Homework 5

In this homework you will implement two classes: a new version of the `MyString` class, and `Scrabble`. The `MyString` class features general-purpose functions that come to play in many applications that perform string processing. The `Scrabble` class features a version of the Scrabble game. The game implementation makes extensive use of the services of `MyString` functions.

I: MyString

Read the class documentation and implement all the functions.

Implementation notes

In HW4 we also wrote a `MyString` class. In reality, you will grow and polish your `MyString` class with additional string processing functions, each time you implement a new string processing function. For the purpose of this exercise though, you have to create a new `MyString` class. Some of the logic that you will use in writing the class functions will be similar to the logic that you used in HW4 – that's fine.

We suggest implementing the `MyString` functions in the order in which they appear in the class. As usual, functions are welcome to call each other, as needed. For example, the implementation of the `subsetOf` function can make use of the `countChar` function.

Write more tests of your own in the `main` function, as needed. Make sure to test all the `MyString` functions before proceeding to Part II (Scrabble).

II: Scrabble

Altogether, there are about 600,000 words in the English language, of which about 550,000 words are rarely used. Most English speakers have a vocabulary of about 20,000 words, while people with a university degree use about 40,000 words.

In the game of Scrabble, a random set of letters, called a "hand", is dealt to a player. The player tries to construct English words from the given hand. Each valid word receives a score, as described below. Naturally, the game favors players who have a good command of the English language: the richer your vocabulary, the better is your ability to compose words from any given hand. Different versions of Scrabble (in many languages and difficulty levels) are widely used around the world, both for entertainment and for educational purposes.

A typical Scrabble game progresses according to the rules described below.

Dealing: The player is dealt a so-called "hand" of n letters, chosen at random from the English alphabet. For example, if we assume $n = 10$, an initial hand can be "aedierbcqr". The player tries to create valid words from the given hand, using each letter at most once. For example, the player can come up with the word "bird". If the word is valid, the player gets a score and the hand becomes smaller. For example, following "bird", the initial hand "aedierbcqr" becomes "aeercq". At this point the player can come up with the word "car", causing the hand to become "eeq". At this point most players will give up and stop playing the hand, since it is unlikely that "eeq" can yield additional valid words. Note though that the user could have composed "care" instead of "car", getting a better score.

Scoring: A word is said to be valid if: (a) its letters form a subset of the current hand, and (b) the word appears in a given dictionary. Each valid word is awarded a score, as follows: ‘a’ is worth 1 point, ‘b’ is worth 3 points, ‘c’ is worth 3 points, ‘d’ is worth 2 points, ‘e’ is worth 1 point, and so on. Here is the complete Scrabble letter values:

The score for a valid word is the sum of its letter scores, multiplied by the word’s length. If the first word that the player constructs uses all the letters of the hand, 50 points are added to the word score.

For example, “cab” is worth (3 + 1 + 3), multiplied by `“cab”.length()`, for a total of 21 points. As another example, suppose that the hand is “eunrvdtesa”, and that the player came up with the word “adventures”. The score of this word will be the sum of the Scrabble points of the letters, times the word’s length, plus an additional 50 point bonus for using all 10 letters in the initial hand.

The score of a hand is the sum of the scores of all the words that the player made from that hand.

Sample hand sessions (in which each initial hand consists of $n = 7$ letters)

// Example of one hand (the user’s input is colored red)

```
Current Hand: p z u t t t o
Enter a word, or '.' to finish the hand:
top
top earned 15 points. Total: 15 points

Current Hand: z u t t
Enter a word, or '.' to finish the hand:
tu
No such word in the dictionary. Try again.

Current Hand: z u t t
Enter a word, or '.' to finish the hand:
tat
Invalid choice. Try again.

Current Hand: z u t t
Enter a word, or '.' to finish the hand:
.
End of hand. Total score: 15 points.
```

// Example of another hand:

```
Current Hand: a q w f f i p
Enter a word, or '.' to finish the hand:
paw
paw earned 24 points. Total: 24 points

Current Hand: q f f i
```

```
Enter a word, or '.' to finish the hand:
if
if earned 10 points. Total: 34 points

Current Hand: q f
Enter a word, or '.' to finish the hand:
.
End of hand. Total score: 34 points.
```

// Example of another hand:

```
Current Hand: a r e t i i n
Enter a word, or '.' to finish the hand:
inertia
inertia earned 99 points. Total: 99 points

Ran out of letters. Total score: 99 points.
```

Implementation

To get started, load the given `dictionary.txt` file into the editor, and review it. The file contains about 80,000 words, and there is no need to read all of them. Just scroll up and down, to get an impression of how the data looks like. Every string in this file is considered a word in the English language.

Implementing `init`: Familiarize yourself with all the static variables in the `Scrabble` class, and complete the implementation of the `init` function. To test your work, use the editor to create a tiny dictionary file of a few words, and call it, say, `test.txt`. Set `WORDS_FILE = "test.txt"` in the class code, and compile and execute `Scrabble.java`. Write test code that prints the `DICTIONARY` array. Make sure that the array contains all the words in `test.txt`, and that all of them are in lowercase.

Implementing playHand: The logic of this function can be described as follows:

```
playHand(hand) Logic:
    get the length of the hand
    set the score of the hand to 0
    while the length of the hand is greater than 0:
        display  the hand
        display “Enter a word, or '.' to finish this hand:"
        use StdIn to get the user’s input
        if the input is “.”:
            break
        if the input is not a subset of the hand:
            display “Invalid choice. Try again”
        otherwise: if the input is not in the dictionary:
                        display "No such word in the dictionary. Try again”
                    otherwise (the input is a valid word):
                        compute and add the score of the input to the score of the hand
                        display the score of the input and the score of the hand
                        update the hand
        display an empty line
    if there are no more letters in the hand:
        display "Ran out of letters” and the score of the hand
    otherwise: display "End of hand” and the score of the hand
```

The statement "`break`" causes Java to jump out of the loop in which it is defined. In the case of the logic above, it will cause Java to jump to the first statement following the while loop.

Inspect the sample hand sessions and the `playHand` function logic, and make sure that you understand how the latter realizes the former.

Implement the `playHand` function, and test it using the `testPlayHand` function (add more tests of your own). The output that your `playHand` function generates must be formatted exactly as the examples given in the sample hand sessions.

If your `playHand` implementation works properly then – zamelvot! – you are almost done. We now turn to describe the "game envelope", starting with a sample game session.

Sample game session

(The player's inputs are colored red)

```
Loading word list from file...
83667 words loaded.
Enter n to deal a new hand, r to replay the last hand, or e to end game:
n

Current Hand: p z u t t t o
Enter a word, or '.' to finish the hand:
tot
tot earned 9 points. Total: 9 points

Current Hand: p z u t
Enter a word, or '.' to finish the hand:
.
End of hand. Total score: 9 points.

Enter n to deal a new hand, r to replay the last hand, or e to end game:
r
Current Hand: p z u t t t o
Enter a word, or '.' to finish the hand:
top
top earned 15 points. Total: 15 points

Current Hand: z u t t
Enter a word, or '.' to finish the hand:
tu
No such word in the dictionary. Try again.

Current Hand: z u t t
Enter a word, or '.' to finish the hand:
tat
Invalid word. Try again.

Current Hand: z u t t
Enter a word, or '.' to finish the hand:
.
End of hand. Total score: 15 points.

Enter n to deal a new hand, r to replay the last hand, or e to end game:
n
```

```
Current Hand: a q w f f i p
Enter a word, or '.' to finish the hand:
paw
paw earned 24 points. Total: 24 points

Current Hand: q f f i
Enter a word, or '.' to finish the hand:
qi
qi earned 22 points. Total: 46 points

Current Hand: f f
Enter a word, or '.' to finish the hand:
.
End of hand. Total score: 46 points.

Enter n to deal a new hand, r to replay the last hand, or e to end game:
n
Current Hand: a r e t i i n
Enter a word, or '.' to finish the hand:
inertia
inertia earned 99 points. Total: 99 points

Ran out of letters. Total score: 99 points.

Enter n to deal a new hand, r to replay the last hand, or e to end game:
e
```

As the game session shown above illustrates, most of the game action is realized by the `playHand` function. The rest of the game session is a simple "envelope", which is realized (in our implementation) using a function named `playGame`. Here is another example:

Sample game session (focusing only on the action realized by the `playGame` function):

```
Loading word list from file...
83667 words loaded.
Enter n to deal a new hand, r to replay the last hand, or e to end game:
r
You have not played a hand yet. Please play a new hand first!

Enter n to deal a new hand, r to replay the last hand, or e to end game:
n
<calls playHand>

Enter n to deal a new hand, r to replay the last hand, or e to end game:
r
<calls playHand>

Enter n to deal a new hand, r to replay the last hand, or e to end game:
n
<calls playHand>

Enter n to deal a new hand, r to replay the last hand, or e to end game:
x
Invalid command.
Enter n to deal a new hand, r to replay the last hand, or e to end game:
e
```

(program exits)

Implementing `playGame`: The game playing logic shown above can be implemented by an infinite loop. In each iteration, the player is asked to enter `n`, `r`, or `e`, and the function responds by either calling `playHand` or by printing some message.

Implement the `playGame` function, and test it by playing several games and trying examples of all the possible user inputs.

Note: The output that your `playGame` function generates must be formatted exactly as the examples given in the sample game sessions.

III: Extension

This section describes an optional extension that you don't have to submit, but you may find cool to implement.

In the basic version of the game, hands are constructed by drawing letters from the English alphabet with equal probabilities.

Consider a version of the game in which vowels are selected with greater probabilities than consonants. This makes the game easier: the more vowels in a hand, the easier it is to construct words. This version could play well in educational applications of the game.

Implement a version of the game in which the hands that the player gets contain letters from the subset "`aeio`" more frequently than the other letters in the alphabet.