CSCI 2134 Lab 4: Finding Common Errors

Fall 2023

Objective

In this lab, you will use the IDE and unit tests to find and correct errors in a small program. You will work by yourself or with a partner to fix code that has many simple errors in it.

Preparation

1. Ensure that you have your Integrated Development Editor (IDE) installed.
2. Ensure that you are able to write and run a JUnit test case in the IDE. You will not have the time to both debug your IDE environment and complete the lab task within the lab time. The lab time should be to concentrate on the lab task.
3. Clone the Lab 4 repo: https://git.cs.dal.ca/courses/2023-fall/csci-2134/lab4/????.git where ???? is your CSID.
4. Review the provided code listed in the Resources section below by reading the code of the class you will be fixing (`Matrix.java`) and the unit tests (`MatrixTest.java`). These files are located in the `src` and `test` directories of the cloned project.
5. You may also wish to read the provided primer on matrix operations, found in the `docs` directory. Note: You do not need to understand all of it, but you do need to have seen it before starting the lab.

Resources

- Primer on matrices and FixList.txt in the **`docs`** directory of the Lab 4 repository.
- Code base to be debugged is in the **`src`** directory of the Lab 4 repository.
- Test code is in the **`test`** directory of the Lab 4 repository.

Procedure

Set-up

1. Decide if you wish to work by yourself or pair up with another student.
2. Enter your bannerID, name, and csid in FixList.txt. If you work with another student enter their information as well. **Only one partner should submit the lab by pushing to git.**
3. Open the Lab 4 project cloned from git.

Part 1: Lab Steps

1. Record the current time.
2. Review the code in `Matrix.java`. Review issues flagged by the IDE (highlighted/underlined code) by hovering your mouse pointer over the highlighted code. You can also click the yellow exclamation point triangle at the top-right of the text editor window to see all warnings. For each warning, decide if it is an issue and, if so, fix it.
3. Note the current time and compute how long it took you to perform the fix.

4. Add an item to FixList.txt. All your fixes should be noted here. You need to specify three things:
    - Location: e.g., “Line 3 in method doThis()” or “Instance variable myVariable”
    - What the fix was: e.g., “wrong comparator operator, should be <=, not <”
    - Amount of time it took to complete the fix
5. **Commit the fix with a useful commit message. One student should push the fix back to the repository.** If you are working with a partner, that partner should also make and commit the same fix locally, **but not push**.
6. Repeat until all issues have been reviewed and fixed.

Part 2: Lab Steps

1. Record the current time.
2. Run the unit tests. Identify which tests fail. Tests are named after the methods they are testing.
3. Select the simplest (shortest) broken method (e.g., getWidth())
4. Review the corresponding unit tests and identify why the test is failing. This review means understanding what the test is expecting and what the method being tested is returning.
5. Look at the selected method and use the information you got from the test to figure out where the method is broken. Note: You may need to use a debugger. ☺
6. Once you find the error, fix it.
7. Rerun the unit tests to make sure you have fixed the problem.
8. Note the current time and compute how long it took you to fix the problem.
9. Note the fix in FixList.txt.
10. **Commit the fix with a useful commit message. One student should push the fix back to the repository.** If you are working with a partner, that partner should also make and commit the same fix locally, **but not push**.
11. Repeat until all the unit tests pass.

Analysis

Compute how long it took you to complete all the bug fixes and the average time per bug fix. Identify the hardest bug you fixed and what made it hard to fix. Also identify how consistent your time was to fixing bugs and what factors may have affected that consistency.

Reporting

1. Add your analysis to the end FixList.txt file.
2. List all the members of your team at the start of FixList.txt.
3. **Commit and push the entire project.**

Reminder: your project is not submitted until it is pushed to gitlab!

Grading

The lab will be marked out of 4 points:

Task	2 Points	1 Point	0 Points
Part 1	Most warnings/issues highlighted by the IDE have been fixed.	Some warnings/issues highlighted by the IDE have been fixed.	No highlighted issues have been fixed.
Part 2	At least 3 methods have been fixed: `getWidth()` `getElem()` `setElem()`	At least two methods have been fixed.	No methods have been fixed.

A Primer on Matrices

A matrix of size $m \times n$ is a 2D grid of values, typically decimal numbers, with m rows and n columns. Element $E[i,j]$ of a matrix, refers to the value in row i and column j. Element $E[1,1]$ is in the top left corner of the matrix and element $E[m,n]$ is the bottom right corner of the matrix. A matrix can be negated, added with another matrix, multiplied by a scalar, or multiplied by another matrix.

Two matrices are equal if their elements-wise are the same. A matrix is negated by negating all the elements. E.g.,

$$\begin{bmatrix} -1 & -2 \\ -3 & -4 \end{bmatrix} = -\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Two matrices can be added only if they have the same size. Addition is performed element-wise, meaning that if $C = A + B$, where C, B, and A, are matrices, then element $C[i, j] = A[i, j] + B[i, j]$ for all elements of C. E.g.,

$$\begin{bmatrix} 6 & 8 \\ 10 & 12 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

Any matrix can be multiplied by a scalar, which is typically a decimal value. Scalar multiplication is performed element-wise, meaning that if $C = sA$, where s is a scalar value and A is a matrix, then $C[i,j] = s \times A[i,j]$. E.g.,

$$\begin{bmatrix} -3 & -6 \\ -9 & -12 \end{bmatrix} = -3\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Two matrices can be multiplied only if the width of the first matrix is equal to the height of the second matrix. If $C = AB$, where A is a matrix of size $m \times n$, B is a matrix of size $n \times p$, and C is a matrix of size $m \times p$, then $C[i, j] = A[i, 1] \times B[1, j] + A[i, 2] \times B[2, j] + A[i, 3] \times B[3, j] + \ldots + A[i, n] \times B[n, j]$ E.g.,

$$\begin{bmatrix} 9 & 12 & 15 \\ 19 & 26 & 33 \\ 29 & 40 & 51 \\ 39 & 54 & 69 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$