CISC 475/675: Advanced Software Engineering

Lab 10:
Code Coverage with

Cobertura

Charlie Greenbacker
University of Delaware
Spring 2009

Overview

- What is Code Coverage?
- Cobertura at a Glance
- Installing Cobertura
- Running Cobertura
- Analyzing Cobertura Results
- Lab “Exercise”

What is Code Coverage?

- A measure of the degree to which a piece of source code has been tested
- Analyzes unit testing to determine exactly which classes, methods, & individual lines of code have been executed during the test
- Enables programmers to identify which parts of their programs are lacking test coverage
- Note: doesn't tell you how good your tests are, just whether all parts of your code are touched
  - Bad tests ***can*** have good coverage, but good tests ***will*** have good coverage

Cobertura at a Glance

- Open source Java code coverage analysis tool
  - Calculates % of code exercised by unit tests
  - Designed to work closely with Ant & JUnit
  - Helps you find "holes" in your testing so that you can create new & more thorough test cases
  - Automatically generates detailed reports
- Works by inserting temporary instrumentation into your classes to track which methods & lines of code are accessed during testing
- Name is "coverage" in Spanish/Portuguese

Installing Cobertura

- Installing Cobertura is as easy as downloading & unzipping the distribution package
  - http://cobertura.sourceforge.net/download.html
  - Move/rename the unzipped directory to a location of your choosing for future access
    - For example, /usa/<username>/cobertura is a good name & location for the Cobertura directory

Running Cobertura

- Cobertura runs as a custom Ant task and interacts with JUnit testing
- Adding/editing a few targets, plus some odds & ends, in the existing build.xml file from Lab 9 for the sample Student project will get us started
- First, add a couple of new properties to the top:

```
<property name="instrumented"
  location="instrumented"/>
<property name="cobertura"
  location="/path/to/cobertura"/>
```

  - Change the “cobertura” property to point to whichever directory contains cobertura.jar, etc.

Running Cobertura (cont...)

- Next, we'll set up a classpath to allow Ant to access the necessary Cobertura files:

```
<path id="cobertura.classpath">
    <fileset dir="${cobertura}">
        <include name="cobertura.jar"/>
        <include name="lib/**/*.jar"/>
    </fileset>
</path>

<taskdef classpathref="cobertura.classpath"
  resource="tasks.properties"/>
```

Running Cobertura (cont...)

- We'll need to modify the build target so that javac can access the Cobertura classpath:

```
<target name="build" depends="init">
    <javac srcdir="${source}" destdir="${build}"
      debug="yes">
        <classpath refid="cobertura.classpath"/>
    </javac>
</target>
```

Running Cobertura (cont...)

- We'll add a couple of actions to the init target:

```
<target name="init" depends="clean">
    <mkdir dir="${build}"/>
    <mkdir dir="${reports}"/>
    <mkdir dir="${reports}/raw/"/>
    <mkdir dir="${reports}/html/"/>
    <mkdir dir="${reports}/cobertura-html/"/>
    <mkdir dir="${instrumented}"/>
</target>
```

Running Cobertura (cont...)

- And some new actions to the clean target too:

```
<target name="clean">
    <delete dir="${build}"/>
    <delete dir="${reports}"/>
    <delete dir="${instrumented}"/>
    <delete file="cobertura.log"/>
    <delete file="cobertura.ser"/>
</target>
```

Running Cobertura (cont...)

- The first new target will add instrumentation to the Java classes to determine which lines of code have been executed and which have not:

```
<target name="instrument" depends="init,build">
    <delete file="cobertura.ser"/>
    <delete dir="${instrumented}"/>
    <cobertura-instrument
      todir="${instrumented}">
        <ignore regex="org.apache.log4j.*"/>
        <fileset dir="${build}">
            <include name="**/*.class"/>
            <exclude name="**/*Test*.class"/>
        </fileset>
    </cobertura-instrument>
</target>
```

Running Cobertura (cont...)

- The next new target will actually run the Cobertura task to perform the code coverage analysis of the unit testing, using the instrumentation we just added:

```
<target name="coverage-check">
    <cobertura-check branchrate="34"
      totallinerate="100"/>
</target>
```

Running Cobertura (cont...)

- The next target generates the Cobertura report:

```
<target name="coverage-report">
    <cobertura-report
      destdir="${reports}/cobertura-html/">
        <fileset dir="${source}">
            <include name="**/*.java"/>
        </fileset>
    </cobertura-report>
</target>
```

Running Cobertura (cont...)

- We also need to modify the run-tests target to make JUnit aware of the instrumented classes and Cobertura files:

```
<target name="run-tests" depends="build">
    <junit printsummary="yes" haltonfailure="no"
      showoutput="yes">
        <classpath location="${instrumented}"/>
        <classpath location="${build}"/>
        <classpath refid="cobertura.classpath"/>
        <batchtest fork="yes" todir="${reports}/raw/">
            <formatter type="xml"/>
            <fileset dir="${source}">
                <include name="**/*Test*.java"/>
            </fileset>
        </batchtest>
    </junit>
</target>
```

Running Cobertura (cont...)

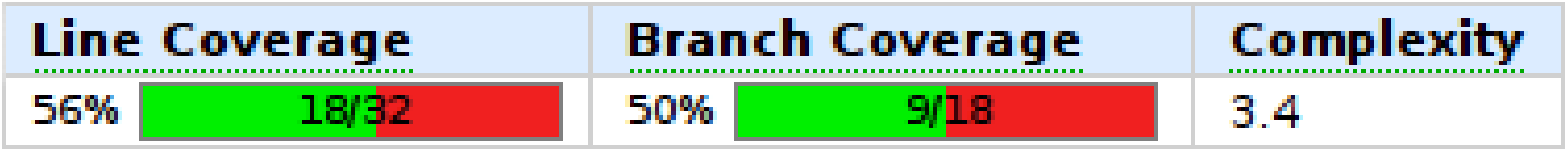
- The final new target is a wrapper for all of the Cobertura tasks & is the one we'll run by name:

```
<target name="coverage"
  depends="build,instrument,test,coverage-report"/>
```

- Now we can finally execute the Cobertura analysis by running `ant coverage` from the command line
  - Recall from the previous lab, you need to use the -lib ~/junit option on the EECIS lab computers, so the command would instead be:
    `ant -lib ~/junit coverage`

Analyzing Cobertura Results

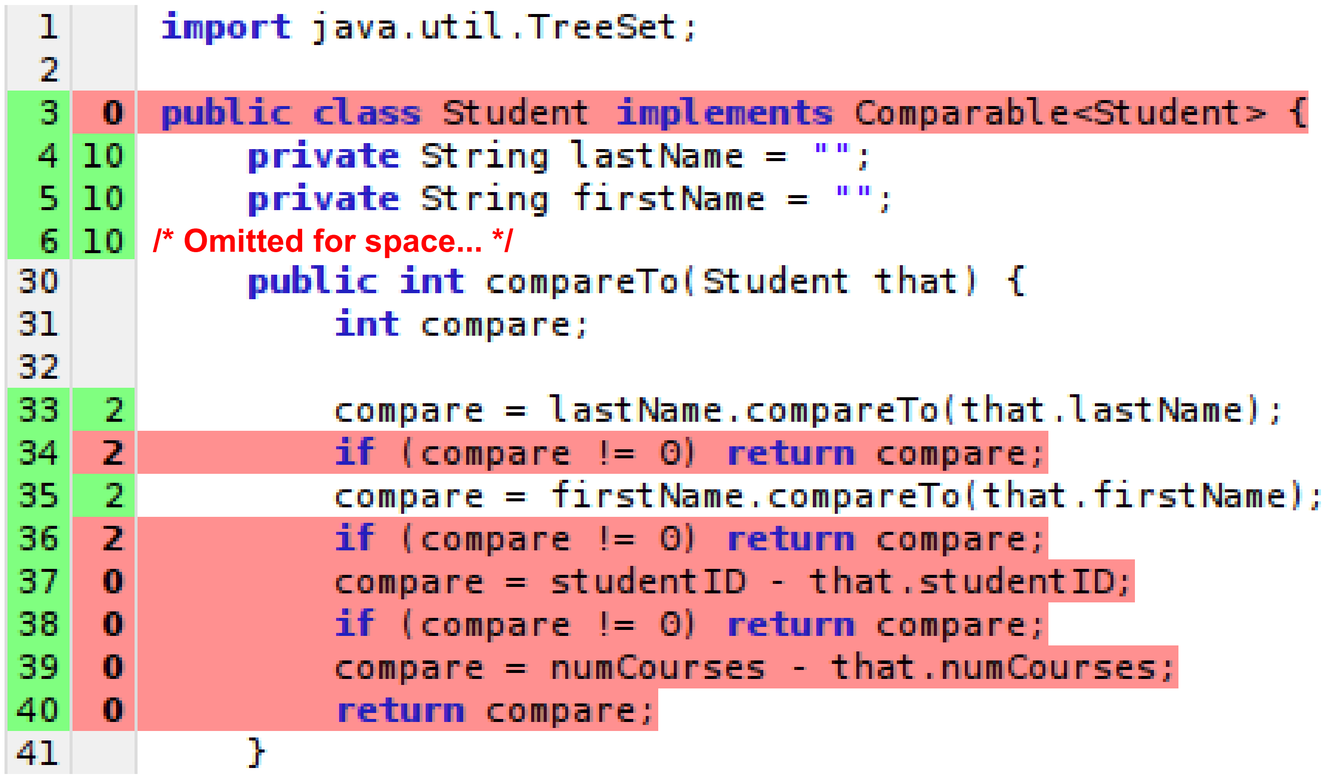
- Cobertura produces reports in HTML format
  - Open “reports/cobertura-html/index.html” to view
- Report summary will have 3 columns per class
  - Line Coverage = % of lines executed by test
  - Branch Coverage = % of logical branches executed
  - Complexity ≈ number of different paths per method



Line Coverage	Branch Coverage	Complexity
56% 18/32	50% 9/18	3.4

Analyzing Results (cont...)

- Clicking on a class link will bring up the individual coverage report for that class
  - Lines highlighted in red are those not executed, to help you create more tests to improve coverage



```
    import java.util.TreeSet;

 0  public class Student implements Comparable<Student> {
10      private String lastName = "";
10      private String firstName = "";
10  /* Omitted for space... */
        public int compareTo(Student that) {
            int compare;

 2          compare = lastName.compareTo(that.lastName);
 2          if (compare != 0) return compare;
 2          compare = firstName.compareTo(that.firstName);
 2          if (compare != 0) return compare;
 0          compare = studentID - that.studentID;
 0          if (compare != 0) return compare;
 0          compare = numCourses - that.numCourses;
 0          return compare;
        }
```

Analyzing Results (cont...)

- Live demonstration...

Analyzing Results (cont...)

- For example, the coverage report for the Student class indicates that the tests did not execute the single line of code inside toString(), as well as several branches within compareTo()
  - If we were to write a test case calling the Student.toString() method, and additional test cases designed to comprehensively exhaust the different branch conditions in Student.compareTo(), we'd see fewer lines highlighted in red, and the coverage scores would increase accordingly

Lab "Exercise"

- You've all got enough to work on for the class project right now, so I'm not going to add a lab assignment on top of that
- However, please DO try out the example for yourself (available on my website)
  - Try writing new test cases to modify the output
  - Consider using Cobertura to analyze CourseCheck
  - Let me know if you run into any problems/questions
- <u>Nothing to submit to me this week</u>