Homework #2: Lizard Spock Yahtzee

CPSC 224 - Software Development

Note: There is no new GitHub Classroom link for this assignment. Use the same repo you used for HW1, just continue your work from your last HW1 commit. There's instructions below to tag your code for grading.

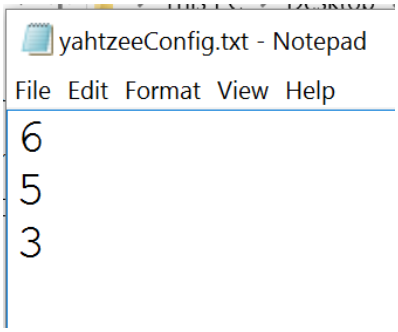
Why is the assignment called Lizard Spock Yahtzee? Watch this video:

https://www.youtube.com/watch?v=Kov2G0GouBw

THE PROBLEM:

Extend your solution to Homework #1 as follows:

Use a file named yahtzeeConfig.txt that's in the root of the repo. The file contains three records. The first record stores the number of sides on each die, the second record stores the number of dice in a hand, and the third record stores the number of rolls allowed per turn. A sample file should be attached to the Blackboard assignment. Note that the values shown are set up to play traditional Yahtzee.



The file should be read at the beginning of your program to retrieve the game configuration values used the last time a game was played. The current values should be displayed and the user given a chance to change the configuration. If the user elects to change, they should be prompted for new values which should then be written to the file. The execution of this portion of your solution should look something like this:

You do not need to play a full game of Yahtzee since that can wait until Homework #3. Your solution is complete when you can play a single hand and calculate and display possible scores for each scorecard line, just like in Homework #1.

```
you are playing with 5 7-sided dice
you get 3 rolls per hand

enter 'y' if you would like to change the configuration y
enter the number of sides on each die 6
enter the number of dice in play 5
enter the number of rolls per hand 3
```

In traditional Yahtzee, the upper scorecard has a line for each possible die value from 1-6. In Lizard Spock Yahtzee, the upper scorecard will need to adjust so that there is a line for each possible die value based on the number of sides on the dice being used.

Although not required in Homework #2, you may want to invest time in the design of lower scorecard scoring. In future assignments you may want the lower scorecard to dynamically adjust for > 4 of a kind and straights longer than a large straight when the sides and number of dice in the configuration make that possible. That said, you can still use the same set of rules and score lines in the lower scorecard from the stock Yahtzee rules.

If you followed the logic in yahtzee1.cpp when implementing HW1, you will find the maxOfAKindFound and maxStraightFound functions readily extend to deal with other than 5, 6-sided dice (5d6 hands). Other parts of the code might not extend as easily.

Phase 2 of homework: Adding tests

For this assignment, we must start adding tests to your code. This means adding JUnit unit tests in the src/test/java/edu/gonzaga/ directory. You can either add more tests to DieTest.java or add another test file for one of your new classes.

You must add at least three unit tests to your project, either in DieTest or your own test file. We will talk about this some in class, but feel free to find additional materials on unit testing[1] with JUnit.[2]

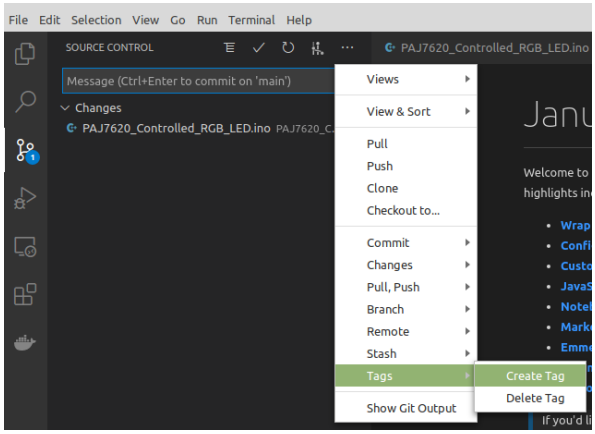
These tests should evaluate the behavior of your classes beyond just "does the class instantiate with the default values".

NOTE: the Git server checkins also build & run the unit tests with your project. You can see them run with HW1.

[1] https://www.vogella.com/tutorials/JUnit/article.html
[2] https://junit.org/junit5/

Project operations and development processes:

- To kick off this assignment, make sure to tag the commit with your work from the last homework, and then continue to work from that commit for this homework. These instructions will name a commit with a tag so you can easily check out the old versions of your project as we progress through each stage.
    - On command line:
        - git tag -a “HW1” -m “my message about it being ready for release / turned in as homework 1”
        - git push --tags
    - Or In intellij:
        - Menu: VCS->Git->tag
        - Then: VCS->Git->push (checkbox for “push tags”)
    - Or in VSCode:
        - Version control plugin on left
        - Top of Source Control bar, use the “...” menu
        - Select Tags->Create Tag
        - Enter “HW1”, then push the repo with tags



- After your code is ready, merged to main, and you’re done with the project, tag the commit with the tag “HW2”
    - On command line:
        - git tag -a “HW2” -m “my message about it being ready for release”
        - git push --tags
    - In intellij:
        - Menu: VCS->Git->tag
        - Then: VCS->Git->push (checkbox for “push tags”)
- It’s optional for this assignment, but you should start looking at how branching works in Git.
    - The workflow is illustrated here: https://guides.github.com/introduction/flow/
    - If you need practice on using this workflow, please create a throw away repo on GitHub to practice on. They’re free and you can try out the various commands, tools, web interfaces, and everything else.
    - Note: IntelliJ will render the git repo’s network of commits and branches under the git log area:
        - See image attached at end of this document for an example
        - VSCode does the same thing under Source Control -> “Git Graph” button at the top

SUBMISSION:

When formatting and turning in your assignment be sure to follow: Dr. Sprint's Coding Standards, which include the use of javadoc style documentation throughout your code.

Your code should be submitted via GitHub Classroom. Your assignment repository should also contain a .pdf document describing the major design and development considerations encountered in implementing your assignment. The document should be named "Homework 2 Summary.pdf" and saved in the Summaries directory within the project repo. This document should be written in a narrative style and should include:

- A summary of the goal or purpose of the program in your own words.
- An overview of the general design you chose for your program.
- A short description of one of your unit tests and how that test demonstrates a correct behavior for your project.
- A UML Class Diagram that includes all classes that are part of your solution. Diagrams must be produced by a software tool and not hand drawn. Primary choices are draw.io and MS Visio.
- A description of any major design and/or programming issues, why these were the major issues, how you addressed them, and why you addressed them the way you did.
- A retrospective of what you would have done differently if you had more time.

IMPORTANT:

This is still an individual assignment. Everything you turn in should be the result of keystrokes done by you. You should not consult nor use any existing classes related to Yahtzee or dice. You should not share your code nor look at the code of your classmates. It is ok to have generalized discussions about java and to have high-level design discussions about the classes, attributes, and methods necessary for the solution.

Handling file paths in java: a quick note

When you try to open a file you need to specify the path to the file. It doesn't need to be absolute, but it does need to exist. The default is the current working directory (called dot "."), but you can put either an absolute path like this in windows or UNIX respectively:

- c:\users\myusername\projects\HW1\stuff\file.dat
- /home/myusername/projects/HW1/stuff/file.dat

The problem is that those are absolute paths. They won't run if you move the project to another directory, and especially a different computer with a different username. The more common approach is to use relative pathnames:

- ./../file.dat

  That will look one directory up from the current working directory ($CWD) for file.dat

You can get the current working directory that the project was launched from using the System.getProperty("user.dir") interface:

```
class Main {
   public static void main(String args[]) {
      System.out.println("Working Directory = " + System.getProperty("user.dir"));
   }
}
```

You could then include that string in your file path since it would be correct for any given operating system or user who runs it since it's set at launch time by the JVM.

The testing of your assignment is done on LInux servers, so Linux paths are preferred. If you're building and testing on a Windows machine, you'll need to use a cross platform approach to building any file paths you end up using. See this article on how to make a path to various directories using the Java libraries instead of hard coding one:

https://www.sghill.net/how-do-i-make-cross-platform-file-paths-in-java.html

---

Reading in files (just like user input);

Handling files is best done with the Scanner library. If you're using something else and it compiles on the GitHub Actions workflow, then that'll be fine too, but Scanner is a very common tool for file I/O:

https://www.geeksforgeeks.org/scanner-class-in-java/#:~:text=Scanner%20is%20a%20class%20in,and%20strings.&text=next()%20function%20returns%20the,first%20character%20in%20that%20string.