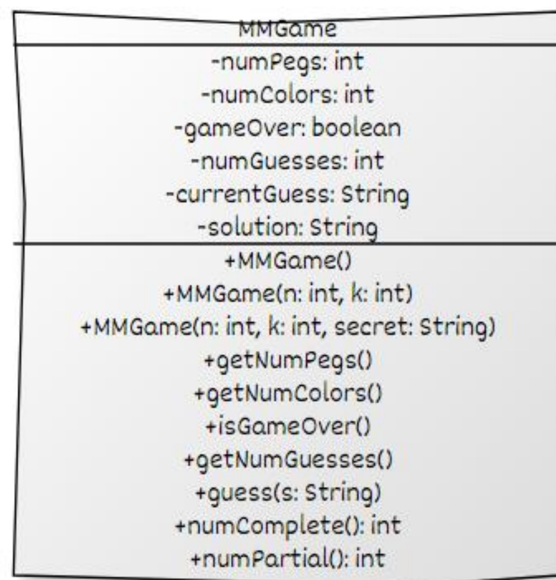
Object-Oriented Design
COMP 2000
Spring 2024

Lab 10-2

Mastermind Part III

Part III is entirely focused on implementing `MMGame`, which represents the Model part of the MVC design. Based on the specification in Part II, your class diagram may have looked similar to the following for `MMGame`:



There is a Javadocs API available for this class here. For this part of the lab you should use the class diagram above rather than what you turned in for Lab 10-1 to make sure that you have everything you need for part IV.

To Do

Create a project directory for MasterMind. Within the src directory create a mastermind package. Then implement the MMGame class as a member of this package, according to the class diagram above, the Javadoc specification, and the description of it from Lab 10-1.

You may want to add a private method to your class that selects the secret code. You can then call this method from the constructor when you want the secret to be generated.

See implementation notes, below, for additional suggestions.

Testing

I have also provided a test driver class called `MasterMind.java`. It is a GUI application that uses JFrame and other Swing/AWT components that looks like the image below. (This is not the full GUI that Lab 10-1 described, but an alternate view that is a quick way to interact with your model class.)

In this interface, n is the number of pegs and k is the number of colors. This test driver only uses the two-parameter constructor for `MMGame`. The secret codes are strings of numerical digits from 0 to k-1. So for example the secret might be "3205".

When you click New Game, a random string of digits is created. You cannot see it but are invited to guess what it is. To make a guess, enter your guess in the box provided and click Make Guess. Your guess must conform to the restrictions imposed by n and k.

After you make a guess, you get feedback in the two boxes complete and partial. Each box will contain a number in the range 0 to n. The number in complete is a count of how many digits in your guess were correct and in the correct position. The number in partial is a count of how many digits in your guess were correct but in the wrong position. These are your clues to use in forming the next guess. When the value in complete is the same as n and partial is 0, you have correctly guessed the answer!

Implementation Notes

Make sure to include your name in comments at the top of your source code.

Begin by declaring all the instance variables. Then implement the three constructors, and the getter methods. While developing your code you might find it helpful to have the constructors print the secret code to the console using `System.out`. Finally, implement `guess()`, `numComplete()`, and `numPartial()`, in that order.

`guess()` **does not evaluate the guess but determines whether or not the user's guess correctly follows the required structure: correct length and every character is valid. Don't forget to return true or false. Also,** `guess()` **should do nothing if** `isGameOver()` **is true.**

`numComplete()` **compares the current guess to the solution character by character, counts the number of complete matches (matching both value and position) and returns that count. This requires a loop but is pretty straightforward.**

`numPartial()` **compares the current guess to the solution character by character, counts the number of non-complete, partial matches (matching value but not position) and returns that count. This is by far the most logically complex part of this lab, requiring some kind of nested loops and possibly auxiliary arrays. It will require some creative problem solving to get correct. Do not attempt this until all the rest of the lab is finished and tested.**

To Turn In

Email `MMGame.java` **to prof. stucki as an attachment.**