CSCI 2134 Assignment 4

Due date: 11:59pm, Sunday, December 3, 2023, submitted via Git

Objectives

Extend an existing code-base and perform some basic class-level refactoring in the process.

Preparation:

Clone the Assignment 4 repository

https://git.cs.dal.ca/courses/2023-fall/csci-2134/assignment4/????.git

where ???? is your CSID.

Problem Statement

Take an existing codebase, add required features, test it, and refactor it as necessary.

Background

The WalkSim application is ready to move on to version 2! Now that all the bugs are fixed, it's time to add some new features, as requested by the client. Your boss has hired you to extend the code. She also mentioned that the original designer of the code did not do a great job and wondered if there was any way to improve the code. She will provide you with (i) the "bug free" codebase, (ii) the existing code specification, and (iii) the requirements of the additions to be made.

Your job is to (i) create a design for the additions, (ii) implement the additions, (iii) create unit tests for the additions, and (iv) identify opportunities for class-implementation and class-interface refactoring, and (v) do some refactoring where appropriate.

Note1:

Although these classes and code are very similar to Assignment 3, this is a standalone assignment. You do not need to "import" your unit tests or bug fixes from Assignment 3. Everything you need is included directly in the Assignment 4 repo. Simply use the unit tests and existing code provided.

Note2:

It's a good learning exercise to compare your solution to Assignment 3 to the code provided in Assignment 4. Look to see if you missed any bugs. Compare the way you fixed the bugs versus the way the bugs are fixed in the Assignment 4 code. Good questions for self reflection are: Are the two fixes for the same bug different? Which fix do you like better? Why?

Tasks

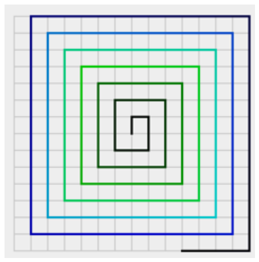
1. Review the old specification (**`specification.pdf`**) in the **`docs`** directory of the repo. You will absolutely need to understand it and the code you are extending.
2. Review the new requirements at the end of this document, which describes all the extensions to be done.
3. Design and implement the extensions using the best-practices we discussed in class.
    a. Hint: consider the refactorings of step 6 and 7, below, while doing this design.
4. Provide a readable, professional looking UML diagram of the updated design. This includes both pre-existing classes and any classes you add to implement the extensions.
    a. There are a lot of online tools available to draw UML diagrams. Creately, draw.io, canva, and lucidchart are all great options.
    b. Commit your UML diagram as **`design.pdf`** in the **`docs`** directory of the repository.
5. Provide units tests, in the form of Junit5 tests, for:
    a. The two new types of walk. For each walk type, create three unit tests similar to the three walk tests of RandomWalkTest.
    b. The new way of saving walk paths. There should be at least two unit tests: one for a valid path and output file, and one unit test which triggers the IOException.
6. In a file in the **`docs`** directory called **`refactoring.txt`**, list all the class-implementation and class-interface refactoring that you will do. In the file, explain the code smell that motivated each refactoring.
    a. Note: it is not expected for you to refactor WalkFrame.java or WalkCanvas.java.
7. Actually implement the class-implementation and class-interface refactorings that you described to do in the previous step.
8. **Commit and push back** everything to the remote repository.
    a. **Note:** you really should be committing frequently throughout this assignment. Recall best practices for debugging, refactoring, and software development generally: make many small commits!

Submission

All extensions and files should be committed and pushed back to the remote Git repository.

Hints

1. A waterfall approach is probably easiest for this simple application. That is, do the design first and look at refactoring as you design. Implement *after* the design is complete.
2. The SpiralWalker with 200 steps should look like this:



Grading

The following grading scheme will be used:

Task	4/4	3/4	2/4	1/4	0/4
Design (10%)	Design is cohesive, meets all requirements, and follows SOLID principles	Design meets all requirements and mostly follows SOLID principles	Design meets most of the requirements.	Design meets few of the of requirements.	No design submitted.
UML (10%)	UML class diagram accurately reflects the implemented design. All necessary relationships and class attributes are correctly included.	UML class diagram accurately reflects the implemented design. Most relationships and class attributes are correctly included.	UML class diagram closely, but not exactly, reflects the implemented design. Most relationships and class attributes are correctly included.	UML class diagram generally reflects the implemented design. Some relationships and class attributes are correctly included.	No UML diagram provided.
Implementation (40%)	All requirements are implemented correctly.	3 of the requirements are implemented correctly	2 of the requirements are implemented correctly	1 of the requirements are implemented correctly	No implementation
Testing (10%)	All 8 unit tests expected are implemented correctly and pass.	At least 6 unit tests are implemented correctly and pass.	At least 5 unit tests are implemented correctly and pass.	2 or 3 unit tests are implemented correctly.	No testing
Refactoring Description (10%)	At least 4 class level refactoring suggestions that follow SOLID principles and make sense.	At least 3 class level refactoring suggestions that follow SOLID principles and make sense.	At least 2 class level refactoring suggestions that follow SOLID principles and make sense.	At least 1 class level refactoring suggestions that follow SOLID principles and make sense.	No refactoring suggestions.
Refactoring Implementation (10%)	At least 4 class-level refactoring suggestions are implemented correctly.	At least 3 class-level refactoring suggestions are implemented, with 1 being done correctly.	At least 2 class-level refactoring suggestion is implemented correctly.	At least 1 class-level refactoring suggestion is implemented.	No refactoring suggestions implemented.
Code Clarity (5%)	Code looks professional and follows style guidelines	Code looks good and mostly follows style guidelines	Code occasionally follows style guidelines	Code does not follow style guidelines	Code is illegible or not provided
Document Clarity (5%)	Documents look professional, includes all information, and easy to read	Documents look good. May be missing some information or less than excellent presentation.	Documents are ok. Some missing information or inconsistent presentation.	Documents are sloppy with significant missing information.	Documents are very sloppy or illegible or not provided.

Specification of Required Extensions

Background

Our customer has requested that WalkSim handles various types of walks and various types of output formats. There may be many such walks and formats in the future. But, for now, there are two new walk types to implement and one new output format to implement.

Extension 1: Changes to Program Input

The customer wants to use `System.in` rather than command line arguments to provide the program with input. The program should perfect the following:

1. The program first prompts the user to enter a positive number of steps to walk, as a single integer. If an integer is not supplied, the program will continue to prompt the user to enter an integer until one is successfully obtained.
2. The program then prompts the user to enter the walker type. The user enters 0 for "Random walk", 1 for "Spiral walk", and 2 for "Bread crumb walk". If 0, 1, or 2 is not obtained, the program continues to prompt the user to enter an integer until a correct one is obtained. If the user chooses "Spiral walk", skip to step 4.
3. If the user chooses Random walk or Bread crumb walk, the program then prompts the user to enter a file name for the `FloatMatrix` to create. If the file cannot be read, or a FloatMatrix cannot be successfully created from the supplied file, the program continues to prompt the user for file names until a valid FloatMatrix is created from the input file.
4. The program then prompts the user to enter the file name to store the output of the program. Note that no validation is needed at this point. If the output file is invalid, the program should behave as it already does.

Hint: Look at the mailbox example in the defensive programming lecture.

Extension 2: Spiral Walker

The customer wants a new walker that walks in a clockwise spiral. The walk should occur as a clockwise spiral of increasing radius. Clockwise means that it first walks north, then east, then south, then west, before restarting the cycle.

- When the Spiral Walker is chosen (see Extension 1 above), the spiral path should be visible and animated in the WalkFrame just as a Random Walker. The spiral path should also be printed to the output file as was the Random Walker.
- Note the spiral walker has no randomness! The path should be: (0, 0), (0, 1), (1, 1), (1, 0), (1, -1), (0, -1), (-1, -1), (-1, 0), …

Extension 3: Bread Crumb Walker

The customer wants a new walker that performs a random walk, but then walks along the same path it took in reverse order to get back to the starting point.

- If the Bread Crumb Walker is asked to walk N steps, it actually produces a path of (N+1) + N coordinates. The final end point of the "forward path" should only be included once, the starting point should be included as both the starting point and the end point.
- When the Bread Crumb Walker is chosen (see Extension 1 above), the path should be visible and animated in the WalkFrame just as a Random Walker. The path should also be printed to the output file as was the Random Walker.

Extension 4: A new output format

The custom wants a new output format: an "integer stream" which contains no punctuation or additional characters besides the integers encoding the coordinates of the path. That is, it outputs one line of text which is a space-separated list of integers.

- If the output file received from the user (see Extension 1) ends in ".txt", output the walk to the file as normal: one coordinate per line "(x,y)".
- If the output file received from the user ends in ".dat", output the walk as an integer stream.
- For example, "output.dat" for a Spiral Walker of 8 steps would contain the line: "0 0 0 1 1 1 1 0 1 -1 0 -1 -1 -1 -1 0 -1 1"

Specification: Non-functional Changes

1. The design should follow the SOLID principles.
2. The customer has informed us that different kinds of walks and output formats will be added in the near future, so the design should reflect this.