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

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

```
java TheOriginalWordle

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

Display of instructions and rules.
```

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! 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 4-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, Y-ellow, B-lue, G-reen, O-range, P-urple) : p1Guess
```

Example:

```
[Round 1, Guess 1] Total Points for SpongeBob : 0
SpongeBob, please enter guess (R-ed, Y-ellow, B-lue, G-reen, 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 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

R	G	B	Y	w	b		

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

If the player guesses correctly :

R	G	B	Y	b	b	b	b

Congratulations, p1Name! You have guessed the code!

If the player does not guess correctly within the 8 guesses:

pXName, you did not guess the passcode in 8 guesses.

2 points will be added to the 8-point total for not guessing correctly.

After Player 1 has finished guessing their code, Player 2 is prompted to guess their code in the same format as Player 1:

```
p2Name, it is now your turn to play.

[Round 1, Guess 1] Total Points for p2Name : 0
p2Name, please enter guess (R-ed, Y-ellow, B-lue, G-reen, O-range, P-urple) : p2Guess
```

[game continues as demonstrated for Player 1]

At the end of each round, the total points for each player is displayed :

```
Totals:
p1Name: totalPoints
p2Name: totalPoints
```

If applicable, start a new round. Output :

```
New codes have been initialized for both players.
Let’s begin [Round 2]

[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.

Error Handling

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, B, Y, G, O, P output :

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

“[Round #, Guess #] Total Points for pXName : #

pXName, please enter guess (R-ed, Y-ellow, B-lue, G-reen, 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 / y or Q / q, output :

```
"Invalid input."
"Play again (Y-es, Q-uit)?"
```

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

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


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

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

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


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

Software Design

TheOriginalWordle.java:

TheOriginalWordle class represents the actual game play of the Mastermind Game. The user interface will be within the terminal window.

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 maximum 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 carries out the actual game play for each player.
    - 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 player1 = new Player(name)
    - When all the rounds of the game have been completed, it will state which player won the game or if there is a tie 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.

TheOriginalWordle (Game class)
<u>+RANDOM_GAME: int</u>
<u>+MIN_ROUNDS: int</u>
<u>+MAX_ROUNDS: int</u>
+<u>MAX_NAME_CHARACTERS: int</u>

```
/**
 *Starts The Original Wordle game
 * @param args args[0] optional seed used for testing which determines
 * the passcode for players to guess
 */
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.java:

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

<u>Public Class Constants:</u>
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.
- MAX_ROWS_COLUMNS: An integer value set to 8 representing the maximum number of rows and columns for the player's game board.
- CODE_LENGTH: An integer value set to 4 representing the hidden passcode length as well as the guess code length.

<u>Private Instance Fields:</u>
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.
- gameBoard: A String 2D array that holds the current gameboard with guesses and feedback.
- ga: An object of the GuessAnalysis class.
- seed: An integer that holds the seed.

<u>Methods:</u>
Below are the methods that will be utilized within this class.

- Player(String playerName, int seed): This method will take in a String to store the player's name, will take in the imputed seed integer, and creates a new GuessAnalysis object.
- public void newRound(): This method creates a new GuessAnalysis object which sets a new passcode for each round. This method also resets the gameBoard and resets the guess number for each round.

- public boolean getCompareCode(String guess):
- public int getNumCorrectColorAndSlot(String guess):
- public int getNumCorrectColor(String guess):
- 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 void resetPoints(): This method resets the total points to 0.
- public void addGuess(): This method increases the number of guesses by 1 for every guess a player makes.
- public String[][] updateBoard(String guess, int numGuess): 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.
- public String getBoard(): This method will return a String representation of the game board for the player.

Player Class
+MAX_NUM_OF_GUESSES: int
+MAX_ROWS_COLUMNS: int
+CODE_LENGTH: int
-totalPoints: int
-numGuess: int
-playerName:String
-gameBoard: String [][]
-ga: GuessAnalysis
-seed: int

```
----------------------------
/**
 *  Player constructor initializes board of guesses array
 *
 *  @param playerName The name of the player
 *  @param seed number seed for passcode generation for testing
 */
player(String playerName, int seed) {
}


/**
 * Creates new GuessAnalysis object, and resets numGuess and gameBoard for each round
 */
public void newRound() {
}
```

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

/**
 *Gets number of pegs in guess that are the same color and slot as the passcode.
 *
 * @param guess player guess of secret code
 * @return number of pegs in guess that are the same color and slot as the passcode.
 */
public int getNumCorrectColorAndSlot(String guess) {
}

/**
 * Gets number of pegs in guess that are the same color as the passcode
 *
 * @param guess player guess of secret code
 * @return number of pegs in guess that are the same color as the passcode
 */
public int getNumCorrectColor(String guess) {
}

/**
 *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
 *@param totalPoints
 */
public void addPoints(int totalPoints) {
}
```

```
/**
 * Resets players totalPoints to zero for each new game
 */
public void resetPoints() {
}

/**
 * Increases numGuess by 1 for every guess player makes
 *
 */
public void addGuess() {
}

/**
     * 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 gameBoard String 2D array of guesses and their appropriate feedback
     */
    public String [][] updateBoard(String guess, int numGuess) {
}

/**
 *Creates a string representation of the gameBoard
 * @return s string representation of the gameBoard
 */
 public String getBoard() {
}
```

GuessAnalysis.java:

The GuessAnalysis class generates the random hidden passcode for each game round and for each player. Then compares the hidden passcode to the current player's guess of the passcode.

<u>Public Class Constants:</u>
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.
- COLORS: A char array of all the color options for the code.

<u>Private Instance Fields:</u>
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.

<u>Methods:</u>
Below are the methods that will be utilized within this class.

- public GuessAnalysis(int seed): This is the GuessAnalysis constructor that takes in a seed, 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, it returns false.
- public int numCorrectColorAndSlot(String guess): This method takes in the player’s guess and 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(String guess): This method takes in the player’s guess and 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(String guess): This method takes in the player’s guess and returns the feedback String for the player's current guess.

GuessAnalysis
<u>+CODE_LENGTH: int</u>
<u>+NUM_OF_COLORS: int</u>
<u>+BLACK_PEG:char</u>
<u>+WHITE_PEG.char</u>
<u>+RED: char</u>
<u>+GREEN: char</u>
<u>+BLUE: char</u>
<u>+YELLOW: char</u>
<u>+ORANGE: char</u>
<u>+PURPLE: char</u>
<u>+COLORS: char[]</u>
-passcode: String

```
/**
 * GuessAnalysis constructor - creates new secret code
 *
 * @param seed -  number seed for passcode generation for testing
 */
public GuessAnalysis(int seed) {
}

/**
 *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.
 *
 * @param guess player guess of secret code
 * @return numBlack the number of black pegs needed in the feedback String.
 */
public int numCorrectColorAndSlot(String guess) {
}

/**
 * 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.
 *
 * @param guess player guess of secret code
 * @return numWhite the number of white pegs needed in the feedback String.
 */
public int numCorrectColor(String guess) {
}

 /**
  * Returns guess feedback pegs (black and white pegs)
  *
  * @param guess player guess of secret code
  * @return result will return the String feedback code
  */
 public String feedback(String guess) {
}
```

Implementation

- You will need to utilize class constants therefore there should not be any magic numbers. The exceptions to the magic numbers are 0 and 1. You will also have class constants to represent the colors of the game. See the Design section above for more specific information.
- Nested 'for' loops will be helpful to create guess opportunities within rounds
- A concept from Lab8A (NestedLoops.java) will be helpful when designing a passcode with no duplicates i.e. for (int j = 0; j < i; j++) …
- **HINT:** You will need to create a 2D array to create the visual game board of the game. This array will hold both the player's guesses and the feedback to each guess. You may want to consider having two separate 'for loops' when creating the array. One 'for loop' for the players guess string to be added into the gameboard (looping through columns 0-3) and one 'for loop' for adding the feedback string into the gameboard (looping through columns 4-7). ***Remember:*** if the first guess of the game board is going to be displayed at the 'bottom' of the game board with the subsequent guess and feedback stacking above the first guess/feedback, then the first guess/feedback will be placed in the seventh indexed row of the array. See chart for visual example:

0,0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1							
2							
3							
4							
5							
6 *Second*	*Guess*	*Shows*	*Here*	b	b		
7 *First*	*Guess*	*Shows*	*Here*	b	b	w	

- You will need to create blank spaces in the appropriate columns for the String feedback in the GuessAnalysis class when there are wrong colors guessed. See code implementation below for the blank space. Please note, you will need to create a constant for the number 4 and you will need to update this method to add the black and white pegs to the String feedback prior to the blank spaces being added.
  public String feedback(String guess) {

```
○ if (result.length < 4) {
○     for (int i = 0; i < (4 - result.length); i++) {
○           resultOne += " ";
○      }
○     result = (result + resultOne);
○ }
○         return result;
○ }
```

- Error Handling: Please note, that each error below will require reprompting of the user for valid input until the user provides that valid input. You will need to implement error handling for the following:
  - For player name input, any name longer than 50 characters.

- For the number of rounds input, anything not between 1 and 10, inclusive.
- For “play again” input, anything other than Y or Q.
- For a guess input that is not equal to four characters.
- For guess input, anything other than R, B, Y, G, O, P.
- For guess input that contains any color duplicates (two or more of the same color):

Testing

See SystemTestPlan_CE Documentation

Team Reflection

This project gave our team insight into real-world applications of software development. Off the bat, we learned the importance of communication with the customer, as this is the driver behind all other parts of the development process. Interpretation of customer instructions can vary so it was necessary to follow up with initial questions to nail down the expectations. For example - during our initial design, players alternated taking turns solving their passcodes, which is not what the customer wanted. We then changed our design to have a player complete their guesses before moving to the next player.

Documentation details was another important thing we learned about. Although we started the course with trial-and-error coding, this exercise required us to look at the issue from a bird's eye view, brainstorming classes, methods and constants *before* writing any code. This also helped the team stay on the same page, as we delegated duties.

However, even as we planned accordingly, this documentation was a WIP from start to finish. We realized the importance of balance between coding and documentation, as each feeds off the other. Initially, it took some time to visualize the class, method and constant implementation but we found that balance as we made progress on the exercise.