, well suited for code generation

- Service auto-discovery through ServiceLoader: `/META-INF/services/javax.annotation.processing.Processor` contains qualified processor names (bit.ly/srvcLdr)

- Docs: bit.ly/annotationProcessing (Oracle JavaDocs)

Project Lombok

- Name: Lombok is an Indonesian Island neighboring Java (“it’s not quite Java, but almost”)

- Project Lombok uses Annotation Processing to extend the AST. It uses internal compiler APIs (Javac and Eclipse)

Source: http://bit.ly/1lOfPbC

- Advantages: Little code, lots of power, no runtime dependencies

- Disadvantages: Relying on undocumented internal APIs, adds code that is not reflected in sources (inconsistent)

Lombok: mutable example

```
@Data
public class MyValueObject {
    private String property1;
    private boolean property2;
}
```

- Generates getters, setters, equals, hashCode, toString
- Additional fine-tuning annotations are available

Lombok: immutable example

```
@Data
public class MyValueObject {
    private final String property1;
    private final boolean property2;
}
```

- Generates constructor, getters, equals, hashCode, toString
- Builder version also available

Google AutoValue

- https://github.com/google/auto/tree/master/value
- “AutoValue […] is actually a great tool for eliminating the drudgery of writing mundane value classes in Java. It encapsulates much of the advice in Effective Java […]. The resulting program is likely to be shorter, clearer, and freer of bugs.” -- Joshua Bloch, author, Effective Java
- Advantages: Only public APIs used, no runtime dependencies
- Disadvantages: Less power and flexibility, only immutable types supported (or is that an advantage?)

zalando

AutoValue Sample code

```
@AutoValue // class needs to be abstract
public abstract class MyVO {
    // use JavaBeans property names
    // or simple field names
    public abstract String getProperty1();
    public abstract boolean isProperty2();

    // factory method for instantiation
    static MyVO create(String p1, boolean p2){
        return new AutoValue_MyVO(p1, p2);
        //"AutoValue_" + abstract class name
    }
}
```

zalando

CGLib BeanGenerator

- https://github.com/cglib/cglib
- CGLib is a “high level” byte code manipulation framework
- Widely used in production code, mostly by IOC and ORM frameworks (Spring, Guice etc)
- BeanGenerator is a playground feature that can create value types on the fly
- Works on Byte Code (Stack programming model): https://en.wikipedia.org/wiki/Java_bytecode

Caveat

- BeanGenerator generates field + getter + setter (no immutable types, no equals(), hashCode(), toString())
- To solve these issues, I am creating a dynamic proxy around the generated class, which in turn uses a full JavaBeans property map for all operations.
- `/* This is O(scary), but seems quick enough in practice. */`
  (Source: http://bit.ly/bigOScary)
- There is probably no sane use case for this

Don’t try this at home!

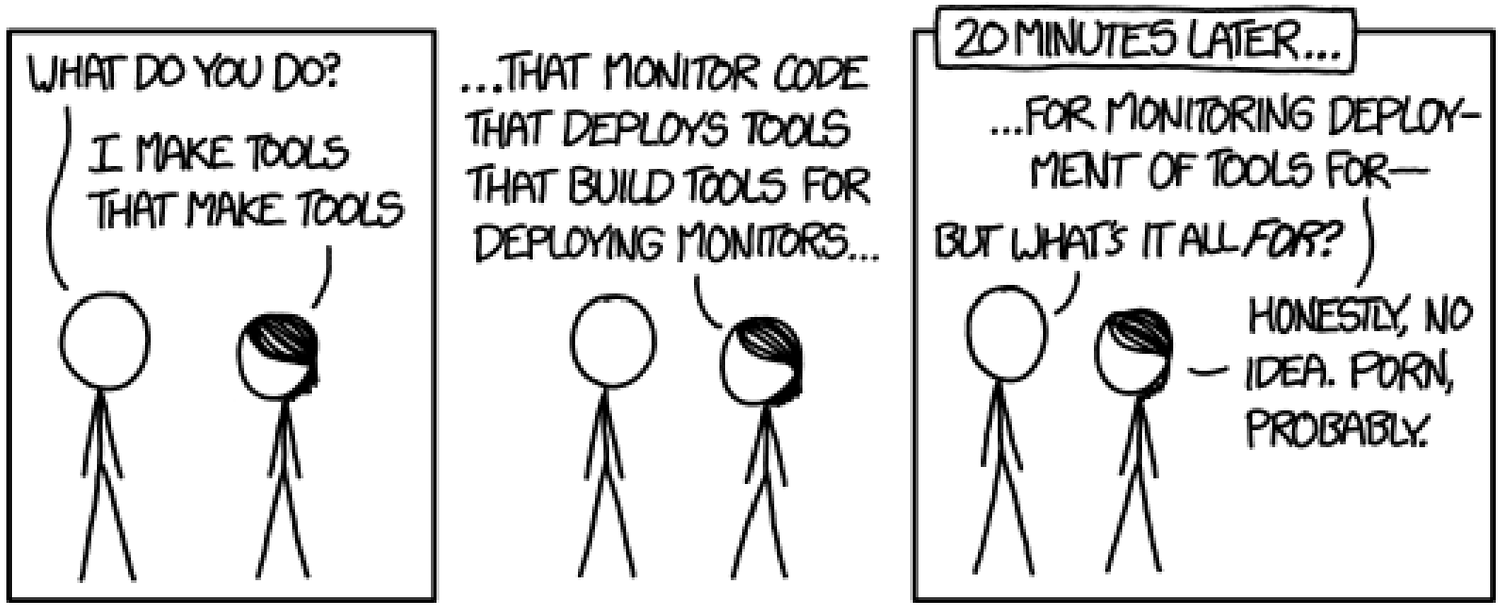
- There is (almost) no sane reason for generating value classes at runtime

- One possible scenario would be a CMS that lets you add new content types at runtime and is backed by a class-based ORM

Source: http://www.richardbealblog.com/dont-try-this-at-home/

Code Generation

- Techniques for creating source code dynamically at build time before compiling
- For those of us who like to “write code that writes code”



Source: http://xkcd.com/1629/

Source: http://www.memes.com/meme/487

zalando

Why code generation?

- Style wars: e.g. instanceof vs getClass()
- Custom requirements: Serializable, Jackson or JPA annotations
- Good news: you can do almost anything
- Bad news: you have to do almost everything yourself
- Most users will prefer “standard” solutions like Lombok or AutoValue

Adding code generation to the Maven Build

- Maven has two dedicated lifecycle phases for code generation, “generate-sources” and “generate-test-sources”
- The easiest way to achieve code generation is to execute a Groovy script in one of these phases, using the Groovy Maven Plugin (http://bit.ly/groovyMavenPlugin)
- By convention, Code generation uses the output folder target/generated-sources/{generator-name}
- Never generate code to src folder!

JCodeModel

- https://github.com/phax/jcodemodel
- Fork of Sun's abandoned JCodeModel project, which did code generation for JAXB
- Programmatic Java AST generation and source code generation
- Friendly, understandable API

Code Generation with JCodeModel (in Groovy)

```
class DtoGenerator {
   def codeModel = new JCodeModel()
   public generate(List<Dto> dtos, File targetDir) {
        dtos.each {
            def c = codeModel._package(it.packageName)
                  ._class(JMod.PUBLIC, it.className)
      defineConstructor(c, it.properties)
      defineGetters(c, it.properties)
      defineEqualsMethod(c, it.properties)
      defineHashCodeMethod(c, it.properties)
      defineToStringMethod(c, it.properties)
      defineCompareToMethod(c, it.compProperties)
   }
   targetDir.mkdirs(); codeModel.build(targetDir) }}
```

Constructor and Fields

```
class DtoGenerator { // continued
  private defineConstructor(JDefinedClass c,
                            Map fields) {
    def con = c.constructor(JMod.PUBLIC)
    def body = con.body()
    fields.each { e ->
      def type = resolveType(e.value)
      def f = c.field(PRIVATE | FINAL,
                      type, e.key)
      def param = con.param(type, e.key)
      body.assign(THIS.ref(f), param)
} } } // continued
```

Value Objects (what else)

- Techniques I haven't explored yet, but might in future:
  - Template-based code generation
  - More efficient byte code generation (ByteBuddy)
  - Interface-based proxies
  - JEP 169 (http://openjdk.java.net/jeps/169)

Use Case: Library Patching

- Scenario: You have to use a third party library, which has known defects
- Best choice: file an issue and / or submit a Pull Request to the library owner
- I'll assume this didn't work, you are dependent on the library, but you don't want to fork it, since you want to stay current
- Techniques that will let you patch lib in an automated way (ideally in a CI / CD pipeline)

Options

- If you have access to the source code, use a source code parser and dynamically patch the sources
- Otherwise, manipulate byte code (change or intercept existing code). Can be done statically at build time or dynamically at class load time
- Create a static patch and write tests which make sure the patch worked (I won’t explore this option)

Baseline

- As example I have written a trivial example class, with some pretty stupid, but harmless bugs:

```
public class FicticiousExample {
    public Integer yUNoReuseInteger(final int value) {
        System.setProperty(RAN_BUGGY_CODE, TRUE);
        return new Integer(value); }
    public String yUStringConcatInLoop(Iterable<String> data,
        String delim) {
        System.setProperty(RAN_BUGGY_CODE, TRUE);
        String v = "";
        final Iterator<String> it = data.iterator();
        if (it.hasNext()) { v += it.next(); }
        while (it.hasNext()) { v += delim + it.next(); }
        return value;
    }
}
```

zalando

Test suite

- In the test, I call the methods and assert that they work correctly and that the system properties weren't set

```
public abstract class FicticiousExamplePatchTest {
    private FicticiousExample fe;
    @After public void checkForEvilSystemProperty() {
        assertNull(System.getProperty(RAN_BUGGY_CODE)); }
  @Before public void initObject() {
    fe = new FicticiousExample(); }
  @Test public void assertCorrectIntegerBehavior() {
          assertTrue(fe.yUNoReuseInteger(123)
                  == fe.yUNoReuseInteger(123));
          assertFalse(fe.yUNoReuseInteger(1234)
                  == fe.yUNoReuseInteger(1234));
  }
  @Test public void assertCorrectStringBehavior() {  // etc.
```

AspectJ

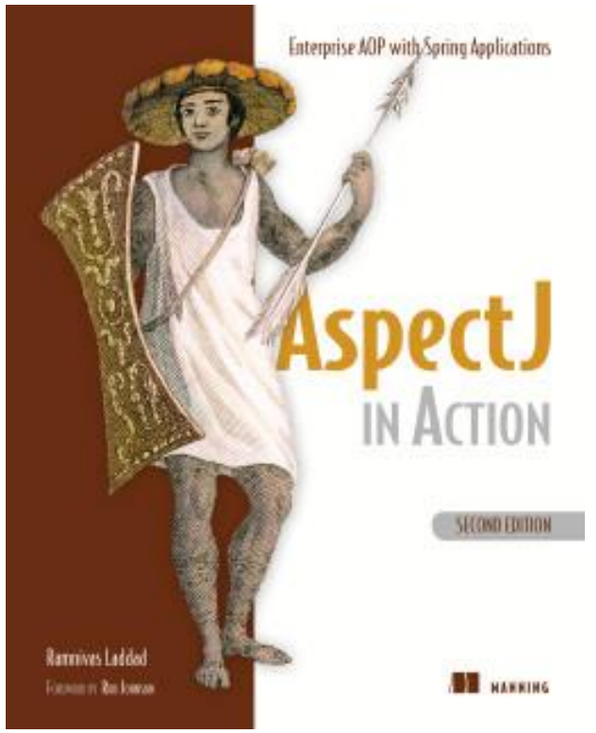
- https://eclipse.org/aspectj/

- Aspect-oriented language
  (.aj format or Java with @AspectJ annotations)

- ajc = AspectJ Compiler (after javac, instead of javac)

- static compilation or load time weaving through agent

- Docs are ancient / non-existent, use mailing list or read “the” book:
  bit.ly/ajInAction



Patching Aspect

```
public aspect FicticiousExamplePatch{
    public pointcut integerMethodCalled(int value) :
        execution(* FE.yUNoReuseInteger(..)) && args(value);
    public pointcut stringMethodCalled(Iterable<String> it, String s):
        execution(* FE.yUStringConcatInLoop(..)) && args(it, s);

    Integer around(int i) : integerMethodCalled(i){
        return Integer.valueOf(i); }
    String around(Iterable<String> it, String s) :
        stringMethodCalled(it, s){
          java.util.Iterator<String> iterator = it.iterator();
          StringBuilder sb = new StringBuilder();
          if(iterator.hasNext()){
              sb.append(iterator.next());
              while(iterator.hasNext()){
                  sb.append(s).append(iterator.next()); }
          }
         return sb.toString(); }
}
```

AspectJ capabilities

- Extending / replacing / enriching code
- Inter-type declarations (~= Traits)
- Policy Enforcement (define custom compiler errors)
- Very flexible, can work on source or byte code
- Standard usage: Cross-cutting concerns (security, logging etc.)

Source Code Manipulation

- Extract sources to generated-sources directory
- Parse and manipulate the sources (non-idempotent!?)
- Compile sources and distribute your library

JavaParser

- http://javaparser.github.io/javaparser/
- Parsers generated from JavaCC (Java Compiler Compiler) -> complete Java 1.8 grammar support
- API not as friendly as JCodeModel (no management of types and references etc.), but the only available fully functional Java source code parser
- Again, I'll be using Groovy to do the actual work. It could be done in Java too, but the build setup would be more complex

zalando

PatchVisitor

```
class MyVisitor extends VoidVisitorAdapter<Void> {

  public void visit(MethodDeclaration n,
                    Object arg) {
    if (n.name == "yUStringConcatInLoop") {
      n.body = new BlockStmt() // delete body
      patchStringMethod(n.body, n.parameters[0],
                                n.parameters[1])
    } else if (n.name == "yUNoReuseInteger") {
      n.body = new BlockStmt() // delete body
      patchIntegerMethod(n.body, n.parameters[0])
    } else super.visit(n, arg) }
```

PatchVisitor (Integer method)

```
private patchIntegerMethod(
    BlockStmt b, Parameter p) {
    def t = new ClassOrInterfaceType("Integer");
    def e = new TypeExpr(); e.type = t
    def mc = new MethodCallExpr(e, "valueOf")
    mc.args.add(new NameExpr(p.id.name))
    b.getStmts().add(new ReturnStmt(mc))
}
// all this for
// return Integer.valueOf(i);
```

Tradeoff

- AST manipulation is hard, very verbose and error prone
- But: It’s still better than trying to do it with Regex (insert obligatory Regex Cthulhu link here)
- friendlier Alternative: walkmod.com (built on JavaParser, “walkmod is an open source tool to apply and share your own code conventions.”)

Source: http://subgenius.wikia.com/wiki/Cthulhu

Use Case: Defect Analysis

- Identify bug patterns, reject them at compile time
- Not mentioning the “good old” tools: CheckStyle, PMD, FindBugs. They are best used in a separate CI build
- Focus on tools that hook directly into the compile process

Source: http://sequart.org/magazine/59883/cult-classics-starship-troopers/

zalando

Test Harness

```
public abstract class AbstractDefectDetectionTest extends AbstractCompilerTest {

    private DefectAnalysisEngine engine;

    @Before public void setupEngine() { this.engine = instantiateEngine(); }

    @Test public void detectWellBehavedClass() {
        CompilationResult r = engine.compile(sourceFileFor(wellBehavedClass()));
        assertThat(r, isSuccess());
    }

    @Test public void detectIllBehavedClass() {
        CompilationResult r = engine.compile(sourceFileFor(illBehavedClass()));
        assertThat(r, isFailureWithExpectedMessage(expectedErrorMessage()));
    }

    protected abstract DefectAnalysisEngine instantiateEngine();
    protected abstract Class<? extends WellBehaved> wellBehavedClass();
    protected abstract Class<? extends IllBehaved> illBehavedClass();
    protected abstract String expectedErrorMessage();

}
```

zalando

ForkedRun

- Helper class that calls a Java class in a separate process, copying the Classpath from the local context, abstracting the output away to a CompilationResult class, that we can run matchers against.

- Fluent API with matchers for converting classpath to bootclasspath and other nice features

- http://bit.ly/forkedRun

zalando

ForkedRun (sample)

```
public final CompilationResult run() {
    final Set<String> classPath = new LinkedHashSet<>();
    final Set<String> bootPath = new LinkedHashSet<>();
   try {
        dispatchClassPath(classPathElements, bootPathElements);
        String javaExe = getExecutable();
        List<String> cmds =
            buildCommands(classPath, bootPath, javaExe);
        ProcessBuilder pb = new ProcessBuilder().command(cmds);
        Process proc = pb.start();
        List<String> errorMsg = gatherOutput(proc);
        int status = proc.waitFor();
        return new CompilationResult(status == 0, errorMsg);

    } catch (InterruptedException | … | URISyntaxException e) {
        throw new IllegalStateException(e); } }
```

False positives

- Example: Immutability (impossible to reliably detect, by any technology known to me)

```
public final class ImmutableUser{

   private final String name; private final Date birthDate;
   private final List<String> nickNames;

   public ImmutableUser(String name, List<String> nickNames,
                        Date birthDate) {
      this.name = name;
      this.nickNames = ImmutableList.copyOf(nickNames);
      this.birthDate = new Date(birthDate.getTime()); }

   public String getName() { return name; }
   public List<String> getNickNames() {
      return ImmutableList.copyOf(nickNames); }
   public Date getBirthDate() { return new Date(birthDate.getTime()); }
}
```

What can we detect?

- Forbidden classes, erroneous usages
- Package-level architecture restrictions (MicroServices or OSGI are a better solution)
- Implementation inconsistent with annotations or interface (e.g. Nullness)

zalando

Google Error Prone

- http://errorprone.info/
- Google 20% project
- Wrapper around javac, with compatible API
- Many defined bug-patterns, most specific to Google (Android, Protobuf, Gwt, Guice)
- New Bug Patterns can only be added via Pull Request to Google :-(
- Integrated with Maven, Gradle, Ant etc.

Obscure bug patterns

- Example: Regex bug detection

```
public class IllBehavedRegex {

  static List<String> splitByAlpha(String s) {
    return Arrays.asList(
      s.split("[a-z")); // bad pattern
  }

}
```

zalando

Checker Framework

- http://types.cs.washington.edu/checker-framework/
- The Checker Framework enhances Java's type system to make it more powerful and useful. This lets software developers detect and prevent errors in their Java programs. The Checker Framework includes compiler plug-ins ("checkers") that find bugs or verify their absence. It also permits you to write your own compiler plug-ins.
- Disadvantage: needs to be installed separately, hard to integrate in a build system
- But: checkers can be used as plain annotation processors

zalando

Java 8 Type Annotations

- As of the Java SE 8 release, annotations can also be applied to any type use […] A few examples of where types are used are class instance creation expressions (new), casts, implements clauses, and throws clauses. http://bit.ly/typeAnnotations

- The Checker Framework embraces these new annotation usages

Source: http://memegenerator.net/

Example: Nullness

```
public class WellbehavedNullness {

  @PolyNull String nullInOut(@PolyNull String s) {
    if (s == null) return s;
    return s.toUpperCase().trim();
  }

  @MonotonicNonNull
  private String initallyNull;

  public void assignField(@NonNull String value) {
    initallyNull = checkNotNull(value);
  }

  @Nonnull public String getValue() {
    if (initallyNull == null)
      initallyNull = "wasNull";
    return initallyNull;
  } }
```

Source: https://imgflip.com/memegenerator

AspectJ (again)

- AspectJ allows custom compiler errors, according to static checks
- Architecture checks: package a may not access package b
- Forbidden usage patterns: e.g. Spring MVC controller may not take OutputStream param
- Forbidden classes: Enforce deprecation of legacy classes
- Most of these problems can be solved differently (MicroServices etc.)

Source: http://www.someecards.com/usercards/viewcard/MjAxMy1jZWRhODllNTI3YWI4Yjkz

zalando

Forbid usage of deprecated types

```
public aspect PolicyEnforcementAspect{

  pointcut badCall() :
        call(* Hashtable.*(..))
     || call(Hashtable.new(..))
     || call(* Vector.*(..))
     || call(Vector.new(..));
   declare error : badCall()
 "Hashtable and Vector are deprecated!";
}
```

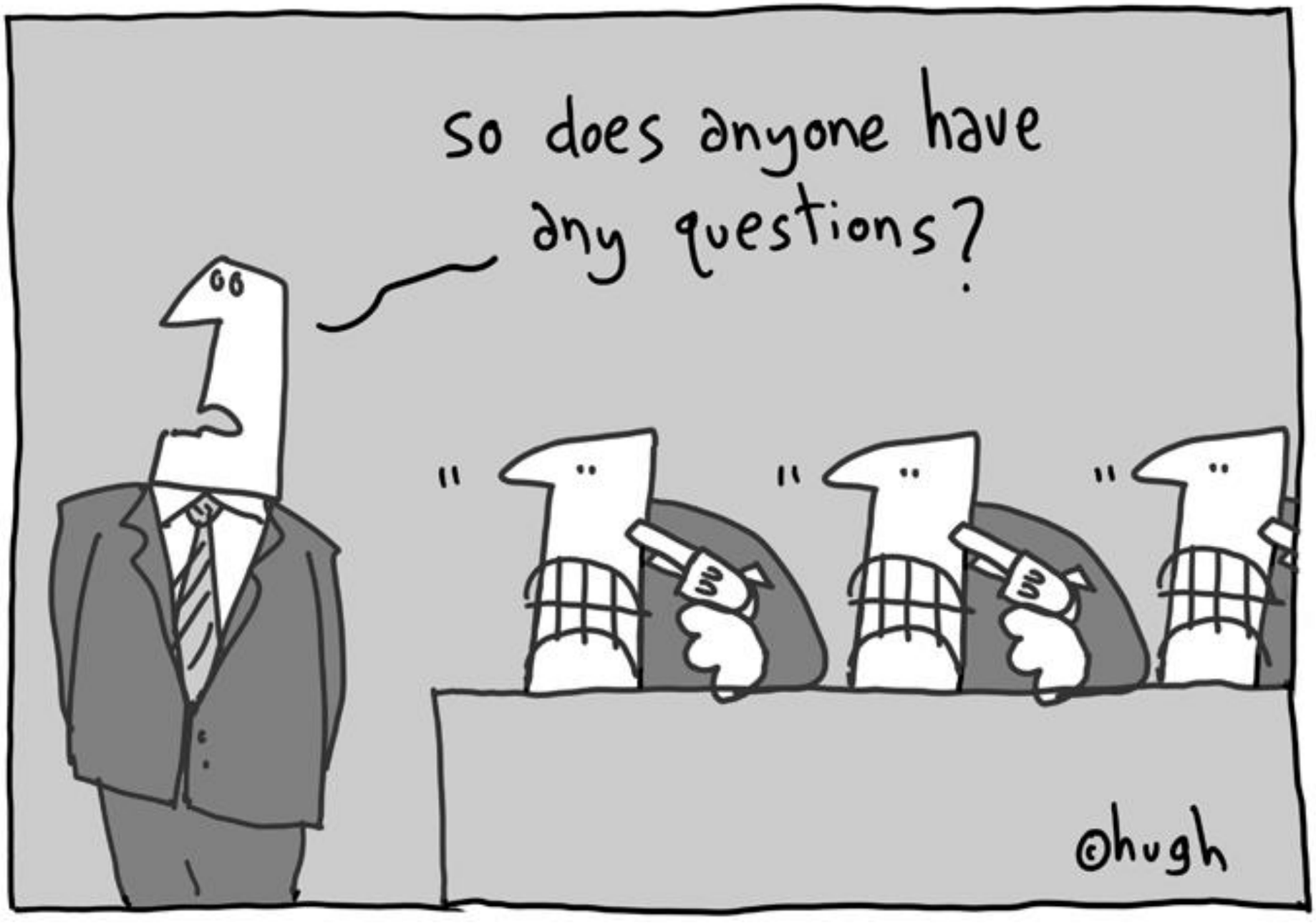
Overview of discussed techniques

Technique	Have I used in real applications?	Do I recommend technique?
Lombok	yes	maybe
AutoValue	yes	yes
CGLib BeanGenerator	no	no
JCodeModel	yes	yes
AspectJ (patching)	yes	maybe
JavaParser	yes	maybe
ErrorProne	no	maybe
Checker Framework	no	yes
AspectJ (policy enforcement)	yes	yes

- Obviously, this is an opinion, not advice. Use at your own risk.
- Author is not affiliated with any of the above libraries.

Where to go from here

- Look at the github code, send me a PR if you have improvements
- Contact me if you have questions
- Suggest more techniques / use cases



Source: http://www.gapingvoidart.com/gallery/any-questions/

JOIN OUR TEAM!

tech.zalando.com/jobs

github.com/zalando

@ZalandoTech


THE PRESENTATION ENDED
THANKS FOR YOUR ATTENTION