Homework 5: Pick Your Letters

HW deadline as per Canvas.

This homework deals with the following topics:

- Arrays
- Strings
- Class-based object-oriented programming

In this HW, we will be implementing the game *Pick Your Letters*, which is a game that involves re-arranging a group of cards in order to make up a word.

About the Game

This game needs two players and we will just play the user versus the computer. The user's moves are decided by the user playing the game, by asking for input, and the computer's moves are decided by the program.

There is NO right answer for the section that asks you to program a strategy for the computer. All we want you to do is come up with a reasonable enough strategy that ensures a human user does not always beat the computer. So, unlike your previous assignments, this one has a creative component to it.

You must use Java arrays for this assignment. (Do not use other data structures such as ArrayLists.)

At the very beginning, the program will prompt the user to enter a number as the Length *L* to be the length of the word they are going to make up. The game starts with a main pile of *n* cards, each labeled with a letter, and $n = 26 * L$. So, there are *L* of each letter. The objective it to be the first player to arrange *L* cards in your hand in an order that forms a word.

The cards in the main pile are shuffled and both the user and the computer are each initially dealt *L* cards from the main pile. As a player receives each card, they must place the card at the far right of their hand in the order they received it. e.g., if a user receives "A", "R", "K", "I", "P" consecutively, their hand is in the same order and looks like: "A", "R", "K", "I", "P".

After the hand is dealt to the user and the computer, there will be $((26 * L) - (2 * L))$ cards left in the main pile. The top card of the main pile (the card to the far left of cards list) is turned over to begin the discarded card pile.

On each player's turn, the player chooses to either pick up the top card from the discard pile, or to pick up the top card from the main pile. The top card from the

discard pile is known. In other words, the discard pile is 'face up' and everyone knows what the letter on it is. The main pile is 'face down' and the player does not know what the card is.

Once a player chooses a card, either from the discard pile or from the main pile, the player decides where in the hand to put the card. The hand is always of the size *L*, so placing a card means that an existing card in the hand is removed and replaced with the new card.

If the player takes a card from the main pile (the one that is 'face down'), the player can reject it and place it in the discard pile. This means that nothing in that player's hand changes during that turn. If the player takes a card from the discard pile (the one that is 'face up'), the player MUST place it into their hand.

The first player whose cards in hand form a word wins. How does the program know the cards form a word? We'll discuss that below, soon!

If, at any point, all of the cards have been removed from the main pile, then all of the cards in the discard pile are shuffled and moved to the main pile. Then the top card is turned over to start the new discard pile.

The Actual Program

We are providing you with a "skeleton" of the game (PickYourLetters.zip) -- the classes and methods have been defined, but just about all of the actual code has been deleted. Your task is to finish the program by adding the necessary code. Javadoc-style comments (included) tell you what must be done in each method.

Below are the classes in the project. Note, some of the class files declare a package (at the top) because those files sit in subdirectories. **Do not remove any of the package declarations in the files or change the structure of the program.**

- **The *PickYourLetters* Class** controls the entire game. It has the *main* method which serves as the entry point to the entire program and it will be the file you will "run".
- **The *WordReader* Class** reads all the words from a file and loads them into an array of strings
- **The *WordPocessor* Class** filters all the words in the array based on a given length
- **The *Cards* Class** creates and manages the main pile of cards and the discard pile of cards

- **The *Human* Class** represents a human player in the game
- **The *Computer* Class** represents a computer player in the game

Below you will find explanations of the methods that need to be written. We are expecting to see these methods with these names and method signatures exactly. Do not change the names of these methods, as we will be running automated tests against each individual method.

You will also have to write some methods by yourself, in addition to the ones listed below. Feel free to name your helper methods whatever you want, as long as they happen to reflect what the method does.

Note that we will make heavy use of arrays in this assignment. Since hand cards look like a horizontally oriented structure, we need to have some convention. Our convention is that the hand in the picture below is going to be represented as {“K”, “K”, “A”, “A”, “A”}.

If *L* = 6, and the player is dealt the last card “S”, the “S” is inserted to the right and the array looks like {“K”, “K”, “A”, “A”, “A”, “S”}. If you want to replace “S” with another card, reference index 5, which means the card with index 5 in the array.

The main pile and discard pile are also going to be represented as arrays. However, cards in piles looks more like a vertically oriented structure. Our convention is that the card on the top of a pile is always to the left of the array.

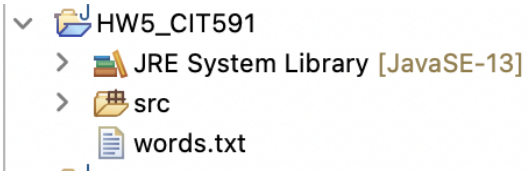
Note that in both the main pile and the discard pile, you should only have access to the top. So, to add/remove a card, consider the top of the pile to be the beginning of the array. **Note: You will have to resize the arrays in order to add/remove cards**.

How does the program know hand cards form a word?

We've already written a method for you in the ***WordReader*** class:

public String[] readFromFile(String fileName)

- This method reads a file with name 'words.txt', which contains all the words acceptable in the game. All the words have a length of 3 - 10, inclusive, and only contain letters (ignore case). This method returns an array containing all these words. Note that 'words.txt' should be outside the 'src' folder as shown below.



After calling this method (shown below), all words are stored in the array named *allWords*.

```
//reads all words from file
WordReader wr = new WordReader();
String[] allWords = wr.readFromFile("words.txt");
```

You can now use *allWords* directly. The first player whose hand cards forms a word that is included in *allWords*, wins.

Methods to Implement

Be sure to add Javadocs to all your methods and comments to your code.

PickYourLetters Class

public void run()

- Launches and controls the game
- Follow the comments to implement this method

public int askForLength(Scanner scanner)

- Ask the user for the number of hand cards so that the length of words players is going to guess is determined
- Prompt again if the user input is not a valid integer, or if the number is not between 3 to 10, inclusive
- Returns the number of hand cards *L*
- Parameter *scanner* is used for getting user input

public boolean checkGameOver(String[] humanHandCards, String[]

computerHandCards, String[] wordsWithSpecificLength)

- Check if the game ends
- If there is a tie, the game ends as well
- Parameter *humanHandCards* is the human's current hand (array)
- Parameter *computerHandCards* is the computer's current hand (array)
- Parameter *wordsWithSpecificLength* is an array containing all the words with the specific length
- Returns true if the human or the computer wins the game, otherwise false

***WordProcessor* Class**

public String[] filterWordList(String[] allWords, int length)

- Given an array of words, and a number, returns an array of words with the specific length
- Parameter *allWords* is the array of all words
- Parameter *length* is the given length

Cards Class

public String[] setUpMain(int length)

- You'll run this method once after the length of words is determined and the array of words is filtered
- Creates a main pile of 26 * *length* cards, represented as an array of lowercase letters (as strings), with *length* of each letter
    - For example, if you called this method with an argument of 2, it will create a main pile of cards represented as an array of lowercase letters (as strings), with each letter included twice:
      {"a", "b", "c", "d", "e", "f", "g", "h", "I", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z", "a", "b", "c", "d", "e", "f", "g", "h", "I", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z"}
- Returns the main pile
- Parameter *length* is the given length

public String[] setUpDiscard()

- Creates a discard pile of 0 cards, represented as an empty array
- Returns the discard pile

public String[] getMainPile()

- Get the main pile

public String[] getDiscardPile() {

- Get the discard pile

public void shuffleMainCards()

- This method shuffles the main pile doesn't return anything
- You are allowed to import the *Random* class and use methods like *nextInt*

public void shuffleDiscardCards()
- This method shuffles the discard pile doesn't return anything
- Again, you are allowed to import the *Random* class and use methods like *nextInt*

public void dealInitialCards(Computer computer, Human human, int length)
- Start the game by dealing two sets of *length* cards each, from the *main pile*
- Make sure that you follow the normal conventions of dealing. So, you have to deal one card to the computer, one to the user, one to the computer, one to the user, and so on.
- The computer is always the first person that gets dealt to (and always plays first)
- Call *setHandCards* for the given *computer* and for the given *human* players, to set each players' initial hand cards
- Remove the card on top of the main pile and put it on the discard pile
- This method doesn't return anything

public void addToDiscardPile(String card)
- Move the given *card* to the top of the discard pile
- Parameter *card* is the given letter to be discarded
- This method doesn't return anything

public String getFirstFromMainPileAndRemove()
- Return and remove the first item of the main pile

public String getFirstFromDiscardPileAndRemove()
- Return and remove the first item of the discard pile

public void checkCards()
- Check whether the main pile is empty
- If so, shuffles the discard pile and moves all the cards to the main pile. Then turn over the top card of the main pile to be the start of the new discard pile.
- Otherwise, do nothing. public String[] play(String[] targetArray) {

Computer Class

public void setHandCards(String[] handCards)
- Set the computer player's hand
- Parameter *handCards* are the computer player's hand

public String[] getHandCards()

- Get computer player's hand

public String[] play(String[] targetArray)

- This method is where you can write the computer's strategy. It is also the method where we are giving you very little guidance in terms of actual code.
- You are supposed to be somewhat creative here, but we do want your code to be deterministic. That is, given particular hand cards and a particular letter (either from the discarded pile or as the top of the main pile), you either always take the card or always reject it.
- Here are some potential decisions the computer had to make:
    - Given the computer's current hand, do you want to take the top card from the discard pile or do you want to take a card from the top of the main pile and see if that card is useful to you
    - How you evaluate the usefulness of a card, and the position it should go into
    - There might be some simple rules you can give to the computer. For example, you can compute how many similarities there are between each word in the computer's target list (array) of words and the letters on hand. And you can test if replacing a specific card with another card can increase the similarities.
- You are allowed to do pretty much anything in this method except make a random decision or make a decision that is obviously incorrect. For instance, selecting a fixed target word is not very smart. We want your computer to determine the target word, or target words, dynamically.
- Also, the computer CANNOT CHEAT. What does that mean? The computer cannot peek at the top of the main pile and then make a decision to go to the discard pile. Its decisions should be something that a human could be able to make as well.
- For a given letter, the computer decides whether to take it or not
- Parameter *targetArray* is an array of words. What this array looks like is up to you. It may contain words that are most similar to hand cards or something else, but it should be reasonable.
- This method doesn't return anything

Human Class

public void setHandCards(String[] handCards)

- Set the human player's hand
- Parameter *handCards* are the human player's hand

public String[] getHandCards()

- Get human player's hand

public void play(Scanner scanner)

- Control the game play for the human
- Parameter *scanner* is used for getting user input

public int askForTheLetterToBeReplaced(int length, Scanner scanner)

- Ask for the index of the letter that the user wants to replace
- Prompt again if the input index is out of range or invalid
- Parameter *length* is the number of cards in the human's hand
- This method returns the index of the letter to be replaced
- Parameter *scanner* is used for getting user input

public boolean askYesOrNo(String msg, Scanner scanner)

- Display *msg* and get user's response
- Prompt again if the input is invalid
- Parameter *msg* is the message to display
- This method returns true if the user answers "y" or "yes", and returns false if the user answers "n" or "no"
- Again, parameter *scanner* is used for getting user input

Note: Remember, you are encouraged to write additional helper methods. **In fact, we will be surprised if you don't add some of your own methods to some of the classes!**

User Interface

You're free to create your own user interface for the game, as long as it makes sense for the user playing. The computer goes first. Your program can print the computer's initial hand (in array form) at the beginning of the game. And for the rest of the game, the human player should also see what's in the computer's hand. After the computer's turn, your program should print the results of the turn. For example, your program might print this if the computer chooses the top of the discard pile or chooses the top of the main pile.

```
Welcome to the game!
Enter a number between 3 - 10 to be length of the word you are going to guess:
4
Filtering the words with given length 4. Please standby ...
-----------------------------------------------
Computer's turn.
Computer took "o" from DISCARD PILE
Computer's hand cards are: [n, o, r, f]
-----------------------------------------------
Your turn. Your hand cards are: [h, d, x, u]
Pick "m" from DISCARD PILE or reveal the letter from MAIN PILE
Reply "D" or "M" to respond:
```

When it's the human user's turn, your program should print the user's hand (in array

form), the top card of the discard pile, and, if they choose to pick from the main pile, the top card of the main pile. For example, your program might print this if the human user chooses the top card of the main pile and replaces a card in their hand.

```
------------------------------------------------
Your turn. Your hand cards are: [h, d, x, u]
Pick "m" from DISCARD PILE or reveal the letter from MAIN PILE
Reply "D" or "M" to respond:
m
The letter from MAIN PILE is "r"
Do you want to accept this letter? Type "y/yes" to accept, "n/no" to discard.
yes
Input the index of the letter to be replaced, e.g. "1":
2
You replaced "x" with "r"
Your hand cards are: [h, d, r, u]
------------------------------------------------
```

Evaluation

The primary goal of this assignment is to get you very familiar with working with arrays, strings, and object-oriented programming, and to have some level of fun while creating a game.

While we want you to spend time on coming up with some kind of strategy for the computer, that is NOT the primary part of the assignment. Any simple, but reasonable strategy will have you doing some fun things with arrays. If the human player always does absolutely nothing at all, that is, if they reject the top discarded card and they reject the top card from the main pile and move it onto the discard, then we want the computer to win. Your computer player should be smart enough to beat the 'stupid, lazy' user.

You will be evaluated on 5 things:

1. Does your game work - 5 points
   Again, please do not worry about us trying to crash your program. Just ensure that reasonable inputs will allow a user to play the game.
2. Method design – 15 points
   In addition to your game working as it should, we will confirm that the required methods above have been implemented correctly, and do exactly what they are supposed to do. We will test the methods individually by running "unit tests" against them with different inputs and by inspecting the results.

   For example, the *addToDiscardPile* method should add the given card to the top (or beginning) of the discard pile. We will call your method with a test card as the argument, then confirm that the top card of the discard pile is what it should be.

3. Strategy – 5 points
   Is your computer's strategy something that makes sense? Please comment your code for that part of the homework very thoroughly, and explain your strategy for the computer.
4. Usability – 3 points
   This is going to be a subjective evaluation. You are making a game. A user who knows the rules of the game should be able to play it without too much trouble. In particular, they should not have to read a single line of your code in order to operate the game.
5. Style – 2 points
   The usual stuff here. Style includes things like appropriate variable names, method names, clear comments, and general readability of your code. Every method must have Javadocs. All non-trivial lines of code must be commented. Style will always be part of your evaluation.

Data Structures

In this HW you are only allowed to use arrays. Do not use ArrayLists! Remember that the main pile, the discard pile, and the two hand cards (user and computer) are just arrays.

Submission

Please submit all the classes in your entire Java project. Make sure it includes everything in your "src" folder. Do not remove any package declaration in any file and keep the entire program in the same provided structure.