Lab 2: Debugging

Team Members:

1. Dhrumil Amish Shah

Date:

2021-01-22

—

Subject:

Software Development Concepts

—

Professor:

Matthew Amy

HfxDonairExpress.java

Defects and Causes:
Defect: toppings ArrayList not initialized at line number 30.
Cause: NullPointerException when tried to add toppings in it.

Defect: No break statement after line number 49. (Added at line 50)
Cause: Execution of subsequent cases.

Defect: Use of '==' to compare two strings at line number 61, 64 and 67.
Cause: Even for the correct topping value, it resulted in false.

Defect: No condition to verify toppings not repeated (Added at line 59)
Cause: The same topping added multiple times.

Defect: No condition to track selected pizza toppings (Added at line 63, 66 and 69)
Cause: The same topping added multiple times.

Defect: Toppings prize assigned directly to price at line 62, 65 and 68.
Cause: False pizza price calculation.

Defect: Wrong discount calculation at line 86.
Cause: False price calculation at the end.

Approach:
1.) Looked at the code and tried to find mistakes and solve them.
2.) Prepared different test cases to run for various input dataset.
3.) Ran program multiple times with different test cases and check where it fails to give the correct output. Noted down the false test cases.
4.) Ran one false test case at a time in debug mode to find the defect and solve it. Features used like step in, out, over, call stack, variables window.
5.) Ran test cases again to verify whether all defects were resolved.

Approximate amount of time required to locate the defects:
Less than 15 minutes

Ackermann.java

Defects and Causes:
Defect: Wrong method call at line number 20 due to method call ackermann(2, 13) at line number 29.
Cause: StackOverflowError (Infinite recursion) and the program crashed.

Approach:
1.) Ran code and found it crashed. (Test case 3 at line number 29 failed).
2.) Commented test case 1 and 2. Ran the test case 3 in debug mode and found that function call ackermann(2, 13) always call the same function repeatedly at line number 20.

3.) Changed function call ackermann(m, n) to ackermann(m, n - 1) at line number 20.
4.) Uncommented test cases 1 and 2. Ran program and it passed all the test cases.

Approximate amount of time required to locate the defects:
Less than 3 minutes

LinkedListTest.java

Defects and Causes:
Defect: append(list) function does not append one list to another.
Defect: copy(list) creates a shallow copy instead of a deep copy.
Cause: Use of shallow copy instead of deep copy. (As both list root elements point to the same node - Hence, only one copy of list exists in the memory.)

Approach/Solution implemented:
1.) Added a new method called addLast(element). This method adds the element at the end of the list.
2.) Commented the code in copy() function that creates the shallow copy and instead wrote a code that creates a deep copy (New linked list in memory).
3.) Append function will now append a deep copy of the list to the end of the second list.

Approximate amount of time required to locate the defects:
Almost 5 minutes

Analysis

Q1.) What strategy or strategies for debugging are most effective for you?
A general strategy I follow to debug code.
Step 1: Go through code and check for errors. If found, resolve those errors.
Step 2: Prepare test cases to cover different scenarios. Run them normally (without debugger) and note down the output.
Step 3: For test cases that did not pass, run them with a debugger attached and step into each line of code to see the result.
Step 4: Write some more code to make failed test cases work. Run code normally with all test cases and note down the output.
Step 5: If all cases passed, code worked otherwise go to Step 3 and repeat.

Other strategies I have used in the past for debugging.
Unit Testing, Integration Testing, Stress Testing, Backtracking, Problem Simplification, Lint tools, Read the Logcat and stack trace to find the error.

Q2.) What makes them effective?
1.) Easy to find bugs as test cases cover almost all the scenarios.
2.) Debugging makes sure that once code tested, it is almost impossible to find bugs again in the same code unless modified.

3.) Use of different testing strategies during debugging can ultimately lead us to the root cause in code. It can then be isolated, focused and solved.
4.) Lint tools help us resolve bugs and errors. It automates checking of code for programmatic and stylistic problems.

Q3.) For which conditions of the code will your strategies be effective or ineffective?

Strategies will be effective for conditions like uninitialized variables, missing statements, false strings comparisons, incorrect variables manipulation, invalid logic.

Q4.) How can a debugger support your strategies?

1.) It helps to step in, out, over a different part of code and provide results at each step to form a hypothesis.
2.) It helps to narrow down a suspicious region in code and decide.
3.) It provides a comprehensive list of tools like a section to view variables, method call stack, memory utilization. Ultimately it helps in solving bugs quickly.