FindBugs
review of Glassfish v2 b09

William Pugh
Univ. of Maryland

http://www.cs.umd.edu/~pugh/
FindBugs

- Open source static analysis tool for finding defects in Java programs
- Analyzes classfiles
- Generates XML or text output
  - can run in Netbeans/Swing/Eclipse/Ant/SCA
- Total downloads from SourceForge: 231,861+
What is FindBugs?

- Static analysis tool to find defects in Java code
- not a style checker
- Can find hundreds of defects in each of large apps such as Bea WebLogic, IBM Websphere, Sun's JDK
  - real defects, stuff that should be fixed
  - hundreds is conservative, probably *thousands*
- Doesn’t focus on security
- lower tolerance for false positives
Common Wisdom about Bugs

• Programmers are smart

• Smart people don’t make dumb mistakes

• We have good techniques (e.g., unit testing, pair programming, code inspections) for finding bugs early

• So, bugs remaining in production code must be subtle, and require sophisticated techniques to find
Would You Write Code Like This?

```java
if (in == null)
    try {
        in.close();
    }
    ...

• Oops

• This code is from Eclipse

• You may be surprised what is lurking in your code
```
Why Do Bugs Occur?

- Nobody is perfect

- Common types of errors:
  - Misunderstood language features, API methods
  - Typos (using wrong boolean operator, forgetting parentheses or brackets, etc.)
  - Misunderstood class or method invariants

- Everyone makes syntax errors, but the compiler catches them

- What about bugs one step removed from a syntax error?
JDK 1.6.0-b92 results

• 44 classes that define equals() but inherit hashCode() from Object
• 31 equals methods that don’t handle null
• 6 statements that always throw a NPE
• 46 branches that if taken guaranteed a NPE
• 11 comparisons of unrelated types
• 7 ignored return values
• 1 infinite recursive loop
Demo

- Live code review of glassfish-v2-b09
- Available as Java Webstart from
Bug Patterns
Hashcode/Equals

- Equal objects must have equal hash codes
- Programmers sometimes override equals() but not hashCode()
  - Or, override hashCode() but not equals()
- Objects violating the contract won’t work in hash tables, maps, sets
- Examples (53 bugs in 1.6.0-b29)
  - javax.management.Attribute
  - java.awt.geom.Area
Fixing hashCode

• What if you want to define equals, but don't think your objects will ever get put into a HashTable?

• Suggestion:

```java
public int hashCode() {
    assert false : "hashCode method not designed";
    return 42;
}
```
Null Pointer Dereference

• Dereferencing a null value results in NullPointerException

• Warn if there is a statement or branch that if executed, guarantees a NPE

• Example:

  // Eclipse 3.0.0M8
  Control c = getControl();
  if (c == null && c.isDisposed())
    return;
Bad Binary operations

if ((f.getStyle() & Font.BOLD) == 1) {
    sbuf.append("<b>");
    isBold = true;
}

if ((f.getStyle() & Font.ITALIC) == 1) {
    sbuf.append("<i>");
    isItalic = true;
}
public static final ASDDVersion getASDDVersion(BigDecimal version) {
    if(SUN_APPSERVER_7_0.toString().equals(version))
        return SUN_APPSERVER_7_0;
}
Unintended regular expression

String[] valueSegments
    = value.split("."); // NOI18N
public TagHelpItem(String name, String file,
    String startText, int startOffset,
    String endText, int endOffset,
    String textBefore, String textAfter){
    this.name = name;
    this.file = file;
    this.startText = startText;
    this.startTextOffset = startTextOffset;
    this.endText = endText;
    this.endTextOffset = endTextOffset;
    this.textBefore = textBefore;
    this.textAfter = textAfter;
    this.identical = null;
}
Confusing/bad naming

• Methods with identical names and signatures
  – but different capitalization of names
  – could mean you don’t override method in superclass
  – confusing in general

• Method name same as class name
  – gets confused with constructor
Bad naming in Eclipse

package org.eclipse.jface.dialogs;
public abstract class Dialog extends Window {
    protected Button getOKButton() {
        return getButton(IDialogConstants.OK_ID);
    }
}

public class InputDialog extends Dialog {
    protected Button getOKButton() {
        return okButton;
    }
}
Bad naming in BCEL (shipped in jdk1.6.0-b29)

/** @return a hash code value for the object. */

public int hashcode() {
    return basic_type.hashCode() ^ dimensions;
}
Read Return Value Ignored

- InputStream.read() methods that read into a byte array return the number of bytes read
  - Can be less than the number requested
  - Programmers sometimes fail to check return value
    - May result in uninitialized array elements being used
    - Program can get out of sync with input stream

- Example (GNU Classpath 0.08):
  ```java
  // java.util.SimpleTimeZone.readObject()
  int length = input.readInt();
  byte[] byteArray = new byte[length];
  input.read(byteArray, 0, length);
  ```
Other Return Value Ignored Errors

• Lots of methods for which return value always should be checked
  – E.g., operations on immutable objects

• Examples:

  // Eclipse 3.0.0M8
  String name = workingCopy.getName();
  name.replace('/', '.');;
**/ * javax.management.ObjectInstance * reference impl., version 1.2.1 **/

    public ObjectInstance(ObjectName objectName,
                          String className) {
        if (objectName.isPattern()) {
            new RuntimeOperationsException(
                new IllegalArgumentException(
                    "Invalid name->"+ objectName.toString()));
        }
        this.name = objectName;
        this.className = className;
    }
Inconsistent Synchronization

• Common idiom for thread safe classes is to synchronize on the receiver object ("this")
• We look for field accesses
  – Find classes where lock on "this" is sometimes, but not always, held
  – Unsynchronized accesses, if reachable from multiple threads, constitute a race condition
Inconsistent Synchronization Example

- GNU Classpath 0.08, java.util.Vector

```java
public int lastIndexOf(Object elem) {
    return lastIndexOf(elem, elementCount - 1);
}

public synchronized int lastIndexOf(
    Object e, int index)
{
    ...
}
```
Unconditional Wait

• Before waiting on a monitor, the condition should be almost always be checked
  – Waiting unconditionally almost always a bug
  – If condition checked without lock held, could miss the notification

• Example (JBoss 4.0.0DR3):

```java
if (!enabled) {
  try {
    log.debug(...);
    synchronized (lock) {
      lock.wait();
    }
  }
}
```

condition can become true after it is checked

but before the wait occurs
Warning Density
## Warning density

- Density of high and medium priority correctness warnings (excluding HE and SE warnings)

<table>
<thead>
<tr>
<th>Warnings/KNCSS</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>SleepyCat DB</td>
</tr>
<tr>
<td>0.5</td>
<td>Eclipse 3.2</td>
</tr>
<tr>
<td>0.9</td>
<td>JDK 1.5.0_03</td>
</tr>
<tr>
<td>1.0</td>
<td>JDK 1.6.0 b51</td>
</tr>
<tr>
<td>1.3</td>
<td>WebSphere</td>
</tr>
</tbody>
</table>
Some new-ish features
Behavior Annotations

• Allow you to provide lightweight specifications through Java 5.0 annotations

• Examples
  • `@NonNull`
  • `@CheckForNull`
  • `@CheckReturnValue`
  • `@Tainted/@Untainted/@Detainted`
  • proposed
Computing bug history

- Keeps track of when bugs are introduced, when they are resolved
- Historical bug data records all bugs reported for any build
- Can see when bugs were introduced and removed
- For example, can report all bugs introduced in the past 3 months
FindBugs
Best Practices
What to look at

- First review high and medium priority correctness
- Low priority warnings are of questionable value
- Other categories (style, performance) worth examining in a code review, but insisting that they all be reviewed immediately will make people unhappy
- Carefully consider and review FindBugs plugins
- Others have written plugins, some of which generate a lot more false positives or give bad advice
Incremental analysis and/or marking

- For sustainable use, you need to have some way to deal with false positives
- mark in database
- Only review new warnings
- Both of these require matching warnings from one analysis with results from a previous analysis
Developers like incremental analysis

- Developers don’t like to be asked to scrub a million line code base and review 1000 warnings
- But they don’t mind (as much) if you ask them to review a new warning introduced by a change they just made
- False positive rate still matters
Questions?