Jacob Geddings

Program Description

Program Details

Program is designed to function as a simple switch-case calculator. User specifies the type of operation they want to perform whether it be addition, subtraction, multiplication, or division. From there the user inputs two variables they want modified and they are returned the result of this procedure.

Source Line of Code

The code itself consists of four separate function calls and two classes. There is also a total of 97 lines of code with four private functions, six private variables, and one public function.

Description of Bugs

Bugs introduced are as follows: break statement following case 1 has been disabled on line 28 allowing a run-on into case 2, case 4 carries a divide by zero error on line 41, case 0 is normally an exit status but the Boolean has been flipped on line 46, additional divide by zero added in on line 51 under default case, and lastly for each function the second operand has been replaced with the maximum negative float value (lines 61, 69, 77, 85).

Utility of Tool

How the Tool Works

Evosuite attempts to interpret the given code and generate several random test cases to stress test the program. Each pass at test generation is random, however; the core test conditions are the same, but the variables it passes will frequently change. For each class, it generates two files: the actual test file and a ‘scaffold’ file. The test file itself is to make it compatible with Maven testing whereas the scaffold shows the programmer what is being done for the test – and allow edits if necessary.

Pros & Cons

This tool correctly interpreted variables and assigned values to try against them, though after seven random test generations it failed to correctly identify problems within the code. Each of the previously mentioned bugs went unchecked by Evosuite, meaning it failed to detect intended outcome, failed to recognize a breach in switch-case formatting, failed to identify a potential overflow of a variable, and failed to notice when a divide by zero was going to occur.

Improvements

Several things come to mind when thinking of ways to improve this tool. The most fundamental of improvements is the need to detect a break in syntax – switch-cases are not meant to lack a ‘break;’ at

the end, and divide by zero should under no circumstances be allowed to pass with the statement “0 failures.” On a more advanced level, Evosuite could use the ability to recognize attempts at overflowing a variable; each of the functions had an operand set to the maximum limits of float and did nothing about it. It also could use functionality to interpret if a variable is going unused. The second operand is effectively useless after the introduction of my bugs – despite this, it carries on with its testing.

Generated Test Cases

Number of Generated Test Cases

Two test files were generated to handle the two given classes; of these files, a total of eight test cases where generated. All testing cases failed to correctly identify a bug within the code.

Detected Bugs

No test case detected a fault.

Usability Discussion

Since the code failed to detect all problems with my code, many of the previously noted “additions” for the tool apply here. It should have tested for formatting problems such as the lack of a break statement. It also could attempt to recognize when the program is not printing out after its inputs – this was probably the result of its random entries or bugs I’ve placed; it’s difficult to determine since Evosuite didn’t even notice. I think the debugger would be improved greatly if it tried to assert an output, since the code clearly failed on several occasions based on its bizarre inputs.