intro2cs/introcsp/introcst - Exercise 3

Objectives

This exercise's purpose is to get you a little more comfortable with basic programming in Java. The exercise will focus on working with loops and using a given API.

Exercise definition

Mastermind or Bulls and cows is a code breaking game for two players.
Your task in this exercise is to write a Java program that plays a simplified version of the game. The computer will always play the side of the *codemaker* while the user will play the *codebreaker*. Specifically, you will implement the logic of the game, as described below, inside the **`main`** method of the **`Mastermind`** class.

The game you will implement has three user-specified parameters:

1. The length of the code, which must be a positive integer.
2. The number of possible values in each position in the code, which also must be positive. The possible values consist of a consecutive ascending sequence of integers starting with 0.
3. The maximum number of guesses the codebreaker has to discover the code. This number also has to be positive.

The program you will write will need to:

1. Play the side of the *codemaker*,
2. Interact with the user; and
3. Keep track of the user's performance:
    a. The number of games the user has played.
    b. The number of games the user has won.
    c. The user's win rate. (Calculated as the number of wins divided by number of games.)
    d. The average win length. (Can be calculated as total number of turns in wins divided by number of wins.)

To help you with your task we provide you with a number of classes, whose API can be found here. Please note that in order to use these APIs, you will have to add the following line in the beginning of your code:

```
import il.ac.huji.cs.intro.mastermind.*;
```

Creating objects: in your program you will need to create objects of two classes that we provide: `MastermindUI` and `Code`. In order to create these objects you will need to use two `static` classes that we also provide: MastermindUIFactory and CodeGenerator, respectively. This mechanism is necessary for us to be able to write the automatic testers that will test your program (these methods will return different objects based on whether your program runs within our testers or by itself). For example, you can create a new `MastermindUI` object like this:

```
MastermindUI ui = MastermindUIFactory.newMastermindUI();
```

and a new `Code` object like this:

```
Code code = CodeGenerator.newCode(4, 6);
```

The statement above will create a new random code with 4 digits from [0..5]

The flow of the Mastermind game you will write is outlined below. All user interaction must be performed through the `MastermindUI` class (UI is a common acronym for *User Interface*): please look for the appropriate method for each interaction in the MastermindUI API.

1. A new `MastermindUI` object is created using the MastermindUIFactory API.
2.
   a. The user is asked for the following game parameters, in order:
      i. the length of the code,
      ii. the number of possible values in the code,
      iii. the maximal number of guesses per game.

      Each of these values must be positive, and if a nonpositive value is received, the program should display an error message and ask for the value of that parameter again.
   b. The UI should be reset using the parameter values provided by the user in step a. above.
   c. All game statistics are reset to their initial values. Note that these values are not managed by the `MastermindUI` object. You should store and update them separately.
3. A new random `Code` is created using the CodeGenerator API.
4. One or more turns take place, until either the user correctly guesses the code, or the maximal number of guesses were used. In each turn, the user enters a guess and in response is presented with the result of this guess. (See the `askGuess()` and the `showGuessResult()` methods in the MastermindUI API.)
5. The game outcome should be shown, and the user performance statistics should be updated and shown. (See `showStats()` method.)
6. The user is asked whether he or she wishes to play another game. If the user declines, then the UI is closed (important!) and the program should complete. If the user chooses to play another game, then he or she is given the option of changing the game parameters. If he or she chooses to change them, then execution continues from step 2 above; otherwise the game board is cleared and execution continues from step 3 above;

Additional notes

1. Division of the code into methods is still not required, but is permitted. Note, however, that you may be penalized for incorrect usage of methods, if you choose to use them.
2. You may not use arrays in this exercise.
3. The **`askGuess()`** method may return values which are outside of the legal range of code values. These can be considered to be slots left blank in the guess.
4. When there are no wins, the average win length cannot be calculated. You should send the value **`Double.NaN`** as the average to the **`MastermindUI.showStats()`** method in such a case. The regular calculation of these values should work, as **`0.0/0.0`** returns **`Double.NaN`**. (This is unlike the integer division **`0/0`** which fails.)
5. In this exercise, we do <u>not</u> specify the exact format of the messages you should display to the user. We do, however, require the following:
    a. When you ask the user for the length of the codes, the message must contain the word "length" or the word "Length".
    b. The message for reporting that the user won a game must contain the word "won" or "Won" followed somewhere in the message by the number of turns the user needed.
    c. The message for reporting that the user lost a game must contain the word "lost" or "Lost".
    d. An error message does not have to contain an indication that it is an error. That information is provided by the UI.

    You may, if you wish, run the school solution, and copy the messages from there.
6. You are invited to share your own UI implementations (using the same API) in the forum. We will choose a the best UI and publish it prominently on the course site.

Suggested plan of action

1. Start early!
2. Read the description of the game (Mastermind or Bulls and cows) to remind you how it works.
3. Next, run the school solution to see what your program should look like to the user.
4. Acquaint yourself with the provided APIs.
5. Write the code for a single turn: asking the user for a guess, and responding to the provided guess. Debug.
6. Extend the above code to a full single game. Debug.
7. Extend the above code to a sequence of games. Debug.
8. Add code for gathering and reporting performance statistics. Debug.
9. Run testers. You guessed it: debug!
10. Enjoy the rest of your week!

School solution

An executable version of the school solution can be invoked from the lab computers by typing:

```
~introcsp/bin/ex3/mastermind
```

on the shell command line.

Submission and Testing

- You should create the following files:
    a. **`Mastermind.java`**
    b. **`README`** (as explained in the course guidelines on the course website)
- Create a JAR file named **`ex3.jar`** containing only these files by invoking the shell command:
  ```
  jar cvf ex3.jar Mastermind.java README
  ```
  The JAR file should not contain any other files.
- It is recommended to check your JAR file by copying it to an empty directory, and running the automatic testers by executing the command:
  ```
  ~introcsp/bin/testers/ex3 ex3.jar
  ```
  and verifying that all the tests pass successfully.
- You may also download the tests JAR from here. Extract the contents of the JAR file using the shell command:
  ```
  jar xvf ex3testing.jar
  ```
  and follow the instructions in the file called **`TESTING.`**
- You should submit the file **`ex3.jar`** via the *Upload File* link under the ex3 link on the course moodle page.