CSCI 2134 Assignment 3

Due date: 11:59pm, Sunday, November 12, 2023, submitted via Git

Objectives

Practice debugging code, using a symbolic debugger, and fixing errors.

Preparation:

Clone the Assignment 3 repository

https://git.cs.dal.ca/courses/2023-fall/csci-2134/assignment3/????.git

where ???? is your CSID.

Problem Statement

Take a piece of buggy code, debug it, and fix it.

Background

You have inherited some buggy code for doing Markov chain simulations. A specification for the entire project is provided in `docs` of the repo. Since last time, new classes include RandomWalker, WalkCavanas, WalkFrame, and WalkSim.

Your boss fired their previous developer because the former developer did not do any testing and did not fix the bugs! Your boss has asked you to fix the code. She will provide you with some unit tests (some of which fail), and sample input and output of what should be produced.

Your job is to fix the bugs: Both the bugs exhibited by the unit tests and the ones by the input test cases.

Note:

Although these classes and code are very similar to Assignment 2, this is a standalone assignment. You do not need to "import" your unit tests from Assignment 2. Everything you need is included directly in the Assignment 3 repo. Simply use the unit tests you have been provided.

Tasks

1. Review the specification (**`specification.pdf`**) in the **`docs`** directory. You will absolutely need to understand it and the code you are debugging. Spend some trying to understand the code. Use a top-down or control flow approach. This will help you a lot later on!

2. **Fix all bugs** that are identified by the tests generated by the unit tests in the following classes:
   - `Coordinate.java`
   - `FloatMatrix.java`
   - `MarkovChain.java`
   - `RandomWalker.java`

3. See **`errors.txt`** file in the **`docs`** directory. One sample entry is provided. For each fix you make in these classes, add an entry to this file that includes:
   a. The file/class name where the bug was.
   b. The method where the bug was.
   c. The test method that exposed the bug, if applicable.
   d. The line number(s) where the buggy code was.
   e. A description of what the bug was.
   f. A description of what the fix was.

   Notice that the sample does not include d, e, or f, since it was not a real bug that was fixed.

4. The previous developer made a set of example input (`exampleX.txt`) and expected output (`goldX.txt`) in the **`test_cases`** directory. `WalkSim.java` can read in different example files using command line arguments or by modifying the exampleFile variable. These tests will likely not pass yet even after fixing the bugs identified by the unit tests.
   - Compare the output files with the expected goldX.txt files.
   - **TIP:** You can easily compare two files using git!
     - `git diff --no-index <fileA> <fileB>`
   - For each output that differs from the expected output, debug the code and determine the reason for the mismatch. Fix any identified bugs missed by the unit tests.

5. Record any new bugs found and fixed from Step 4 in the previously created **`errors.txt`**

6. **Commit and push** the bug fixes and the **`errors.txt`** file to the remote repository.

Submission

All fixes and files must be committed and pushed back to the remote Git repository.

Grading

The following grading scheme will be used:

Task	4/4	3/4	2/4	1/4	0/4
Bugs found [unit tests] (20%)	At least 6 bugs are correctly identified and documented.	4 to 5 bugs are correctly identified and documented.	3 bugs are correctly identified and documented.	1 or 2 bugs are correctly identified and documented.	Zero (0) bugs are correctly identified and documented.
Bugs fixed [unit tests] (20%)	At least 6 bugs are correctly fixed. All unit tests pass.	4 to 5 bugs are correctly fixed.	3 bugs are correctly fixed.	1 or 2 bug are correctly fixed.	Zero (0) bugs are correctly fixed.
Bugs found [input tests] (20%)	At least 3 bugs are correctly identified and documented.	Two (2) bugs are correctly identified and documented.	One (1) bug is correctly identified and documented.	N/A	Zero (0) bugs are correctly identified and documented.
Bugs fixed [input tests] (30%)	At least 3 bugs are correctly fixed. All input tests pass.	Two (2) bugs are correctly fixed.	One (1) bug is correctly fixed. Some input tests pass	N/A	Zero (0) bugs are correctly fixed.
Document [`buglist.txt`] Clarity (10%)	Document looks professional, includes all information, and easy to read	Document looks ok. May be hard to read or missing some information.	Document is sloppy, inconsistent, and has missing information	Document is very sloppy with significant missing information	Document is illegible or not provided.

Hints

1. You will probably need to use a symbolic debugger to make headway. Using print-statements will be possible but extremely painful.
2. You may need to step through the code to find the bugs.
3. There are about 3-5 bugs in the code (in addition to the ones identified by the unit tests). The single bug report should cover all of them.
4. **Assume there are no bugs in WalkCanvas.java or WalkFrame.java.**