Assignment 2 C212 FA22

Last modified 2022-09-09 5:00 PM

Summary

The gist of this assignment document's contents are as follows:

- Program Description: turn-based guessing game
    - Sample program output
- Hints, edge-cases, and design considerations
    - Should we treat numbers as strings or integers?
    - Edge cases we consider optional
    - Running a game with a while-loop
- Deliverables & Reminders
    - Use a project named `Assignment2`, and a class named `CodeCracker`
    - Submit a .zip file containing, at minimum, `CodeCracker.java`
- Rubric
    - Also visible on Canvas assignment page
- Changelog
    - No updates yet…

Program Description

Write a program, `CodeCracker.java` - another guessing game for two players – but different from Assignment 1's game. You are going to write a program that generates a 3 digit number at random and let the players guess this code in turns. Each time the player inputs his guess, the program responds with a score reflecting how close the player's guess was from the secret code. At the start, the program should ask for the player's names and do a coin toss to decide who goes first.

The objective of the game is for the player to guess the number correctly using the information revealed each turn.

You can generate and store the random number in any way appropriate. It's up to you to decide whether you want to represent the random number using Strings or as an integer in the interval [100, 1000). If you want to work with integers, then, of course, you can't have numbers like 071 as possible secret numbers. This is fine. See the Hints section with more information on these two different but equally valid approaches to completing the assignment.

Sample program output

User input in bold and italics.

```
Welcome to Code Cracker by <Your name>
Enter player 1 name: John
Enter player 2 name: Mary
Can you crack the code?
The computer has thought of a 3 digit number.
Flipping a coin to decide who goes first.
Mary gets the first turn…
Mary, make a guess: 888
You guessed: 888
You got 0 correct: XXX
John, make a guess: 145
You guessed: 145
You got 1 correct: X4X
Mary, make a guess: 349
You guessed: 349
You got 2 correct: X49
John, make a guess: 549
You guessed: 549
You got 3 correct: 549
John wins!
Bye bye!
```

Hints, edge-cases, and design considerations

Should we treat the secret number as a String or int or something else? Well, there are multiple ways to solve this and all of those choices would work. In fact, the assignment is just about as difficult regardless of whether you choose to use Strings or ints or multiple of them.

Your first instinct might be to generate an integer that's smaller than 1000 but bigger than 99. However, consider that we're not really concerned with doing any math on this secret number, so we don't need to deal with the tradeoffs we'd face with making the secret number just one integer. But we've already given enough hints on this page :)

If you do use one integer to store the secret number: as previously stated, you don't need to worry about not including numbers starting with one or many zeroes not being possible to generate.

As far as other edge-cases we won't require you to deal with, here are a few:

- You don't need to prevent a user from trying to input a non-numerical guess
- You don't need to ensure user input is a three-digit number

In other words, assume users always input three-digit numbers when asked, so there aren't really many edge-cases regarding user input – except, for example, if a user guesses right the very first time. If you want the experience dealing with these optional edge-cases, you're welcome to try handling them. As long as what you submit still runs as expected in the general case, you won't lose credit for attempting to handle edge-cases. Since future assignments likely will require you to handle these, it would be a good idea to try them now so you can get feedback on your approach.

If you're not sure how you'd run a game like this where the users can take potentially infinitely many turns: recall we saw something similar in the demo during the third week's lab. You can view this demo file on the Lab03 Canvas assignment page. Note that you should model your solution on the "better approach" in this file and not the "bad approach"! This is reflected in the rubric: you should not collect guesses outside the game's main while loop.

Deliverables & Reminders

Submit a .zip file on Canvas. This .zip file can either be your zipped project folder from your IDE, or a zipped folder containing your GuessingGame.java file.

Your IntelliJ project can have any name appropriate – `Assignment2` would be a good choice.

In addition to solving the problem, your code should follow these usual requirements:

- All variables should have informative names.
- Your code should be clean, presentable, and consistent with Java conventions which includes:
    - Classes are named in PascalCase, while variables are in camelCase
    - Be consistent with your spacing and opt for readability (see Assignment 1's PDF for an example).
    - Use correct indentation for both code and curly braces.
        - If your choice of indentation or lack of curly braces is reasonable – for example, leaving out curly braces for an if-statement with only one line of code in the body – you won't be penalized.
    - (As always, ask your grader for clarifications. We won't "pre-grade", but we can answer questions on whether parts of your code meets cleanliness expectations.)
- Every program must have comments. There should be header comments in your program including your name, a brief introduction of the program, and a short step-by-step description of what the program does and how it does it. You should have at least one comment in-line with the code besides that.

Your input/output should match the examples. You'll want to check all the edge-cases when you test your solution, because the graders will be doing that. The edge-cases we are considering only optional (i.e., you can assume these don't happen) are listed above.

Ensure your submission contains the necessary file(s) and is the correct version of your code that you want graded. This is your warning. You will not be allowed to resubmit this assignment after the due date if you submit the wrong file.

Remember that this is graded on accuracy and you are not allowed to collaborate on your solutions with others. Plagiarism or cheating will be dealt with seriously.

Rubric

A rubric is also available on the assignment page on Canvas, but here is an outline of the point breakdown for different categories.

- (20pts) Code cleanliness, conventions, and comments
  - (20/20pts) for >95% correctness and all required comments
  - (15/20pts) for repeated minor mistakes or one egregious mistake, or for missing either overall or inline comments
  - (10/20pts) for code that is sloppy and hard to understand or is missing all comments
  - (<=5/20pts) for code that has no comments, is hard to understand, and consistently does not follow Java conventions
- (50pts) Program works in the general case
  - (5pts) for collecting users' names
  - (10pts) for determining the first player's turn via a "coin-flip"
  - (15pts) for collecting user guesses in a loop
    - (-8pts) if guesses are ever collected outside the loop
  - (5pts) for determining and displaying the count of correct digits
  - (10pts) for determining and displaying the revealed digits
  - (5pts) for determining and displaying the winner
    - (-2.5pts) for checking for a winner inside the loop body
- (10pts) Program works in all non-optional edge-cases
  - (-5pts) for each edge-case that was acknowledged and had a fix attempted, but not completely addressed
  - (-10pts) for each edge-case overlooked in solution
  - No penalty for attempting to address an optional edge case. Graders: leave feedback on such attempts.
- (20pts) Valid, logical, and consistent approach for generation of, storage of, displaying of, and comparing the user guess against the secret number.

Changelog

If any corrections, clarifications, etc. are made to the assignment after its release, they will be listed here. Any changes more substantial than simply fixing typos will get corresponding Canvas announcements. That way, you don't need to keep checking whether updates have occurred.