**SE401: Software Quality Assurance & Testing**

**Activity 5.2: Unit Testing and Ant**

# Requirements of the Lab Exercise

1. Write a JUnit test suit to adequately test the pluralize() method. You may use any combination of functional testing techniques to develop the test cases.
2. Use Ant to run the testing process, which includes compilation and the execution of the test suite.

**Test Cases**

**T1: NULL Word (FAIL):**

The program was returning a null object if the input was null; however, this can be fixed by checking before accepting the word if it is null or not and if null return an error message.

**T2: Special Characters (SUCCESS):**

The program was successfully throwing an exception in case of special characters which is totally correct.

**T3: Irregular Word (SUCCESS):**

The program was running successfully with irregular words that are registered in the program from the beginning. For instance, “man” will return “men” which is correct.

**T4: Test Numbers (FAIL):**

In the case of numbers as an input, the program was failing by returning “s” at the end of the numbers which is incorrect. For example, “1234” the program returns “1234s”.

**T5: Test Random Word (FAIL):**

The program was returning “s” at the end of the given word if it contains random numbers which is wrong. For instance, “asdes” the program returns “asdess”.

**T6: Test Plural (SUCCESS):**

If the input word was already plural, the program was returning the same word in its plural form which is 100% correct.

**T7: Capitalized Word (SUCCESS):**

The program was returning any output in small letters even if the input was capitalized. For example, “BAG” will return “bags”.

**T8: Test Uncountable Words (FAIL):**

The program was adding “s” to the words that are not defined from the beginning, so for the uncountable words that can’t be pluralized the program was adding “s” which is wrong. For instance, “money” the program returns “moneys”.

**T9: Test Numbers and Letters (FAIL):**

Similarly with the previous defects, the program was adding “s” if the given input contains a word and numbers. For example, “man123” the program returns “man123s” which is wrong.

**T10: Test Multiple Words (FAIL):**

If the program receives two words together it will return the two words with “s” at the end which is wrong. For instance, “boy girl” the program returns “boy girls”.

**Additional Test Cases:**

@Test

public void testAddition() {

assertTrue(pluralize("sheep").equals("sheep"));

assertTrue(pluralize("qunatity").equals("qunatities"));

assertTrue(pluralize(null).equals(null));

assertEquals("equipment", pluralize("equipment")); }

@Test

public void test2() {

assertTrue(pluralize("cars").equals("cars"));

}

**Defects:**

expected:<water[]> but was:<water[s]> expected:<bo[y]s> but was:<bo[ie]s> expected:<da[y]s> but was:<da[ie]s> expected:<wi[v]es> but was:<wi[f]es>

Unexpected exceptions

<java.lang.StringIndexOutOfBoundsException>

<java.lang.AssertionError> <java.lang.NullPointerException>

There are several defects that appeared during testing:

• D1. NULL word

• D2. Numbers input

• D3. Random letters

• D4. Uncountable words

• D5. Numbers and letters together

• D6. Multiple words input

**Proposed Fixes**

* Input validation
* Add more words to the defined list
* Handle spaces if found in the input

**Discussion:**

Line Coverage: 96%

The line that have not been covered is “public class StringUtil” which is the declaration of the class. Since it is a static method this line will not be executed.

Branch Coverage: 76%

The tool shows that “if (c1 == 'y'” in not been covered but one of my use cases is “family” which should cover this.

**Ant Build Script**

<project name="MyProject" default="run" basedir=".">

<description>

Ant build file

</description>

<!-- set global properties for this build -->

<property name="src" location="src"/>

<property name="build" location="bin"/>

<property name="cobertura" location="C:\cobertura"/>

<property name="instrumented" location="instrumented"/>

<property name="coverage.xml" location="coverage-xml"/>

<property name="coverage.html" location="coverage-html"/>

<path id="cobertura.classpath">

<fileset dir="${cobertura}">

<include name="cobertura.jar" />

<include name="lib/\*\*/\*.jar" />

</fileset>

</path>

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

<target name="init">

<!-- Create the time stamp -->

<tstamp/>

<!-- Create the build directory structure used by compile -->

<mkdir dir="${build}"/>

</target>

<target name="compile" depends="init"

description="compile the source " >

<!-- Compile the java code from ${src} into ${build} -->

<javac srcdir="${src}"

destdir="${build}"

debug="on"/>

</target>

<target name="test1" depends="compile">

<junit printsummary="yes" fork="yes" haltonfailure="yes">

<classpath location="${build}"/>

<formatter type="plain"/>

<test name="edu.depaul.se433.StringUtilTest"/>

</junit>

</target>

<target name="instrument" depends="init,compile">

<!--

Remove the coverage data file and any old instrumentation.

-->

<delete file="cobertura.ser"/>

<delete dir="${instrumented}" />

<!--

Instrument the application classes, writing the

instrumented classes into ${build.instrumented.dir}.

-->

<cobertura-instrument todir="${instrumented}">

<!--

The following line causes instrument to ignore any

source line containing a reference to log4j, for the

purposes of coverage reporting.

-->

<ignore regex="org.apache.log4j.\*" />

<fileset dir="${build}">

<!--

Instrument all the application classes, but

don't instrument the test classes.

-->

<include name="\*\*/\*.class" />

<exclude name="\*\*/\*Test.class" />

</fileset>

</cobertura-instrument>

</target>

<target name="test2" depends="init,compile">

<junit fork="yes">

<!--

Note the classpath order: instrumented classes are before the

original (uninstrumented) classes. This is important.

-->

<classpath location="${instrumented}" />

<classpath location="${build}" />

<!--

The instrumented classes reference classes used by the

Cobertura runtime, so Cobertura and its dependencies

must be on your classpath.

-->

<classpath refid="cobertura.classpath" />

<formatter type="plain" />

<test name="edu.depaul.se433.StringUtilTest"/>

</junit>

</target>

<target name="coverage-report-xml">

<!--

Generate an XML file containing the coverage data using

the "srcdir" attribute.

-->

<cobertura-report srcdir="${src}" destdir="${coverage.xml}" format="xml" />

</target>

<target name="coverage-report-html">

<!--

Generate a series of HTML files containing the coverage

data in a user-readable form using nested source filesets.

-->

<cobertura-report srcdir="${src}" destdir="${coverage.html}"/>

</target>

<target name="coverage" depends="compile,instrument,test2,coverage-report-xml,coverage-report-html"

description="Compile, instrument ourself, run the tests and generate JUnit and coverage reports."/>

<target name="clean"

description="clean up" >

<!-- Delete the ${build} and ${dist} directory trees -->

<delete dir="${build}"/>

<delete dir="${instrumented}"/>

</target>

</project>