CPSC 221
Debugging a Set With VS Code

"As soon as we started programming, we found to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs."
--Maurice Wilkes, 1949

"To err is human. To really foul things up requires a computer."
--Anonymous

Objectives

The purpose of this assignment is to become familiar with the debugging facilities in VS Code - to observe the internal workings of a program *while it is running*.

This exercise reflects the same kind of testing situation you will be facing as we continue through the rest of the semester.

Files

The `SetTester` test class contains 111 tests to confirm that `ArraySet` is a valid implementation of the `SimpleSet` ADT. Unfortunately, `ArraySet` contains some bugs. In this assignment, you will use the VS Code debugger to find and fix all bugs in `ArraySet`, so that all 111 `SetTester` tests pass. (See sample final console output, below.)

Copy all of the following files into an VS Code folder.

- ArraySet.java - an array-based implementation of `SimpleSet` - this is the file you are trying to fix
- SimpleSet.java - an interface defining a simple Set ADT - DO NOT MODIFY
- SetTester.java - a test class for `SimpleSet` implementations - DO NOT MODIFY
- README - an example README file from the student who coded this; if this had been a real project

Tasks

Before anything else, read through the javadocs of the `SimpleSet` interface to understand what each method is expected to do. Look through `ArraySet` and understand the relationship between the array and `rear` variable. Also skim through `SetTester` to become generally familiar with the approach to testing being used.

Document your debugging process, as you work, in a plain text file named `Debug.txt`.

For each bug you are tracking down:

- name the failed test where you are beginning your hunt
- where you placed breakpoints
- where the bug was found
- what you did to fix it.

It would also be helpful to describe how you recognized the bug. For example, you might list the contents of the list before and after a particular method call, and explain how that result revealed the problem.

You may place debugger breakpoints wherever you need to in `SetTester`, but NO MODIFICATIONS may be made to the `SetTester` code - do not add so much as a print statement. Use the debugger to determine the state of the program as it is running. Likewise, NO MODIFICATIONS are allowed in `SimpleSet`. The `ArraySet.java` is the **only** file you may edit.

Begin debugging:

- Run `SetTester` to get current test results. Tests are ordered from more basic to more advanced. If basic tests are failing, more advanced tests that build on basic scenarios are meaningless.
- Start with the first test that is failing. Find that test in `SetTester` and place a breakpoint at the line where it is called.
- Run `SetTester` in debug mode. When it pauses at your breakpoint, use a combination of step-over and step-into to step through the code and observe the state of the `ArraySet` as it is tested. **Be aware**, the test that revealed the bug may not be testing the method where the bug is actually occurring. The problem may actually be in a method call that *preceded* the failed call.

- When you have located a bug, stop the debugger and attempt to fix it.
- Rerun `SetTester` (go back to the top of these steps). Continue this process until all 111 tests in `SetTester` pass.

Note: To ease the challenge, somewhat, you can be confident that the `toString()` method is bug-free, so the `Strings` showing Set contents while debugging are reliable.

Debugging tips & tricks

- When looking at a FAILed test, think about the whole sequence of steps that led up to the failed call. The last call is often not where the problem actually occurred. For example, if there is an error in the `add()` or `remove()` method, then it is reasonable to assume that `size()` might FAIL, even if the `size()` method is correctly written.
- After placing a breakpoint, plan to make multiple run throughs with the debugger. Your first run through may be fairly quick - simply trying to get a feel for the sequence of steps leading up to the FAIL result. Subsequent run throughs may be slower - observing changes in the data through the sequence of calls, and trying to identify where something unexpected occurs. As you rule out early steps, your run throughs may quickly advance to steps where you suspect the bug is occurring and then slow down greatly.
- You will find your own rhythm of how to step-into and step-over. When in doubt, step-into. This will give you the opportunity to read more code and hunt down what is happening. You're learning a new tool, so take your time to try your best to understand what is going on.
- Once you have corrected all of the bugs and passed all 111 tests, try breaking code with off by one errors and other accidental bugs you think would be easy to make in the `ArraySet` code. See what happens to the test results as a result of each bug you introduced. (Just be sure to fix them, again, before turning in the assignment!)
- In addition to helping you fix buggy code, the debugger is also a useful tool for getting familiar with working, bug-free code!

Grading

Points will be awarded according to the following breakdown:

Tasks	Points
All bugs found and fixed correctly	10
Debug process well documented in *Debug.txt*	10

Submission

Submit this assignment (a plain-text file named `Debug.txt` and the `ArraySet.java` source files) as a .zip file with the following format (Note: there is **no** `README` for this assignment; the `Debug.txt` is the place to describe your progress and how the lab went):

Lab06_*LastName_FirstName*.zip

Sample Console Output

Sample Output (when all 111 tests PASS).

```
NEW EMPTY SET
toString() output: []
emptySet_testToString                                   PASS
emptySet_testIsEmpty                                    PASS
emptySet_testSize                                       PASS
emptySet_testContainsA                                  PASS
emptySet_testAddA                                       PASS
emptySet_testRemoveA                                    PASS
=========================================================
SCENARIO: [ ] -> add(A) -> [A]
toString() output: [1]
emptySet_addA_A_testToString                            PASS
emptySet_addA_A_testIsEmpty                             PASS
emptySet_addA_A_testSize                                PASS
emptySet_addA_A_testContainsA                           PASS
emptySet_addA_A_testContainsB                           PASS
emptySet_addA_A_testAddA                                PASS
emptySet_addA_A_testRemoveA                             PASS
emptySet_addA_A_testRemoveB                             PASS
=========================================================
SCENARIO: [A] -> remove(A) -> [ ]
toString() output: []
A_removeA_emptySet_testToString                         PASS
A_removeA_emptySet_testIsEmpty                          PASS
A_removeA_emptySet_testSize                             PASS
A_removeA_emptySet_testContainsA                        PASS
A_removeA_emptySet_testAddA                             PASS
```

```
A_removeA_emptySet_testRemoveA                              PASS
=========================================================
SCENARIO: [A] -> add(A) -> [A]
toString() output: [1]
A_addA_A_testToString                                       PASS
A_addA_A_testIsEmpty                                        PASS
A_addA_A_testSize                                           PASS
A_addA_A_testContainsA                                      PASS
A_addA_A_testContainsB                                      PASS
A_addA_A_testAddA                                           PASS
A_addA_A_testAddB                                           PASS
A_addA_A_testRemoveA                                        PASS
A_addA_A_testRemoveB                                        PASS
=========================================================
SCENARIO: [A] -> add(B) -> [A,B]
toString() output: [1, 2]
A_addB_AB_testToString                                      PASS
A_addB_AB_testIsEmpty                                       PASS
A_addB_AB_testSize                                          PASS
A_addB_AB_testContainsA                                     PASS
A_addB_AB_testContainsB                                     PASS
A_addB_AB_testContainsC                                     PASS
A_addB_AB_testAddA                                          PASS
A_addB_AB_testAddB                                          PASS
A_addB_AB_testAddC                                          PASS
A_addB_AB_testRemoveA                                       PASS
A_addB_AB_testRemoveB                                       PASS
A_addB_AB_testRemoveC                                       PASS
=========================================================
SCENARIO: [A,B] -> remove(A) -> [B]
toString() output: [2]
AB_removeA_B_testToString                                   PASS
AB_removeA_B_testIsEmpty                                    PASS
AB_removeA_B_testSize                                       PASS
AB_removeA_B_testContainsA                                  PASS
AB_removeA_B_testContainsB                                  PASS
AB_removeA_B_testAddA                                       PASS
AB_removeA_B_testAddB                                       PASS
AB_removeA_B_testRemoveA                                    PASS
AB_removeA_B_testRemoveB                                    PASS
=========================================================
SCENARIO: [A,B] -> remove(B) -> [A]
toString() output: [1]
AB_removeB_A_testToString                                   PASS
AB_removeB_A_testSize                                       PASS
AB_removeB_A_testIsEmpty                                    PASS
AB_removeB_A_testContainsA                                  PASS
AB_removeB_A_testContainsB                                  PASS
AB_removeB_A_testAddA                                       PASS
AB_removeB_A_testAddB                                       PASS
AB_removeB_A_testRemoveA                                    PASS
AB_removeB_A_testRemoveB                                    PASS
=========================================================
SCENARIO: [A,B] -> add(C) -> [A,B,C]
toString() output: [1, 2, 3]
AB_addC_ABC_testToString                                    PASS
AB_addC_ABC_testIsEmpty                                     PASS
```

```
AB_addC_ABC_testSize                                    PASS
AB_addC_ABC_testContainsA                               PASS
AB_addC_ABC_testContainsB                               PASS
AB_addC_ABC_testContainsC                               PASS
AB_addC_ABC_testContainsD                               PASS
AB_addC_ABC_testAddA                                    PASS
AB_addC_ABC_testAddB                                    PASS
AB_addC_ABC_testAddC                                    PASS
AB_addC_ABC_testAddD                                    PASS
AB_addC_ABC_testRemoveA                                 PASS
AB_addC_ABC_testRemoveB                                 PASS
AB_addC_ABC_testRemoveC                                 PASS
AB_addC_ABC_testRemoveD                                 PASS
=========================================================
SCENARIO: [A,B,C] -> remove(A) -> [B,C]
toString() output: [2, 3]
ABC_removeA_BC_testToString                             PASS
ABC_removeA_BC_testIsEmpty                              PASS
ABC_removeA_BC_testSize                                 PASS
ABC_removeA_BC_testContainsA                            PASS
ABC_removeA_BC_testContainsB                            PASS
ABC_removeA_BC_testContainsC                            PASS
ABC_removeA_BC_testAddA                                 PASS
ABC_removeA_BC_testAddB                                 PASS
ABC_removeA_BC_testAddC                                 PASS
ABC_removeA_BC_testRemoveA                              PASS
ABC_removeA_BC_testRemoveB                              PASS
ABC_removeA_BC_testRemoveC                              PASS
=========================================================
SCENARIO: [A,B,C] -> remove(B) -> [A,C]
toString() output: [1, 3]
ABC_removeB_AC_testToString                             PASS
ABC_removeB_AC_testIsEmpty                              PASS
ABC_removeB_AC_testSize                                 PASS
ABC_removeB_AC_testContainsA                            PASS
ABC_removeB_AC_testContainsB                            PASS
ABC_removeB_AC_testContainsC                            PASS
ABC_removeB_AC_testAddA                                 PASS
ABC_removeB_AC_testAddB                                 PASS
ABC_removeB_AC_testAddC                                 PASS
ABC_removeB_AC_testRemoveA                              PASS
ABC_removeB_AC_testRemoveB                              PASS
ABC_removeB_AC_testRemoveC                              PASS
=========================================================
SCENARIO: [A,B,C] -> remove(C) -> [A,B]
toString() output: [1, 2]
ABC_removeC_AB_testToString                             PASS
ABC_removeC_AB_testIsEmpty                              PASS
ABC_removeC_AB_testSize                                 PASS
ABC_removeC_AB_testContainsA                            PASS
ABC_removeC_AB_testContainsB                            PASS
ABC_removeC_AB_testContainsC                            PASS
ABC_removeC_AB_testAddA                                 PASS
ABC_removeC_AB_testAddB                                 PASS
ABC_removeC_AB_testAddC                                 PASS
ABC_removeC_AB_testRemoveA                              PASS
ABC_removeC_AB_testRemoveB                              PASS
```

```
ABC_removeC_AB_testRemoveC                                  PASS
=====================================================
SCENARIO: [] -> add 1000 elements
empty_addManyElements_bigSet                          PASS

Total Tests: 111,  Passed: 111,  Failed: 0
```