CS136 Lab Section 1

Logan Behnke and Joshua Pollock

4 April 2017

**Lab 8 – Unit Testing**

**Problem Statement:**

The problem given by this lab is to create a tester class for the Sudoku class. This tester will utilize the JUnit library to test various methods of the Sudoku class and ensure that there are no bugs within our Sudoku class. We will also be required to add on to the Sudoku class so that it will be able to verify that the puzzle does not contain letter characters, i.e. ‘j’.

**Constraints:**

1. Use the JUnit library to verify the Sudoku class
2. Put the sudoku puzzle under tests given by the lab guidelines
3. The Sudoku puzzle should never be allowed to contain an invalid character

**Assumptions:**

1. testisRowValid, testisColValid, and testisSquare valid will not show signs of error unless the square is valid (assertingFalse). This means if there are repeated numbers, the test will not display an error
2. testCharactersLayouts and testSetSquareException should not display errors if the exception is thrown, as the exception is expected. If there are no invalid characters, the test should trigger.

**Features:**

* testCharactersLayouts
* testGetSquare
* testSetSquare
* testSetSquareException
* testisRowValid
* testisColValid
* testisSquareValid
* testisValid
* testisSolved
* testisBoardSolved

**Planning:**

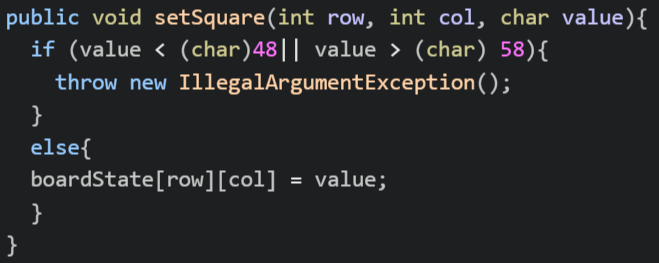
We started off by looking at the given clockpkg files. We also looked at the Vogella JUnit tutorial (http://www.vogella.com/tutorials/JUnit/article.html) as well. After looking at these two things we decided to start setting up the skeleton for our testing file. We referenced the clockpkg tester as an example to create this file but our skeleton was set up using the requirements given by the lab:

* (new) Instantiating Sudoku puzzles with invalid characters or layouts should throw an IllegalArgumentException
* getSquare should return the correct character for a given location
* setSquare should set correctly for a given location
* (new) setSquare should throw an IllegalArgumentException if it is passed an invalid character
* isValid should return false if there are any repeated numbers in a row
* isValid should return false if there are any repeated numbers in a column
* isValid should return false if there are any repeated numbers in any of the nine 3x3 sub squares
* isValid should return true if no earlier rules are violated
* isSolved should return false if there are any blank spaces
* isSolved should return true if no earlier solve or validity rules are violated

After finishing the skeleton file based off these requirements we began to implement the tests for these.

**Implementation:**

­­­­

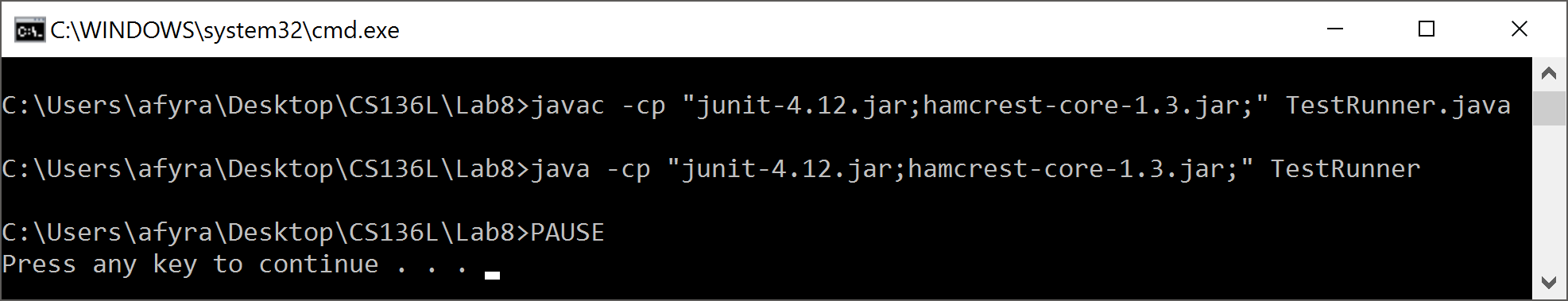
We started off by getting all the simple tests done. The only ones that we had left out were the setsquare and Sudoku layout test (the tests that throw exceptions), as these needed to have changes made to the Sudoku class to function properly. The rest of the tests were quite simple to implement. They just use the tests that have already been implemented in the Sudoku class. For example, the test for isValid just asserted true that the squares, rows, columns, or puzzles were valid using the .isValid() method.   
  
 The changes that needed to be made to the Sudoku class were very minor. The second constructor was changed to check if the inputted starting config had characters in between (char)48 and (char)58. This translates to the integers 1-9 on an ASCII Table (<https://www.cs.cmu.edu/~pattis/15-1XX/common/handouts/ascii.html>). The setSquare() method was also given this checker. The changes were:  


Finishing up the tester, we just needed to add the testers for these new lines of code. We had the test expect an IllegalArgumentException. This completed all the tests we needed to perform on the Sudoku class.

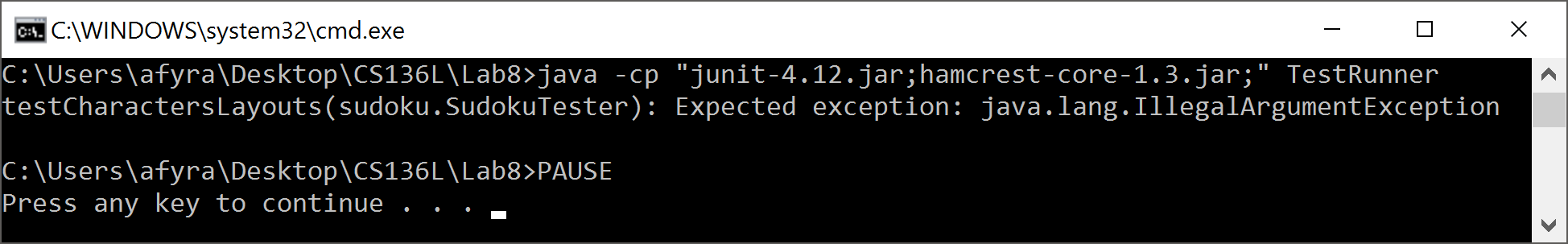
**Running Application:**

testCharactersLayouts-

Pass-

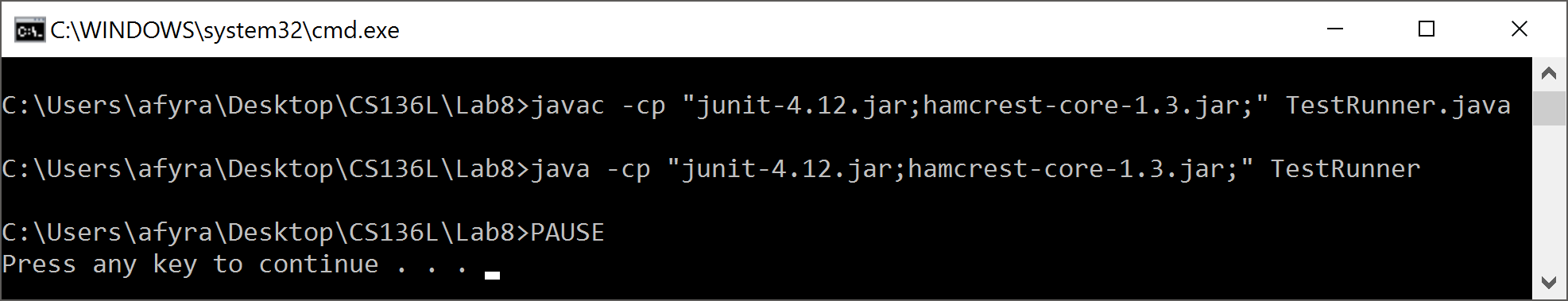


Fail-

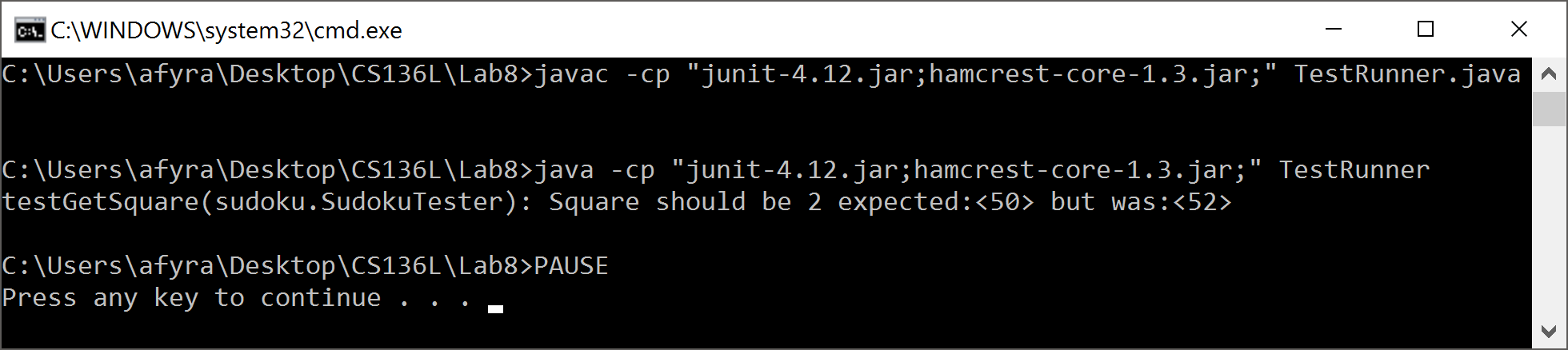


testGetSquare-

Pass-

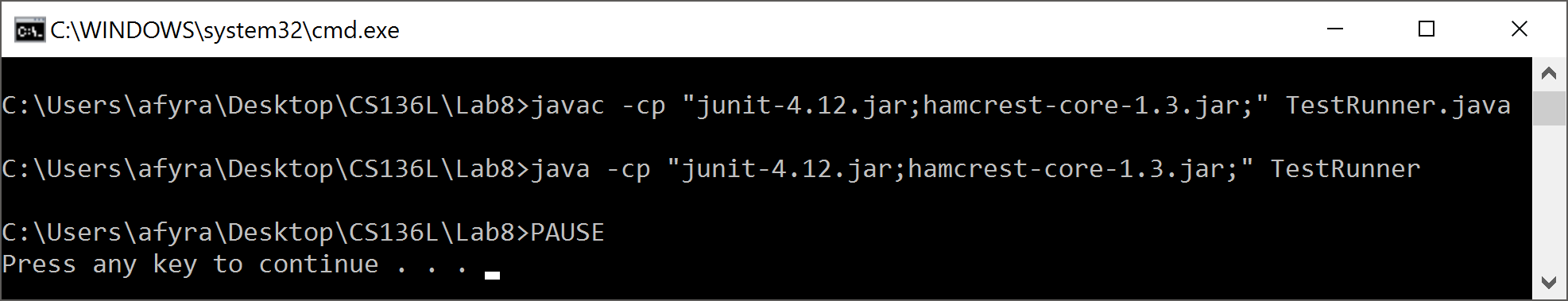


Fail-

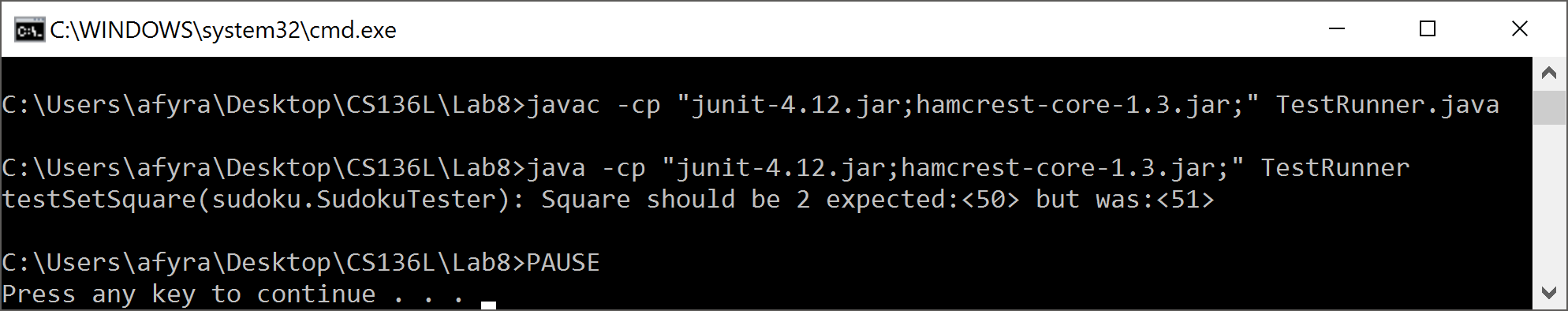


testSetSquare-

Pass-

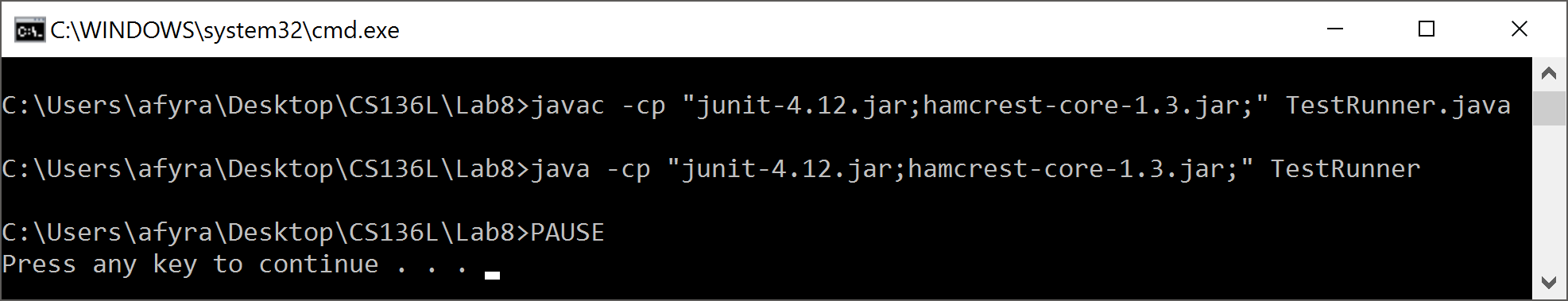


Fail-

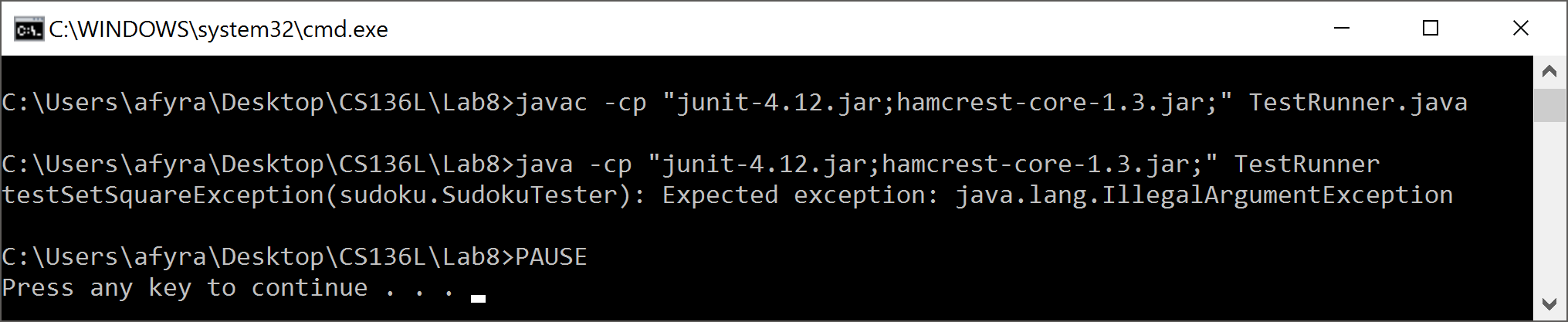


testSetSquareException-

Pass-

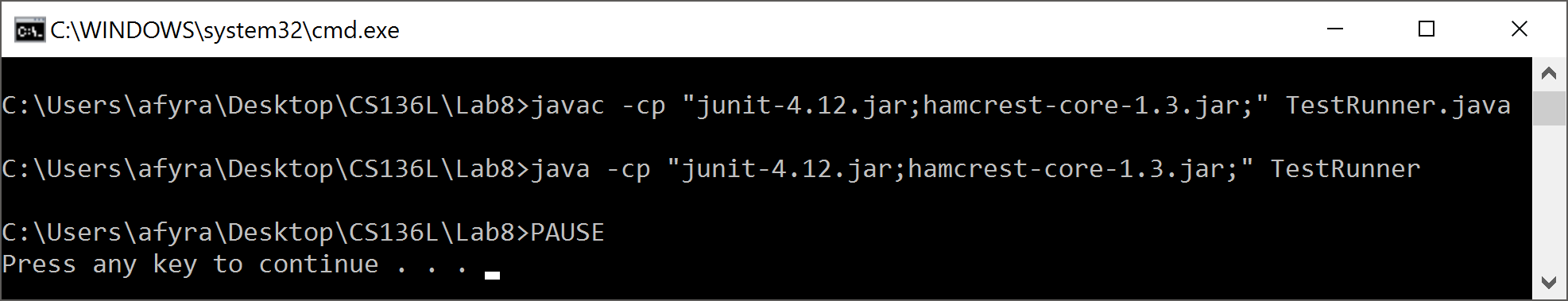


Fail-

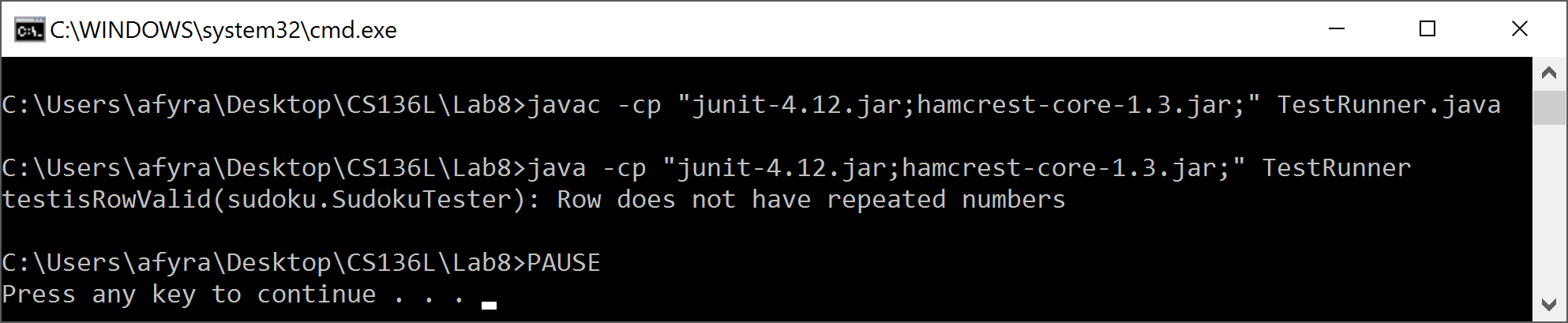


testisRowValid-

Pass-

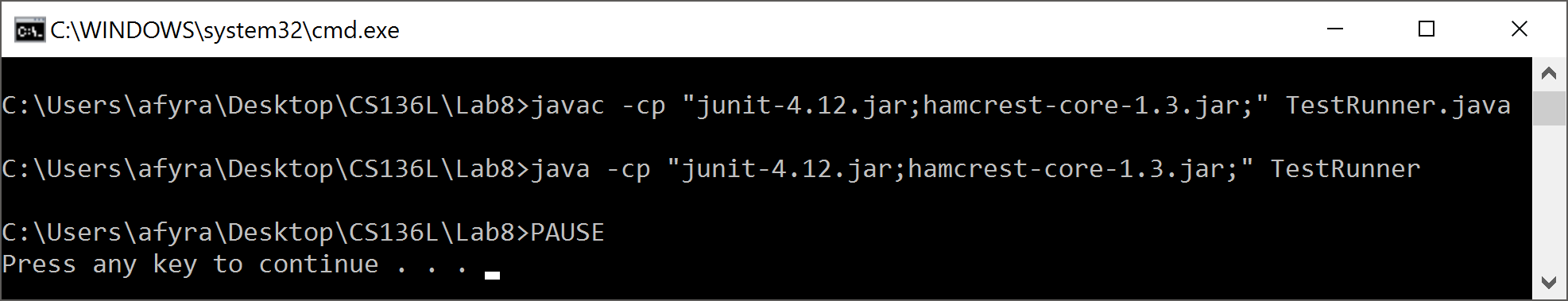


Fail-

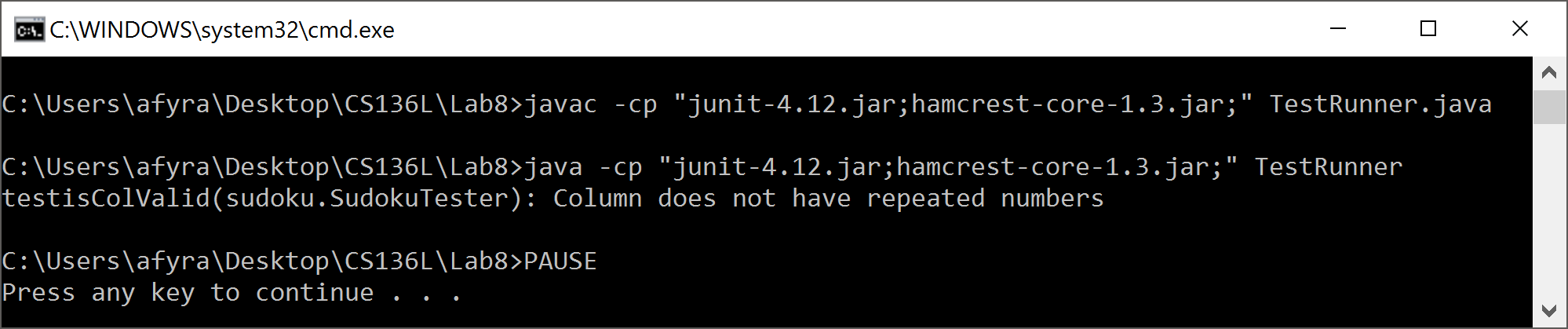


testisColValid-

Pass-

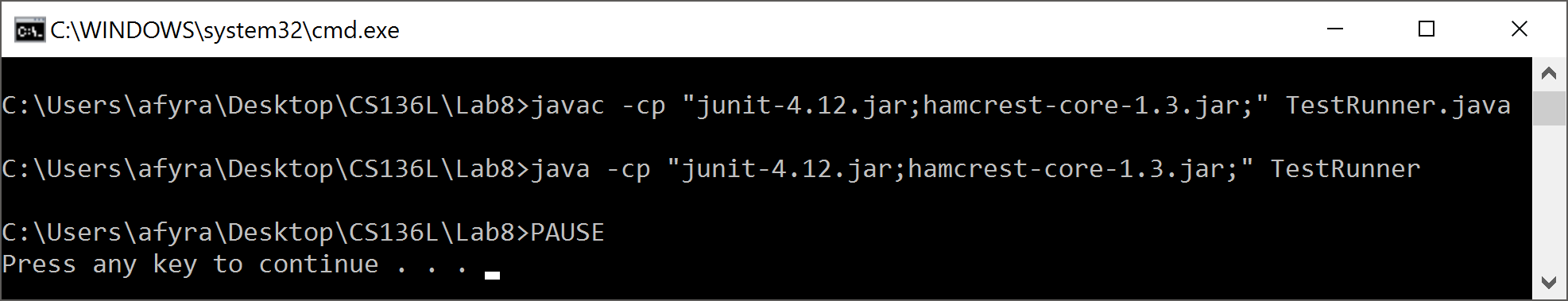


Fail-

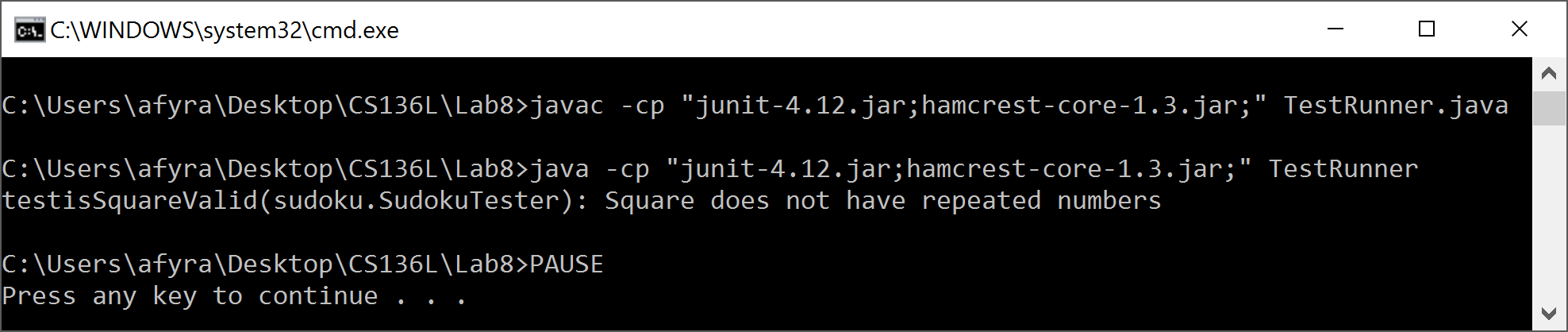


testisSquareValid-

Pass-

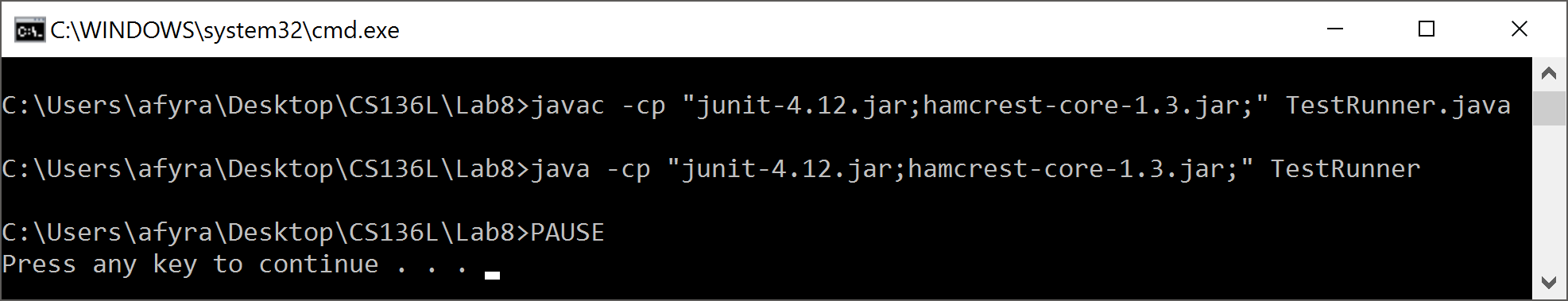


Fail-

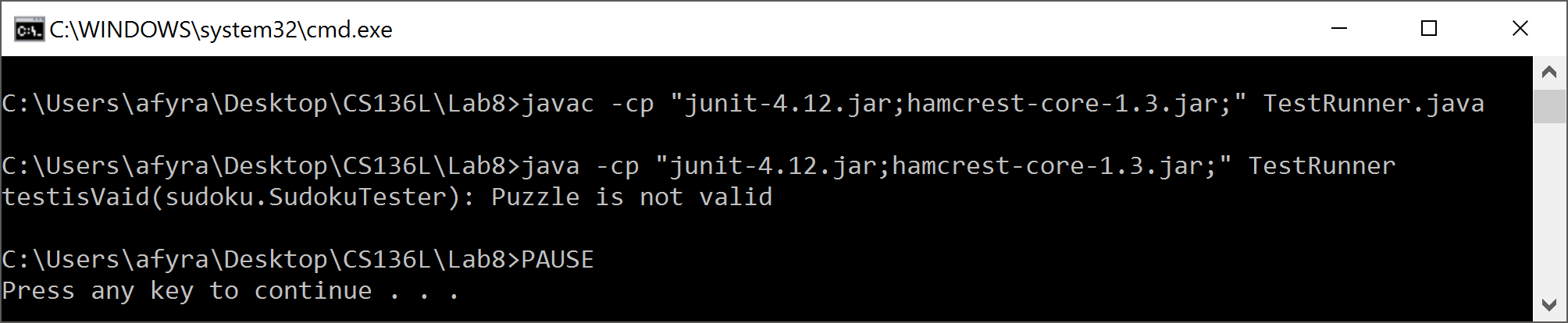


testisValid-

Pass-

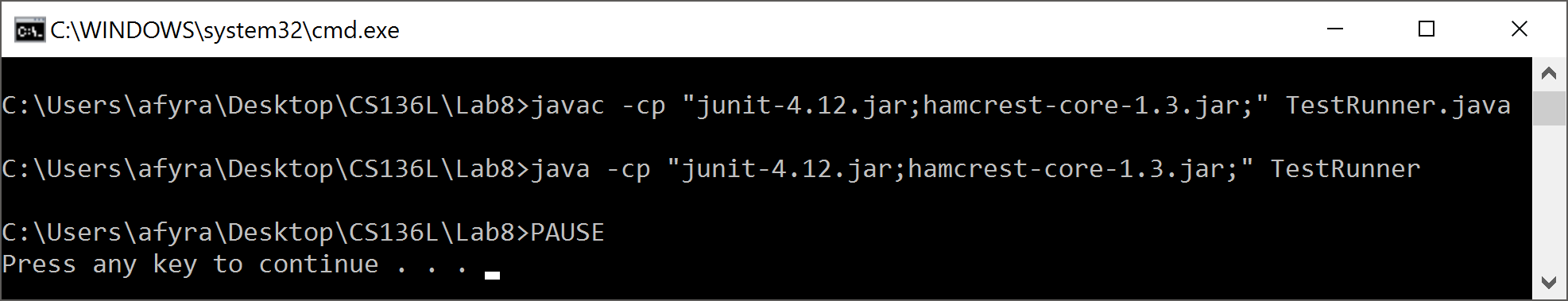


Fail-

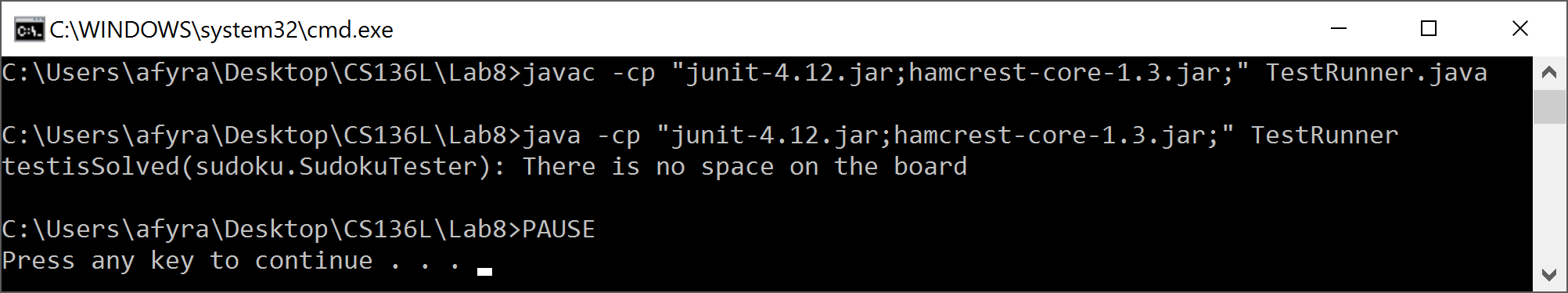


testisSolved-

Pass-

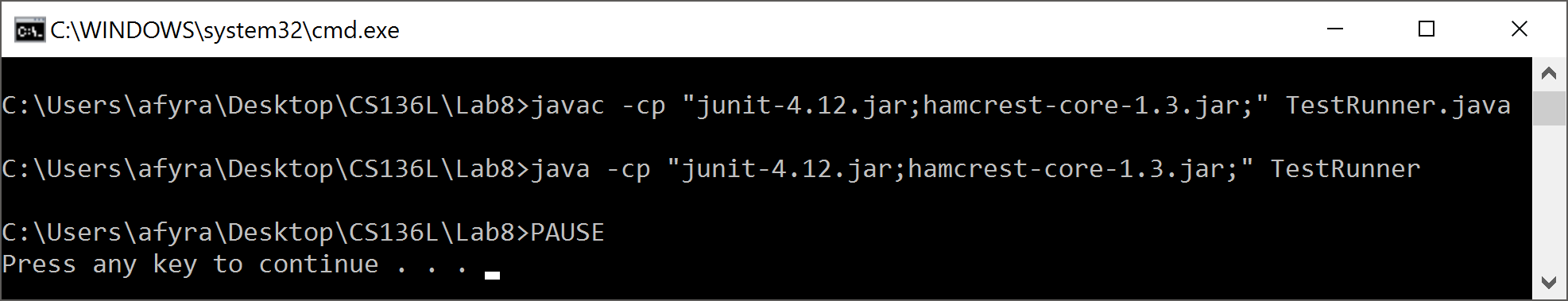


Fail-

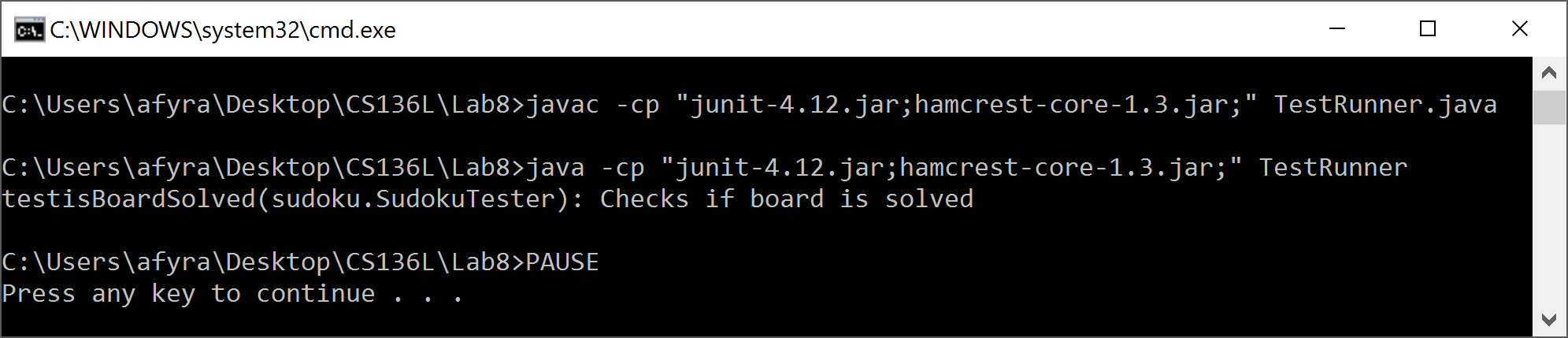


testisBoardSolved-

Pass-



Fail-



**Reflection & Refactor:**

This lab cleared up a lot about JUnit for us. We had both never used JUnit before this lab and did not understand much about it. The given clock package was quite helpful to us. Being able to see how JUnit tests are implemented into a file turned out to assist us greatly on this lab. During the class period, we discussed if the isValid/isSolved tests should assertTrue or assertFalse. We decided in the end that they should assertFalse, because they are testing if the isValid method is working or not. They now will tell the user if the inputted puzzle isValid. For example, if a number was repeated, the test would pass and tell the user nothing. If a number was not repeated, the user would be informed that there were no repeats in the square, row, or columns. We believed this would show the most useful information for the user.

Overall, we are quite pleased with our solution. We found it was quite easy to implement the lab requirements into our solution. We needed to do additional research on how to get a test to check for a thrown exception, but a quick search and we knew how to implement it. This lab has helped to improve our use of JUnit and cleared up any concern we had about using it.