SAARLAND UNIVERSITY
COMPUTER SCIENCE

Universität des Saarlandes
Software Engineering Chair
Prof. Andreas Zeller
Gordon Fraser
David Schuler

3. Project: Test Case Generation and Delta Debugging

The task of this project is to (a) write a random test case generator, and (b) to apply delta debugging on test cases.

1 Get the project

You can check out the project from our SVN repository. Your user name is your matriculation number, and the password was sent to you by e-mail. The location of the repository is:
`https://prog2.cs.uni-saarland.de/debugging/students/project3/<MATR>`
At the end of the project (17th June, at 23:59) the version in the repository will be graded.

For this project you will need a Java version $\geq$ 1.6.

2 Overview

The *TestGenerator* produces a test by creating objects and invoking methods using the *Reflection API* and in parallel it also generates Java statements that can reproduce the test.

The *DeltaDebugger* takes a test case that fails with an Exception and applies *ddmin* in order to produce a minimized version of the test case that fails with the *same* Exception.

3 Test Generation

The class *TestGenerator* receives a class name for which it should generate one JUnit 4 test. This is done by generating instances of the class to test and invoking methods on them. Obtaining parameters for a method often involves recursively generating new objects.

In order to get information about the methods and constructors of a class, you can use the Java Reflection API. There are several tutorials on the net for using reflection, e.g.
`http://www.ibm.com/developerworks/library/j-dyn0603/`

Among others, the following properties get passed to the *TestGenerator*:

test.dir The directory the generated test is written to. This is only the base directory. Depending on the package name of the test the actual directory of the test has to be a subdirectory.

test.name The name of the test. The resulting class should have this name, and the name of the produced file is this name with ".java" appended.

test.package The package name of the test.

3.1 Generating Objects

Instances of objects can be obtained by either invoking the constructor, a static method, or non-static methods of other classes, or obtaining a reference from a static field. Primitive parameters and Strings can be generated directly using a random generator. All other objects have to be generated recursively.

3.2 Object Pool

The implementation should reuse created objects with a given probability. Therefore, an object pool is used. All objects that are created by the test generator are passed to the object pool. When objects are needed the object pool can be queried for existing objects. The system property *object.reuse* gives the probability for reusing objects. It ranges from 100 for always trying to reuse an object from the pool to 0 for never reusing an object. When there is no object of the needed type in the pool, it of course has to be created first. On the other hand when there is no way to create a new object, an object from the pool should be used (even when reuse is set to 0). If no object can be generated for a class that is needed as an argument, *null* can be used.

3.3 Handling Interfaces

One challenge is to create instances for abstract classes and interfaces. In Java there is no easy way to get all implementors of a class or interface. Therefore, this information has to be computed in advance. The test generator should thus check which interfaces the classes under test and the return values of their methods implement. Furthermore, the system property *classes.file* gives the location of a text file that contains a list of classes that should also be considered as implementations of their interfaces and super classes. The file format is one fully classified class name per line, see *src/main/resources/classes-list.txt* for an example. For example if an object of type *Map* is needed, the test generator should generate a *HashMap* if this class is in the text file.
Note: When generating objects all classes in the *classes.file*, and all classes to test, and the super-classes of both should be considered as sources for objects.

3.4 Excluded Methods

Some methods of an object should not be considered for testing. These are the following methods:

- Methods that are defined in *java.lang.Object* and are not overwritten.
- Methods and constructors with the *synthetic* modifier.
- The *valueOf* -method of an *enum*.

3.5 When to stop

The test should end if a given number (specified by the system property *test.limit*) of method and constructor calls to the class under test is reached. Within this limit all public methods and

constructors of the class under test should be called at least once with non null arguments if possible.

The second stopping criterion is defined Exceptions and is described in more detail below.

3.6 Exceptions

If an Exception is thrown by invoking a method or a constructor, the behavior of the test generator depends on the type and location of the Exception. Either a Exception is considered as a bug in a unit under test, or as a wrong call produced by the test generator. If the Exception is considered to be a bug, then the statement that caused the Exception is added to the test and the test generation finishes. If the Exception is considered to be caused by a wrong call, then the statement is not added to the test and the test generation continues.

The different types of Exceptions are categorized as follows:

Error Exceptions that are a subclass of *Error* are considered as a wrong call. (Details on the different types of Exceptions can be found in chapter 11 of the Java Language Specification.)

Checked Exceptions Checked Exceptions are considered as a wrong call.

Unchecked Exceptions 1 If the Exception is of type *ArrayIndexOutOfBoundsException*, *NegativeArraySizeException*, *ArrayStoreException*, *ClassCastException*, or *ArithmeticException* and is thrown in a method of the class under test, this is considered to be a bug.

Unchecked Exceptions 2 If the Exception is of type *IllegalArgumentException*, *IllegalStateException*, *NullPointerException*, and is thrown by a public method that is (transitively) called from a method of the class under test, this is considered to be a bug.
Note: A stack trace of an Exception gives only the name of the method, and not its signature. In the presence of multiple methods with the same name, it thus cannot be decided which method produced the Exception without accessing the source code. Therefore, we consider a method in the stack trace to be public, if there exists at least one public method with the same name in the class.

This heuristic is taken from the paper *JCrasher: an automatic robustness tester for Java* For more details consider the paper which is available from our website (user name and password is *tad10*).

3.7 Resulting JUnit Test

The test case produced by *TestGenerator* should be a JUnit 4 test class and should contain exactly one test method. Exceptions that might be thrown by the test should not be wrapped in try-catch blocks. Instead, the test method should declare that it throws an Exception. The location of the test is specified by the system properties *test.name* and *test.dir*, the directory inside the *test.dir* should reflect the package name as specified by *test.package*.

4 Delta Debugging

The *DeltaDebugger* class receives a JUnit 4 test that it should minimize using the *ddmin* algorithm. The minimized test case should fail with the same Exception. This means that the Exception of the same type is thrown by the same statement in the minimized test and the original one.

The test is minimized by applying *ddmin* to the source code, where each line can be considered as a circumstance. During the run of the algorithm, different versions of the test have to be compiled and run. Therefore, the following system properties get passed:

dd.file The location of the file that contains the test to minimize.

dd.class The fully qualified class name of the test to minimize.

temp.dir A directory that can be used for producing intermediate versions during delta debugging.

min.dir Output directory where the minimized test is written. Again, this is only the base directory. Depending on the package name of the test, the actual directory of the test has to be a sub-directory.

5 Hints

- The public tests for this project come as JUnit 4 tests. These tests are in the *src/test* directory.
- The class *org.apache.commons.lang.RandomStringUtils* can be used to produce random Strings.
- The method *escapeJava()* in *org.apache.commons.lang.StringEscapeUtils* escapes the characters in a String using Java String rules.
- In order to programmatically compile Java files, you can use the Compiler API (see *javax.tools.JavaCompiler*).
- To load a class that is not on the classpath, you can create your own class loader (e.g. *java.net.URLClassLoader*) for loading the class.

Enjoy!