Term Project (two people, two weeks, two classes)

Submit Assignment

Due Jul 10 by 11:59pm **Points** 100 **Submitting** a file upload **File Types** zip and 7z

Heard about or played a number guess game called "**Bulls and Cows**"? If no, please see on a **wikipedia page (https://en.wikipedia.org/wiki/Bulls_and_Cows)** to learn how to play this old and simple game without computers.

Instead of 'Bulls' and 'Cows', people may use 'A' and 'B'. That is, '**2 Bulls 1 Cows**' is equivalent to '**2A1B**' and '**no Bulls 3 Cows**' is the same as '**0A3B**', so on and so forth.

But, no, this project won't ask you to make a Java console program of Bulls and Cows but one similar and we'll call it '**iGuess**', which also names the project. It is for sure our iGuess is a unique version of puzzle or guess game and there is no same 'iGuess' game you can find or google for a Java program of this iGuess. So, although you are welcomed to mine the web for 'iGuess', you are advised to spend time in developing yours rather searching existing ones and get disappointed.

I. How to play iGuess between your computer (the host) and you (the player)

This is the first part (or mode) of iGuess project. When completed, it allows a person (player) to run the program to play.

In this host mode, the program plays the role of **host** of the game and serves the person **player.**

When the program runs in the host mode, it should:

1. Create a 5-character random string as a 'secret word' with '@' always included but in a position randomly selected. That is, the string would look like '@xxxx' or 'x@xxx' or 'xx@xx' or 'xxx@x' or 'xxxx@', where each 'x' must be drawn from at least one or a mixture of two or all three of the following three sets of characters:
   - **{ V, W, X Y, Z }**
   - **{ 5, 6, 7, 8, 9 }**
   - **{ =, ?, %, @, $ }** - you may use "EQ-PAD" to help to memorize these five symbols by
     - E for 'eaual' or '='
     - Q for 'question mark' or '?'
     - P for 'percentage' or '%'
     - A for 'at' or '@'
     - D for 'dollar sign' or '$'
2. The secret word must contain no duplicates, i.e., every character is unique.

3. Because '@' must always appear in the 5-char string, the program won't select it again from the 3rd set shown above.
4. Keep the secret word in the program without displaying it unless, for testing purpose, '*****' is entered to request to reveal the secret word. Once revealed, the game is forced to be terminated and over.
5. Ask the player to enter a guess string to this secret word.
6. Reject guesses that are malformed except '*****' to reveal the secret word. Here malformed means any guess containing invalid characters. They should be rejected by showing a warning message followed by another guess.
7. Score the guess by printing three data: the accumulated count of guesses, the guess string, and a "score" for the guess. The score would be a 4-character string such as '0A2B', '1A3B', '3A1B', 2A0B', etc. Here 'A' is equivalent to 'Bulls' and 'B' is equivalent to 'Cows' in the *Bulls and Cows* game mentioned above.
8. Repeat step 6~8 until the secret word is revealed, or, the player wins the game, that is, the guess makes a score of '5A0B', meaning the guess hits and matches the 'secret word' completely.

II. How to play iGuess between you (the host) and your computer (the player)

This is the second part (or mode) of iGuess project. When completed, your program allows you to host the game (i.e., you create a secret word as defined in section I and score every guess) and it makes guesses until it wins. During the development or testing of your program, you may set a condition (e.g., the total count of guesses is already 10) to allow your program to request to reveal the secret word and quit the game. It's up to you to include this function or not in the program.

In this player mode, your program should:

1. Display a total count of guesses with a new generated guess, which should be a 5-char string with '@' always included and the rest four characters drawn from the same three character sets defined in section I above.
2. Wait for a score (e.g., '0A3B') to be returned and entered by the host, who is you. It's suggested the score is entered on the same line of the the count and guess displayed in the above step. So, it would look like
   - Guess #3   8@XYZ (mailto:8@XYZ)   1A2B
   - Guess #4   5@WYZ (mailto:5@WYZ)   1A1B
   - Guess #5   @8=%Z   4A0B
3. Repeat the above two steps until the guess wins a score of '5A0B' as shown below.
   - Guess #3   8@XYZ (mailto:8@XYZ)   1A2B
   - Guess #4   5@WYZ (mailto:5@WYZ)   1A1B
   - Guess #5   @8=%Z   4A0B
   - Guess #6   @8=6Z   5A0B   -- You win!!

Except for the very first guess, your program should implement a strategy or formula to generate a new guess based on all scores collected. A good strategy or formula should use as much as possible the information or hints provided by the scores to quickly approach the secret word. This would be the most challenging component to be implemented in the entire program.

As a clue for everyone, one possible method is to generate a list of all possible secret words (using all scores returned) that could be the next guess, then to prune the list by keeping only those words that would give an equivalent score to how the last guess was scored. Then, the next guess can be any word from the pruned list. Either the guess correctly hits the secret word or run out of words to guess, which indicates a problem with the scoring or a problem with the list itself.

III. How to play iGuess between your computer and another computer for a competition

This is the third part (or mode) of iGuess project. When completed, it allows you to run your program to play against someone's program running in another computer, each serves as a host as well as a player and takes its turn to make a new guess and score the competitor's guess based on its own secret word.

The winner of the competition should be one of the two programs that scores '5A0B' with less total count of guesses. A draw game is possible when both programs end up with a tie of count, i.e., they both win a game with same total count of guesses. In this case, it would require both programs to play one or more times until a real winner shows up, if it is desired.

In the competition mode, every program plays double roles: the host and the player. When it's your program's turn to make a guess (as a player), the generated guess should be passed and entered into the competitor's program (as a host), which scores the guess and displays the score. The score is then passed and entered into your program to generate next guess. Because this is a competition, so before you continue to play with a new generated guess, it would be the other program's turn to pass its guess to your program for a score to return. This turn-by-turn process is repeated until both programs get a score of '5A0B'.

There is no new code to write into the program for the competition mode because both host and player modes are already implemented in the previous two sections. However, this also means your program needs to offer a menu of three modes in the beginning of execution to allow one to be selected to run.

IV. Project Requirements

- **PROJECT TEAM** - This is a project for a small team of two. You will find someone in the class to be your project partner. You and your partner will work in a way you two agree with and ideally share the same grade of the project. In case your partner quits from the project for some reason (e.g., sick, withdrawn from the course, etc.), the grade will be split by the instructor after evaluation of the case.
- **PROJECT DEADLINE** - Midnight of July 10, Wed. Absolutely no extension.
- **PROJECT SUBMISSION** - Each team needs to submit (by either one of the team members) only one zip file, **SummerJava.zip**, containing a folder '**SummerJava**' with two files:
  1. **projApp.java** - a package of **COP2510.project.UI** is required for the class projApp, which is also the main class of the project. The main() method in this class serves as the *commander* or control center

of each game, so it should contain only the key statements to drive the game. Detailed game playing and operation logic, including keyboard input and screen display, are not expected to appear in the main(). In other words, an ideal main() should maintain only the least number of statements. All input/output (i.e., user interface) related statements should be supported by other methods in the same class, which is why this class is loaded into the package called **COP2510.project.UI**, and statements of game-related logic are supposed to be handled by the iGuess class below.

2. **iGuess.java** - a package of **COP2510.project.business** is required for the class iGuess. This class is responsible for fields and methods to run a game in all three modes, i.e., host, player, and competition. The class should be designed to be *UI-independent*, meaning there should be no statements in the methods of this class that use System.out.println() or Scanner object to catch keyboard input. All UI related operations should leave to methods of projApp class.

The textbook projects presented in Chapter 5 or 14 (and a few others) provide very good examples of how to organize and design the above two classes.

- **PROJECT LIMIT**- Only classes and packages introduced in this course or developed by your team are allowed to import and use in the project. For example, you may use ArrayList but not HashMap, you may use Random (introduced in lectures) or StringBuilder classes (introduced in Chapter 9) but not Timer class (because it is defined in javax.swing package which is not introduced in the course.)
- **PROJECT GRADING** - This project is worth of 100 total points determined by the following three parts together:
  1. **70 points** graded by the instructor or TA based on the submitted files.
  2. **10 points** graded by peer evaluation, i.e., you decide how much your partner deserves between 0 and 10.
  3. **20 points** graded by competition describe next.
- **PROJECT COMPETITION** - In the class of July 15, Monday, all 24 teams will be randomly divided into 4 groups, 6 teams per group, for two rounds of game tournament. Winning or losing a game in the tournament is judged by the total count of guesses in the game.
  - Every team begins with 10 free points before the tournament, if the team's program can execute in all three modes. If not, the team automatically loses 2 points to each team in the group and has zero points to be added in the total grade.
  - Any team absent from the competition will lose all 20 points from the third part of project grade.
  - In the first tournament, every team will play against all other 5 teams and wins/loses 2 points, depending on each game's result. A draw game requires both teams to play at least one more round until a real winner is announced. If a team wins all five games, it results 10 + 5*2 points or 20 points to be added to its project grade.
  - At the end of first tournament, the team with the highest points on hands is eligible for the 2nd tournament for *extra points*. Ideally, there will be 4 teams participate in the 2nd tournament. No teams in this tournament loses points. Instead, the team of the first place earns 10 extra points, the team wins the second place earns 5 extra points, and the team in the third place earns 3 extra points and the last one earns no extra points.
  - Every team will have a paper log to record each game played with another team. The record should include the competing team's ID or number, the game result (i.e., win/loss and total count of guesses

of both teams) and signature of a representative of both teams. All teams are required to turn in their paper log for the third part of project grade.