Mastermind

Team 2

Lyndsay Barnes (ljbarnes), Casey Beise (cmbeise) and Will Greene (wgreene)

Introduction- The Original Wordle

This game titled, “The Original Wordle” is designed to mimic the game Mastermind implemented by the programming language Java. The goal of the game is to guess a randomized, hidden sequence of four colors in the correct order while gaining the least amount of points. This is a two player game in which player’s take turns trying to guess a randomized sequence of colors in the least amount of guesses possible. The maximum number of guesses per sequence is 8 guesses. A player gains one point for each guess and receives two extra points if the sequence is not guessed within the maximum number of guesses allowed. Feedback regarding the guess success is provided back (in the form of white and black key pegs) to the player after each guess. White key pegs will represent a color that has been guessed correctly but placed in the wrong position of the sequence. Black key pegs will be displayed for each color guessed correctly and in the correct position of the sequence. When a player receives four black key pegs as feedback the player has guessed the hidden sequence of four colors correctly. The game is won by the player who has the least amount of points at the end of the last round. Please note that the number of rounds is determined by the players at the start of the game.

Software Requirements

Mastermind will be executed as follows (error handling is at the bottom of the Software Requirements) :

```
java TheOriginalWordle randomSeed

        Welcome to Mastermind!
A game of logic, with a bit of chance.
```

Players will enter their names (p1Name & p2Name), and number of rounds (numRounds).

```
This game is for 2 players.
Player 1 name? p1Name
Player 2 name? p2Name

Number of rounds? numRounds

Great! 4-color codes have been initialized for both players. Let's begin.

Note : guess format must be 4 characters,
       no spaces included and no duplicates e.g. BOYG
```

The round, guess number, total points and player name will be shown for each player. Player 1 will guess their secret code first for each round. The player will enter their four color guess (p1Guess). Below is an example for Round 1 for Player 1.

Player 1: Round 1, Guess 1

```
[Round 1, Guess 1] Total Points for p1Name : 0
p1Name, please enter guess (R-ed, G-reen, B-lue, Y-ellow, O-range, P-urple) : p1Guess
```

Example:

```
[Round 1, Guess 1] Total Points for p1Name : 0
p1Name, please enter guess (R-ed, G-reen, B-lue, Y-ellow, O-range, P-urple) : RGBY
```

After the initial guess is made, guesses will be displayed as below before the player is prompted to make another guess. The first four columns will display each guess. Feedback on the guess is given with the colors black and white in the last four columns. Players will receive 1 point for each incorrect guess. Below is an example of feedback of an incorrect first guess with a prompt for guess 2 and an example of a correct first guess.

Player 1: Round 1, Guess 2

p1Name Guess Log

R	G	B	Y	w	b		

[Round 1, Guess 2] Total Points for p1Name : 1
p1Name, please enter guess (R-ed, G-reen, B-lue, Y-ellow, O-range, P-urple) : **p1Guess**

If the player guesses correctly :

R	G	B	Y	b	b	b	b

Congratulations, pXName! You have guessed the code!
pXName will continue guessing.

If the player finishes guess 8 without guessing correctly, output for each player :

pXName, you did not guess the code. 2 points will be added to your Total Points.

After Player 1 has finished their turn, Player 2 will be prompted to guess their code in the same format as the examples for Player 1 above.

At the end of each round, output :

Round X Summary:
p1Name Total Points : totalPoints
p2Name Total Points: totalPoints

If applicable, start a new round. Output :

New codes have been initialized for both players.
Let's begin [Round **numRounds++**]

[Round 2, Guess 1] p1Name … etc.

At the end of the game, after the round summary, output :

Congratulations, pXName! You are the winner!

In case of a tie - instead of the above, output :

Oh no! It's a tie! Try playing again to settle the tiebreak.

Finally, output :

Play again (Y-es, Q-uit)?

User can select to play again or quit the program.

<u>Error Handling</u>

For player name input, any name longer than 50 characters, output :
"Name too long - max 50 characters"
"Player X name?"

For guess input, anything other than R, r, B, b, Y, y, G, g, O, o, P, p output :
"Invalid input : use characters R, B, Y, G, O, P only."
"pXName, please enter guess (R-ed, G-reen, B-lue, Y-ellow, O-range, P-urple) :"

For number of round input, anything not between 1 and 10, inclusive, output
"Invalid number of rounds. Please enter an integer between 1 and 10, inclusive."
"Number of rounds?"

For “play again” input, anything other than Y or Q, output :

 “Invalid input.”
 “Play again (Y-es, Q-uit)?”

For a guess input that is not equal to four characters:

 “Invalid input, guess format must be 4 characters, no spaces included e.g. YOBG”
 “pXName, please enter guess (R-ed, G-reen, B-lue, Y-ellow, O-range, P-urple) :”

For guess input that contains any color duplicates (two or more of the same color):

 “Invalid input, guess format must be 4 characters, no spaces included e.g. YOBG”
 “pXName, please enter guess (R-ed, G-reen, B-lue, Y-ellow, O-range, P-urple) :”

Software Design

Remember: You still will not be writing code at this point in the process.

<<Complete this section of the formal documentation by planning the classes, methods, and fields that will used in the software. Put the details of the userInterface design choice in this section.

Provide information about each class used in your project and how the classes are related.

- For each class that is **NOT** used to create objects (SEE Project 4 for an example), list:
  - STATIC METHOD HEADERS with javadoc to describe function, inputs, outputs
- For each class that **IS** used to create objects (SEE Project 5 Implementation Section for an example), list:
  - INSTANCE VARIABLES
  - CONSTRUCTOR()
  - INSTANCE METHOD HEADERS with javadoc to describe function, inputs, outputs

>>

```
TheOriginalWordle (Game class)
+RANDOM_GAME: int
+MIN_ROUNDS: int
+MAX_ROUNDS: int
+MAX_NAME_CHARACTERS: int
-----------------------------------------------
Public static void  main(String[] args) {
}

/**
 * This method will print the display message.
 * This method will also print the directions on how the game is played
 * and the rules of the game.
 *
 */
public static void displayWelcome() {
}
```

```
Player Class
+MAX_NUM_OF_GUESSES: int
-totalPoints: int
-playerName:String
-numGuess: int

-----------------------------
/**
 *  Player constructor initializes board of guesses array
 *  @param playerName The name of the player
 */
Player(String playerName) {
}
```

```
/**
 *This is a getter method for playerName
 *@return playerName
 */
public String getName() {
}


/**
 *This is a getter method for totalPoints
 *@return totalPoints
 */
public int getTotalPoints() {
}

/**
 *Adds points to totalPoints for every guess made by player
 */
public int addPoints() {
}

/**
 *getter method for numGuess
 *@return numGuess
 */
public int getGuess() {
}


PassCode
+CODE_LENGTH: int
+NUM_OF_COLORS: int
+BLACK_PEG:char
+WHITE_PEG.char
+RED: char
+GREEN: char
+BLUE: char
+YELLOW: char
+ORANGE: char
+PURPLE: char
-passcode: String
-feedback: String
--------------------------------------------
/**
 * PassCode constructor - creates new secret code
 */
public PassCode() {
}
```

```
/**
 *Compares guess with passcode
 *@param guess player guess of secret code
 *@return true if code is equal to guess
 *              false if code is not equal to guess
 */
public boolean compareCode(String guess) {
}

/**
 * This method evaluates the player's guess to determine if
 * there are any colors in the correct sequence compared to the passcode.  If found, the number of black pegs
 * will be incremented.
 *
 *@return numBlack the number of black pegs needed in the feedback String.
 */
public int numCorrectColorAndSlot() {
}

/**
 * This method evaluates the player's guess to determine
 * if there are any colors correctly guessed compared to the passcode.
 * If found, the number of white pegs will be incremented.
 *
 *@return numWhite the number of white pegs needed in the feedback String.
 */
public int numCorrectColor() {
}

/**
 *Returns guess feedback pegs (black and white pegs)
 *
 *@return feedBack code
public String feedback() {
}

/**
 * This method will return the String 2D array that continually
 * updates and stores each guess with its appropriate feedback per a single round.
 *
 *@param guess players attempted String guess of the hidden code
 *@param numGuess the guess number associated to the player's guess that is first passed in
 *@return String 2D array of guesses and their appropriate feedback
 */
public String[][] giveFeedback(String guess, int numGuess) {
}
```

Implementation

<< What programming concepts from the course will you need to implement your design that may require explanation? Briefly explain how each will be used during implementation. Make sure to explain any java class methods you may have used that we did not cover in class (for example: the Scanner class useDelimiter() method), specific classes from the javax.swing package used in a GUI implementation, any exceptions thrown, specific output formats, etc.
See the implementation section in our projects 1-4 to get ideas on what to add to this section.

>>

TheOriginalWordle.java Implementation:

TheOriginalWordle class represents the actual game play of the Mastermind Game.

Public Class Constants:
Below are the class constants that will be declared and initialized in this class.

- RANDOM_GAME: An integer value set to -1 representing the use of no seed.
- MIN_ROUNDS: An integer that is set to 1 that represents the minimum number of rounds a player can choose to play per game.
- MAX_ROUNDS: An integer value that is set to 10 that represents the minimum number of rounds a player can choose to play per game.
- MAX_NAME_CHARACTERS: An integer value that is set to 50 that represents the maximum number of characters allowed for a player's name.

Methods:
Below are the methods that will be utilized within this class.

- public static void main(String args[]): The main method will print information to the console for the user.
  - The main method will prompt the players to input their names and the number of rounds they wish to play within the appropriate parameters of rounds.
  - Error handling will be needed here for the player names fitting into the MAX_NAME_CHARACTERS limit. The player will be reprompted to input their name if an error occurs.
  - Error handling will also be needed for checking a valid input of the number of rounds. The player will be reprompted if the number of rounds inputted is found invalid.
  - This method will initialize new players by calling the Player Class constructor.
    - Example: Player p1 = new Player(name)
  - When all the rounds of the game have been completed, it will state which player won the game and then the players will be prompted if they want to play the game again or quit the game.
    - Example: Congratulations, SpongeBob! You are the winner!

Play again (Y-es, Q-uit)?

- public static void displayWelcome(): This method will print the display message. This method will also print the directions on how the game is played and the rules of the game.

Player.java Implementation:

The player class represents the current players information (Ie: player name, total points, current guess number, etc.)

Public Class Constants:
Below are the class constants that will be declared and initialized in this class.

- MAX_NUM_OF_GUESSES: An integer value set to 8 representing how many times a single player can attempt to guess the hidden passcode.

Private Instance Fields:
Below are the private fields that will be declared in this class.

- totalPoints: An integer that holds the current player's total number of points.
- playerName: A String that holds the current player's name.
- numGuess: An integer that holds the current player's passcode guess attempt number.

Methods:
Below are the methods that will be utilized within this class.

- Player(String playerName): This method will take in a String to store the player's name, and creates a new Passcode object to store.
- public String getName(): This is a getter method that returns the String of the player's name.
- public int getTotalPoints(): This is a getter method that returns the integer value of the player's total points.
- public int addPoints(): This method will increment the player's points after each guess. It should be noted if the player does not guess the correct passcode after guess attempt eight, then two extra points will be added to the player's total points, making a total of 10 points added during that round.
- public int getGuess(): This method will return the player's current guess attempt. It should be noted that the maximum number of guess attempts is 8.

Passcode.java Implementation:

The passcode class generates the random hidden passcode for each game round and for each player.

Public Class Constants:
Below are the class constants that will be declared and initialized in this class.

- CODE_LENGTH: An integer value set to 4 representing the hidden passcode length as well as the guess code length.
- NUM_OF_COLORS: An integer value set to 6 that represents the total number of color options that are utilized when randomly generating the four-colored hidden passcode.

- BLACK_PEG: A char set to ‘b’ that provides feedback to the player that a color that has been guessed correctly as well as guessed in the correct sequence of the hidden passcode.
- WHITE_PEG: A char set to ‘w’ that provides feedback to the player that a color has been guessed correctly but was not located in the correct sequence of the hidden passcode.
- RED: A char set to ‘R’ to represent the color red.
- GREEN: A char set to ‘G’ to represent the color green.
- BLUE: A char set to ‘B’ to represent the color blue.
- YELLOW: A char set to ‘Y’ to represent the color yellow.
- ORANGE: A char set to ‘O’ to represent the color orange.
- PURPLE: A char set to ‘P’ to represent the color purple.

Private Instance Fields:
Below are the private fields that will be declared in this class.

- passcode: A String that holds the randomly generated four colored passcode for the current player.
- feedback: A String that holds the feedback for the players guessed passcode.

Methods:
Below are the methods that will be utilized within this class.

- public PassCode(): This is the Passcode constructor that creates the randomized hidden passcode, and it stores it into the passcode variable.
- public boolean compareCode(String guess): This method takes in the player’s guess. This method returns true if the player’s attempted guess matches the hidden passcode with the correct colors in the correct order of sequence. Otherwise, returns false.
- public int numCorrectColorAndSlot(): This method evaluates the player’s guess to determine if there are any colors in the correct sequence compared to the passcode. If found, the number of black pegs will be incremented. This method returns the number of black pegs needed in the feedback String.
- public int numCorrectColor(): This method evaluates the player’s guess to determine if there are any colors correctly guessed compared to the passcode. If found, the number of white pegs will be incremented. This method returns the number of white pegs needed in the feedback String.
- public String feedback(): This method returns the feedback String for the player's current guess.
- public String[][] giveFeedback(String guess, int guess number): This method will return the String 2D array that continually updates and stores each guess with its appropriate feedback per a single round. This method will be utilized to create the visual game board.

Testing

- System Testing
  The System testing of the <<PROGRAM NAME>> program should be performed according to the SystemTestPlan_CE.pdf document. List any test files required to run these tests. Discuss any special testing scenario requirements (for example, test arrays with prespecified values to be used instead of random numbers while in test mode, command line arguments to run in test mode, etc).
- Unit Testing - no unit test files or documentation are required this semester, so this can be omitted.

Team Reflection

- Challenges faced/Lessons learned
  - What did the team learn during the project development process?
  - What problems arose and how did you solve them?

In addition to above team reflection that must be included in this document each team member must also complete an individual reflection (not in this document).

The following information should be included in each individual's peer review form of themselves.

- What did you like/dislike about the exercise?
- Any suggested changes/improvements?
- Add any comments/thoughts you care to share.