



# RejmiNet

## *Testing Java™ Code: Beyond the IDE*

**Ian F. Darwin**  
RejmiNet Group Inc.  
<http://www.rejmi.net/>

TS-9667

# Goal of This Talk

## What You Will Learn

Learn to **improve Java™ technology program correctness in one session!**

- Beyond checking that Java Development Kit (JDK™)/IDEs do
- Using free, open source tools

Partly drawn from my new O'Reilly e-book *Checking Java Programs*  
<http://cjp.darwinsys.com/>

IDE = Integrated Development Environment

# Why More Checking?

Because We Can, and Should

- Java platform's original goals included **reliability**
  - Why there is as much checking as there is
  - Garbage collection, threading support, ...
  - Compile time and run time checking
- But it's never enough
  - Software still has bugs, right?
  - Standard tools do not look at patterns, nor at many potential sources of failure
  - So we bring in **extra tools**
  - Must couple with developer education!

# What's Wrong With This Code? (Five Second Test)

```
// Part of SaveAction.actionPerformed()
if (fileName != null && !doingSaveAs) {
    doSave(fileName);
    return;
}
int returnVal = chooser.showOpenDialog(this);
if (returnVal == JFileChooser.APPROVE_OPTION) {
    File file = chooser.getSelectedFile();
    if (file.exists() && doingSaveAs) {
        int ret =
            JOptionPane.showConfirmDialog(theFrame,
                "File exists, overwrite?", "Overwrite?",
                JOptionPane.YES_NO_OPTION);
        if (ret != 0); // "Yes" is the 0th option...
            return;
    }
    doSave(file);
}
```

Developers can spend up to 50% of their time understanding code before they can fix it...

Error here:  
file does not get saved!

# What Tools?

## My “Big Three” Add-On Tools

- PMD
  - Source code analyzer
- FindBugs
  - Class file analyzer
- NASA Java technology-based PathFinder (JPF)
  - Runtime state verifier
- And a few more for good measure



# Agenda

**PMD**

FindBugs

JPF

Other

# PMD Checks Your Java Source



- PMD reads source code looking for patterns
- Performs its own parsing to an AST (tree)
  - Not a full compile, so can survive, e.g., missing imports
  - Tests match patterns in the AST
  - Extensible via XPath or Java code
- Can be run standalone, with Eclipse or NetBeans™ IDE, Ant or Maven, and other tools (even emacs!)
- Get PMD from:
  - <http://pmd.sourceforge.net/>

Like the name 'Java', PMD is not an acronym.

# Running PMD Interactively

- Use provided `pmd.sh` or `pmd.bat`  
`pmd.sh $home/javasrc html basic,unusedcode`
- Script/batch or Ant task need three arguments:
  - Directory or Java Archive (JAR) file (or one Java class file)
    - Ant more flexible: use standard `<fileset>`
  - Report format: `text`, `html`, `htmlsummary`, `xml`
  - Set of rules to run (e.g., “`basic,imports,unusedcode`”)
- Optional arguments: see documentation
- Ant also requires `<taskdef>`



# PMD Works With Ant

```
// Part of cjp/build.xml
<target name="pmd">
  <taskdef name="pmd"
    classname="net.sourceforge.pmd.ant.PMDTask"
    classpathref="full.classpath"/>
  <pmd shortFileNames="true">
    <ruleset>basic,imports</ruleset>
    <formatter type="text"
      toFile="pmd-results.txt"/>
    <fileset dir="${src}">
      <include name="**/*.java"/>
    </fileset>
  </pmd>
</target>
$ ant pmd; more pmd-results.txt
SaveAction.java:30      An empty statement (semicolon) not
part of a loop
SaveAction.java:30      Avoid empty if statements
```

# PMD Works in Your IDE

- PMD under an IDE lets you run:
  - On demand
  - Automatically
- On-demand gives you more flexibility
  - In Eclipse, shows own “Perspective” :-)
- Automatically is more reliable
  - No “I forgot” excuses

# PMD Warnings

- Run PMD on a large project
  - Hundreds of warnings!
- Throttle back by:
  - Options (GUI or CLI)
  - Code Markers
- Code markers are:
  - `// NOPMD` on the line that generates the warning
  - `@SuppressWarnings("PMD.SomeWarningName");`

# Extending PMD

- PMD has lots of rules already
- Easy to add your own in XPath or Java platform
- Has visual tool for exploring AST from code bits
- e.g., ban library code throwing SQLExceptionz  

```
//MethodDeclaration/NameList/Name[@Image='SQLException']
```
- Embed in a 60-line XML file
  - Download CJP book example (see last page)

# PMD's Weapon of Mess Detection: CPD

- Copy-and-Paste bloats code, prevents reuse
  - Easy to commit, hard to ferret out
- CPD finds it, even with variable name changes
- Example from JDK software:

Found a **294 line (531 tokens)** duplication:

Starting at line 486 of `src/java/lang/StrictMath.java`

Starting at line 575 of `src/java/lang/Math.java`


```
public static int round(float a) {  
    return (int)floor(a + 0.5f);  
}
```

...etc...

# Running CPD

- Command line or Ant task
- Specify min tokens
  - Too low matches getters
  - Too high causes false negatives
  - Start around 100 to avoid getter/setter

# CPD Details

- Third implementation, uses Karp-Rabin algorithm
  - Does JDK software classes source in under 5 seconds 
- Works on Java technology, JavaServer Pages™ (JSP™), C, C++, and PHP code
  - Use -language option if not Java code



# DEMO

PMD







# Agenda

PMD

**FindBugs**

JPF

Other

# FindBugs Digests Your Class Files



- From University of Maryland
- Checks .class files (using Apache BCEL)
- Wide range of checks
  - A few optionally examine source code
- Get FindBugs from:
  - <http://findbugs.sourceforge.net/>

The name FindBugs and the FindBugs logo are trademarks of the University of Maryland.

# Running FindBugs

- Choose **level**: low (most verbose), medium, high
- Run from:
  - Command line (three dozen options!)  
`findbugs -medium -textui $HOME/javasrc`
  - Ant-Similar to PMD: taskdef, invoke it
  - Eclipse; NetBeans IDE (3<sup>rd</sup> party)
    - Automatic or on-demand; uses Java technology Perspective; own marker
  - Its own GUI
    - Flexible runner GUI
    - Can create project file for later use with CLI

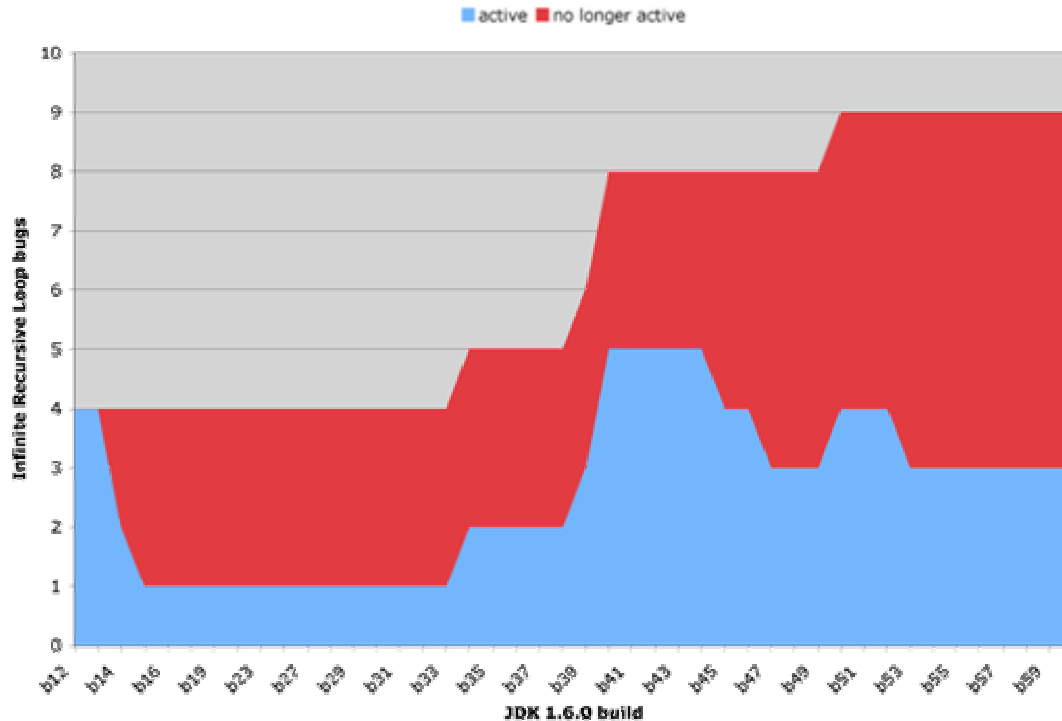
# Throttling Back FindBugs

- Use high priority option
- Use an XML excludes file
  - Individual bugs
  - Entire categories: Bad practice, correctness, etc.
- Use Annotations
  - Has own `@SuppressWarnings` (in own package)

# FindBugs Data Mining



- Supports storing longitudinal data in report files
- Allows for building reports; data for graphing



From the FindBugs web site, used under the Creative Commons Attribution License



# DEMO

## FindBugs



# Interlude: PMD and FindBugs

- PMD checks source code, FindBugs checks compiled .class files
- Overlap in what they find is only about 50%
  - Worth running both



# Agenda

PMD

FindBugs

**JPF**

Other



# JPF Runs Your Code



- Developed at NASA for testing software used in rocket control systems
  - No margin for error!
- **Dynamic state checker**
  - Can test all paths through code by “backing up and trying again”
- Includes own Java Virtual Machine (JVM™)—implemented in Java technology
- Get JPF from:
  - <http://javapathfinder.sourceforge.net/>

# JPF Finds Deadlocks

- Good news
  - Can find threading deadlocks
  - Can find “race conditions”
- Bad news
  - JVM interface does not yet support AWT or sockets
    - Rockets don't have a GUI :-)

# Running JPF

- Actually best to check out and build from svn

```
svn checkout https://svn.sf.net/svnroot/javapathfinder/trunk  
build-tools/bin/ant run-tests jar
```

- Then run jpf shell script or batch file
  - `export CLASSPATH=myproject.jar`
  - `jpf MyMainClass`

# JPF Ant

- Can run under Ant
- No taskdef—Just run `java gov.nasa.jpj.JPF`
- Must provide `<classpath>` including your classes and JPF jars
- **MUST** specify `fork=true`

# JPF IDE

- No IDE plug-ins at present
- SVN repository is an Eclipse project
  - Just tell Eclipse about it
  - Or even do initial checkout with Subclipse
- Advice: Use Eclipse “Variables” to refer to “External JARs” from a normal project

# JPF and States

- Every method call potentially changes state
- Assume you call `Random.nextInt(10)`
  - Has 10 possible outcomes; might be used in `switch`
  - Normal testing (JUnit?) will exercise only one
- JPF provides custom API for this:
  - `Verify.random(10);`
  - Under JPF will enumerate *all states*
  - *Run under “normal” JVM software will act like `nextInt(10)`*

# JPF and Thread Deadlocks

- Assignment
  - On paper or on your laptop, write a **very** short program that will result in a thread deadlock
    - Condition when no non-daemon threads are runnable

```
public class InstantDeadlock {  
    public static void main(String[] args)  
        throws Exception {  
        Thread.currentThread().join();  
    }  
}
```

# Finding Thread Deadlocks

- That one's easy to find
  - Unless buried in 150KLOC
- Real-world thread deadlocks much harder to find
- JPF catches them quickly



# Thread Deadlock Found

```
===== snapshot #1
thread index=0,name=main,status=WAITING,this=java.la
ng.Thread@3,target=null,priority=5,lockCount=1
waiting on: java.lang.Thread@3
call stack:
    at java.lang.Object.wait(Object.java:474)
    at java.lang.Thread.join(Thread.java:190)
    at InstantDeadlock.main(InstantDeadlock.java
:8)
===== results
error #1: gov.nasa.jpf.jvm.NotDeadlockedProperty
```

# JPF Summary

- Good tool for finding certain classes of errors at runtime
  - State enumeration provides coverage
  - Deadlock checking useful in Threads
- Requires more setup than PMD, FindBugs
- Majorly extensible and configurable
  - Not in a 50-minute talk—see documentation



# Agenda


PMD

FindBugs

JPF

**Other**

# Pushing Javac

- Standard javac does checking required by JLS
- More semi-supported warnings with -Xlint
- e.g, javac -Xlint:path 
  - Warn about non-existent JARs on classpath!
- Half a dozen others (some toggles)
  - Read current JDK software doc for details!

# Pushing the IDE

- Eclipse, NetBeans IDE can do considerable checking
- Eclipse
  - Enable project-specific settings
  - Eclipse will save these settings to project CVS/SVN

# JUnit

“Never...have so many owed so much to so few lines of code”

- JUnit is **the** best-known Unit Testing framework for Java technology; every coder should use it
- I would love to talk about it for an hour, or for **two days**
- If you don't already use JUnit, download it from <http://www.junit.org/> and order a copy of J.B. Rainsberger's *JUnit In Action*
  - **Just do it!**

# No Shortage of Other Tools

- Jlint
  - Venerable code analyzer
- VerifyDesign
  - Checks that code only uses allowed type
  - Supporting program-to-interface
- Jikes
  - IBM open source compiler with lots of warnings
- More!
  - See <http://pmd.sourceforge.net/similar-projects.html>

# For More Information

## See also:

- Related sessions
- Vast literature on software quality!
- **My book: *Checking Java Programs***  
(O'Reilly "Short Cut" series)
  - See <http://cjp.darwinsys.com/> for examples download
  - <http://www.oreilly.com/catalog/9780596510237>
- PMD: <http://pmd.sourceforge.net/>
- FindBugs: <http://findbugs.sourceforge.net/>
- JPF: <http://javapathfinder.sourceforge.net/>
- JUnit: <http://www.junit.org/>



# Summary

- PMD: Static source code checker
- FindBugs: Static byte-code checker
- JPF: Dynamic model checker
- More important: ***Tool builder/explorer mindset!***

**May the Source be with  
But not ~~you~~ bugs!**



# Q&A

Ian Darwin

<http://cjp.darwinsys.com/>



# RejmiNet

## *Testing Java Code: Beyond the IDE*

**Ian F. Darwin**  
RejmiNet Group Inc.  
<http://www.rejmi.net/>

TS-9667