Computing and Algorithms I

Project 4

CS-101 **Fall, 2017**

Project 4 will be an extension of projects 2 and 3 and will include design and code. The design portion will be worth 50 points, the code portion will also be worth 50 points. In project 4 you will create a GuessingGame class from which you will create GuessingGame objects. There will be some changes to the user experience when playing this game which will be discussed later in this document.

Requirements

The Class `GuessingGame`

You are to write a class named `GuessingGame`, an instance of which is the representation of the state of a guessing game which is being played by the user. The following constants and fields (represented by public class constants and private instance variables) are required in your object:

- EASY_GAME – this class constant will have the value true.
- DIFFICULT_GAME – this class constant will have the value false.
- DEFAULT_MAXIMUM_RANGE – this class constant will have the value 10.
- GAME_WON – this class constant holds an `int` value for the instance variable gameState.
- GAME_LOST – this class constant holds an `int` value for the instance variable gameState, note this value must be different from the one in GAME_WON.
- GAME_IN_PROGRESS – this class constant holds an `int` value for the instance variable gameState. The value stored in GAME_IN_PROGRESS must be different from the values store in GAME_WON and GAME_LOST.
- gameState – this instance variable holds the state of the game. Only three values are allowed: GAME_WON, GAME_LOST, and GAME_IN_PROGRESS.
- numberToGuess – this field holds the "randomly" generated number which the user is trying to guess.
- numberOfGuesses – this field holds the number of guesses the user has made so far.
- largestPossibleNumber – the user will be guessing a number from 1 up to some maximum, this field holds that maximum.
- maximumNumberOfGuesses – the user will not be allowed unlimited guesses, this field holds the maximum number of guesses the user is allowed to make. This value is an integer.

- currentMinimumRange – At any point, the user has narrowed the search range by making a guess. This number is the largest guess made so far which is too low. For example, if numberToGuess equals 45 and the user has guessed 40, but has not guessed any number between 41 and 45 inclusive, then currentMinimumRange will be set to 40.
- currentMaximumRange – Similar to currentMinimumRange, this is the smallest guess made so far which is too high. Initialize this variable to Integer.MAX_VALUE so the test in the mutator method for this variable will work appropriately.
- guessTooLow – this field is a boolean value. Its value will be true if a guess has been made and the value is too low. Its value will be false if a guess has been made and the value is too high. If a correct guess has been made, or if no guesses have been made, guessTooLow will be false.
- easyGame – this field is a boolean value. If EASY_GAME, this is an easy game. If DIFFICULT_GAME, this is a difficult game.

The following constructors and methods must be implemented:

- `GuessingGame(int largestPossibleNumber, boolean difficulty)` this constructor will call the mutator methods for all the fields of the class. These mutator methods will be described later.
- `GuessingGame(int largestPossibleNumber)` this constructor is equivalent to calling the previous constructor with a value of EASY_GAME for difficulty.
- `GuessingGame(boolean difficulty)` this constructor is equivalent to calling the constructor `GuessingGame(int largestPossibleNumber, boolean difficulty)` with DEFAULT_MAXIMUM_RANGE for the value of largestPossibleNumber.
- `GuessingGame()` this constructor is equivalent to calling `GuessingGame(int largestPossibleNumber, boolean difficulty)` with DEFAULT_MAXIMUM_RANGE for the value of largestPossibleNumber and EASY_GAME for difficulty.
- Every instance variable will have an accessor method.
- setNumberToGuess(int number) a constructor will call nextInt from a Random object appropriately and pass the result to this method. Make this a private method.
- setNumberOfGuesses(int number) this method will also be private. A constructor will call this method with the value 0.
- setLargestPossibleNumber(int number) – largestPossibleNumber is set to the maximum of number and DEFAULT_MAXIMUM_RANGE.
- setEasyGame(boolean difficulty) – copy the value of easy into the instance variable easyGame.
- computeMaxNumberOfGuesses() – call this private method in a constructor after calling both setEasyGame(boolean) and setLargestPossibleNumber. This method will compute the number of guesses allowed based on whether this is an easy or difficult game. If the game

is easy, the result is one half of largestPossibleNumber (round up). If the game is difficult, write a while loop to continually divide the value stored in largestPossibleNumber by 2 until the value becomes 1. Count the number of divisions (iterations of the while loop) then add 1. This value will be returned for a difficult game. Note, do not change the value stored in the instance variable largestPossibleNumber for this calculation.

- setMaximumNumberOfGuesses(int number) – in a constructor, pass the value returned by computeMaxNumberOfGuesses() to this method.

- setCurrentMinimumRange(int min) – in a constructor, pass 0 as min to this method. When the user has guessed a number that is too low, replace currentMinimumRange with the guess if the guess is greater than currentMinimumRange. (This method will determine whether or not to change the value in the instance variable).

- setCurrentMaximumRange(int max) – in a constructor, pass largestPossibleNumber to this method. When a user makes a guess that is too high, but smaller than currentMaximumRange, replace the value in currentMaximumRange with the guess. (This method will determine whether or not to change the value in the instance variable).

- setGameState(int state) – in a constructor, pass the value GAME_IN_PROGRESS to this method. Note that if the user wins the game, this mutator will be called with GAME_WON. If the user loses, this mutator will be called with GAME_LOST.

- setGuessTooLow(boolean tooLow) – in a constructor, pass the value false to this method. When guessing, pass the value true to this method if the guess is too low, otherwise pass the value false to this method.

- makeGuess(int guess) – this method handles a user guess. If the variable gameState has the value GAME_WON or GAME_LOST, this method will do nothing. If gameState has the value GAME_IN_PROGRESS, this method will increment the instance variable `numberOfGuesses`. If the guess is correct, this method will call setGameState(int state) with the value GAME_WON. If the guess is too low, set the instance variable `guessTooLow` to true, and call setCurrentMinimumRange(guess); note that setCurrentMinimumRange should only modify currentMinimumRange if guess is higher than its present value. If the guess is too high, set the instance variable `guessTooLow` to false, and call setCurrentMaximumRange(guess); note that setCurrentMaximumRange should only modify currentMaximumRange if guess is lower than its present value. If numberOfGuesses and maximumNumberOfGuesses are equal and the game has not been won, then this method will call setGameState(int state) with the value GAME_LOST.

- hint() – this method will return a String instructing the user to choose a number between currentMinimumRange and currentMaximumRange.

- quit() – this method will call setGameState(int state) with the value GAME_LOST.

- toString() – this method will report the state of the game. If the game is still being played, the return value will state the number of guesses remaining for the user. If the game has been won, the return value will contain numberOfGuesses and numberToGuess in a congratulatory sentence. If the game has been lost, numberOfGuesses and numberToGuess will be in the return value in a respectful sentence.

The Class `GamePlayer`

You are to write a class named `GamePlayer`, an instance of which will interact with the human player and the `GuessingGame` object in order to play a game.

`GamePlayer` will have two instance variables, `guessingGame` of type `GuessingGame` and player of type Scanner.

`GamePlayer` will have an accessor and mutator method for the instance variables.

The constructor, `GamePlayer(Scanner player)` will communicate with the human player of the game by using the Scanner object to read the player inputs. This constructor will present a menu of options to the user such as:

```
1.  choose difficulty level.
2.  pick upper bound for guess.
3.  play game.
```

The user will be instructed to choose an option (by typing in the number of the option). If option 1 is chosen, the user will be asked to choose either easy or difficult. If option 1 is never chosen, the difficulty level will be `GuessingGame.EASY_GAME` by default. If option 2 is chosen, the user will be asked to input the upper bound for the game. The constructor will continue to display the menu and interact with the user (so that difficulty level and upper bound may be changed any number of times) until the user selects option 3 to play the game. The constructor of `GamePlayer` will call the appropriate constructor of `GuessingGame` (the one with the fewest arguments) and set `guessingGame` to reference that object before returning.

The method `play()` will interact with the human player using the Scanner object initialized in the constructor. This method will help the player play a game. This method will continually present a menu of options to the player, read the option selected by the player, and act on it. One of the options will cause play to execute the return statement. An example of such a menu is:

```
1.  make a guess.
2.  get a hint.
3.  print statistics.
4.  quit this game.
```

Option 1 will request the user to enter a guess, and will pass this guess to the GuessingGame object's method makeGuess. If this guess causes the user to either win or lose the game, option 1 code will print the statistics (value returned by the toString method of the GuessingGame object).

Option 2 will print the value returned by the GuessingGame object's hint method.

Option 3 will print the value returned by the GuessingGame object's toString method.

Option 4 will check if the game state is GAME_IN_PROGRESS. If so, the GuessingGame object's quit method will be called. Option 4 will print the statistics (as in option 3) and then will return.

The Class `PlayGame`

You are to write a class named `PlayGame` which contains the main method. The code in this class will create a Scanner object to read from the keyboard. There will also be a loop which will construct a `GamePlayer` object and call the method play() of that object. This loop will ask if the user wants to play another game after the method play() returns. If the user wants to play, the code in this loop repeats; otherwise the application will end.

Requirements

You will be writing a design document similar to the one you wrote for project 3. In particular, you will be writing 3 UML class diagrams,the class diagram legend, and a UML class interaction diagram. You will also write a data table for the classes `GuessingGame` and `GamePlayer`, which will list the instance variables and class constants. You will also write a data table and algorithm in the design document for each method in your design.

Data Table

Each method in your design will have a data table except for accessor methods.

Algorithm

Each method in your design will have an algorithm except for accessor methods. No method will have more than 40 statements.

Input

Design your solution to use a single Scanner object instantiated from the Scanner class to read input from the keyboard as input by the user. Note this Scanner object will be passed as an argument from main to the `GamePlayer` constructor.

Processing

The processing is cleary laid out in the project description.

Output

Design your output to write to the terminal using System.out methods. All communication with the user should be clear.

Deliverables

This project is due on Tuesday November 21. If you complete it and turn it in at the beginning of class on Monday November 20 you will earn a 5 point early bonus. There are two blackboard areas for this project: Project4a will be the area for you to turn in the pdf of your design document; Project4b will be the area for you to turn in the code for the project. Zip all of your java source files into one file named project4.zip. No paper prints are required for this project, all your work will be uploaded onto blackboard as described.

Grading

The portions of the design will have the following maximum number of points. Full points will be given to complete and correct designs.

1. UML class diagrams – 22 points
2. legend – 2 points
3. UML class interaction diagram – 3 points
4. data tables for classes `GuessingGame` and `GamePlayer` – 3 points
5. data tables for methods – 10 points
6. algorithms for methods – 10 points

The code will be graded on the following:

1. Program compiles and runs correctly – 25 points.
2. Style for a working program – 10 points.
3. Clear, easy to read and understand, output – 10 points.
4. Template for code followed – 5 points.