



STACKFRAME

TorqueWrench

Gene McCulley
mcculley@stackframe.com

StackFrame, LLC

- Small Company
 - 2 founders, 6 employees total
- Software Development
 - Java, C/C++, Ada
- Network Administration
 - Linux

Customers/Projects



CESI



Dignitas Technologies, LLC

 Applied Research Associates, Inc.



TorqueWrench

- Static analysis of Java bytecode
- Warns about violations of some coding practices
- Warns about potential problems in code base
- Intended to assist continuous integration
- Generates reports in HTML

Deployment/Installation

- Requires JRE 1.4 or higher
- Implemented entirely in Java
- Can be invoked multiple ways
 - Command line
 - Ant task
 - Eclipse (coming soon)

Philosophy

- Intended to help find and eliminate the problems that plague large projects
- Lots of effort goes into minimizing false positives
- Interfaces more important than implementation

Availability

- Not yet released
- In use by various projects
- Evaluation licenses available

Advantages of bytecode analysis

- No source needed
- Doesn't care what kind of compiler is used
- Doesn't care what language variant is used
- Can be used on classes for which no source is available

Disadvantages of bytecode analysis

- Warnings susceptible to false positives
 - Compiler optimizations obscure intent
- Disconnect from source

Architecture

- StackFrame developed bytecode parsing engine
 - In use by other projects
- Checkers plug in through extension mechanism
- Checkers relatively easy to build

Warnings

Abstract class should be an interface (asci)

- Lets you know when an abstract class defines no methods of its own
- Often a result of refactorings
- Changing to an interface allows clients more flexibility

ArrayIndexOutOfBoundsException caught (aibc)

- Sure sign of a very dubious coding practice:

```
...
try {
    return values[++i];
} catch (ArrayIndexOutOfBoundsException aibc) {
    i = 0;
    return values[i];
}
```

Wasteful constructors invoked

- Boolean constructor invoked (bci)
 - `new Boolean(false)` vs. `Boolean.FALSE`
- String void constructor invoked
 - `String s = new String();`
- String(String) constructor invoked
 - `String s = new String("foo");`

Class implements constant interface pattern (cifp)

- Some folks actually recommended this at one time to get around unwieldy namespace syntax

- ```
public interface Color {
 public static final int RED = 1;
 public static final int GREEN = 2;
 public static final int BLUE = 3;
}
```

```
public class Stoplight implements Color {
 ... // doesn't really implement Color
}
```

- Some projects disallow this in favor of enum pattern

# Cloneable

- Cloneable does not call `super.clone()` (cnsc)
- Cloneable does not promote `clone()` (cdnp)
- Cloneable throws `CloneNotSupportedException` (ctcn)
- Defines `clone()` but is not cloneable (dcnc)

# Swallowed Exceptions

- Warns when code catches an exception and then does nothing with it
- This can cause problems for client code
- Some projects rightly disallow this
- Some special cases excluded (e.g. `InterruptedException`)

# ConcurrentModificationException caught (cmec)

- Another dubious practice
- Generally indicates misunderstanding of thread safety and Collections

# equals() and hashCode()

- Defines equals() but not hashCode()  
(ewhc)
- Defines hashCode() but not equals()  
(hcwe)

# Duplicate method (dupm)

- Finds suspicious methods with identical code
- Usually a result of cut and paste

# finalize()

- Empty finalize method (efnm)
- finalize() invoked (fini)
- finalize() not protected (fmnp)

# Exception class needs chaining constructor (ecnc)

- i.e. Exception classes should have a constructor that takes a Throwable as a cause
- Some projects have this as a rule so that all client code can easily preserve the stack trace when rethrowing

# Externalizable

- Externalizable has Serializable methods (ehsm)
- Externalizable needs void constructor
- Usually a result of misunderstanding how Externalizable works or evolution from Serializable

# Field eclipses field in superclass (fesc)

- Class defines field with same name as superclass
- Usually result of concurrent evolution
- Can cause hard to find errors

# Maximum Cyclomatic Complexity Exceed (mcce)

- Uses the McCabe metric
- Threshold value is user-defined (defaults to 15)
- Suggests that a method should be broken into multiple helper methods

# Member exposes concrete Collection class

- Public methods taking or returning ArrayList instead of List & HashMap instead of Map

```
public ArrayList getDeathRays(); // bad
public List getDeathRays(); // better
```

- Better to hide implementation from client code

# IllegalMonitorStateException caught (imse)

- Another dubious practice
- Usually indicates misunderstanding of `synchronized/wait()/notify()`

# virtual/static

- Inner class should be static (icss)
  - Triggered when inner class never references instance of enclosing class
  - Can improve performance/clarity/memory use
- Virtual method should be static (vmss)
  - Can improve performance/clarity



# Write only

- Write only field (wof)
- Write only local variable (wolv)

# Unused

- Unused local variable (unlv)
- Unused method (uum)
- Unused field (uuf)

# Method looks like constructor (mlc)

- Common result of accident/  
misunderstanding of constructor syntax

```
public class Foo {

 public void Foo() {
 importButNonObvious();
 }

}
```

# Member exposes non-visible type (envt)

- ```
public class Foo {  
  
    class InnerStuff {  
    }  
  
    public void doSomething(InnerStuff i) {  
    }  
  
}
```

Public fields

- Public instance field (pif)
- Public static fields (psf)
- Useful for some small areas, usually disallowed on big projects

Return value ignored

- Done for special cases where it is obviously wrong (e.g. immutable types such as String)

```
s.trim();  
return s;
```

Self assignment

- Self assignment of class field (sacf)
- Self assignment of instance field (saif)
- Self assignment of local variable (salv)
- Sometimes caused by typos and can be hard to find

Demo

Questions?