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 support many applications that need string processing services. The `Scrabble` class features a simple version of the Scrabble game. The game implementation makes extensive use of the `MyString` functions.

MyString

Read the class documentation and implement all the functions.

Implementation notes

In HW4 we also wrote a `MyString` class. For the purpose of this exercise, we'll create a new `MyString` class. 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.

You can use any possible `String` function, including `subString`, which can be quite useful in the implementation of the `insertRandomly` function.

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

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.

Our version of the 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 $n = 10$, 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 include a 50 points bonus for using all 10 letters in the initial hand. Finally, in our version of the game, if the word contains the letters ‘r’, ‘u’, ‘n’, and ‘i’, an additional bonus of 1000 points is given.

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

Sample hand session (the user’s input is colored red)

```
Current Hand: k a o g t o z b q c
Enter a word, or '.' to finish playing this hand:
zoo
zoo earned 36 points. Score: 36 points

Current Hand: k a g t b q c
Enter a word, or '.' to finish playing this hand:
tak
No such word in the dictionary. Try again.

Current Hand: k a g t b q c
Enter a word, or '.' to finish playing this hand:
at
at earned 4 points. Score: 40 points

Current Hand: k g b q c
Enter a word, or '.' to finish playing this hand:
.
```

The Dictionary

Our Scrabble program uses a dictionary stored in a file named `dictionary.txt`. The file contains about 80,000 words, and there is no need to read all of them. Start by loading the file into the editor, and get an impression of how the data is formatted (a long sequence of uppercase strings). It is surprising that every string in this file is a valid word in the English language, including strange things like AAHING and AASVOGEL.

We also provide a smaller subset of this file, named `TinyDictionary.txt`. You may want to use it in some of your tests.

Reading inputs from files, and from the terminal

Our Scrabble application has two "reading challenges". We have to read words from a given file, and we have to read the user's inputs from the terminal. We will perform both tasks by using the services of a class named `In`. The `In` class is a complex, object-oriented class, and there is no need to read or understand its code. Instead, we'll focus on how to *use* it, as a black box abstraction. This is illustrated in the `Demo.java` class, which is included in this homework materials.

The `Demo` class builds a dictionary, and then gets into a loop that prompts the user to enter how many random words s/he wants to see from the dictionary. It's not a terribly exciting game, but it illustrates some important programming practices. So, to get started, compile the classes `In.java` and `Demo.java`. Then run `Demo`, and play the game. Finally, read the `Demo` class documentation carefully, and make sure that you understand. This understanding will pave the way to a painless implementation of the `Scarbble` class, which comes next.

Sample Scrabble game session (the user's inputs are colored red)

```
% java Scrabble
Loading word list from file...
83667 words loaded.
Enter n to deal a new hand, or e to end the game:
n
Current Hand: u c m t a u e e c k
Enter a word, or '.' to finish playing this hand:
mate
mate earned 24 points. Score: 24 points

Current Hand: u c u e c k
Enter a word, or '.' to finish playing this hand:
cue
cue earned 15 points. Score: 39 points

Current Hand: u c k
Enter a word, or '.' to finish playing this hand:
kuc
No such word in the dictionary. Try again.

Current Hand: u c k
Enter a word, or '.' to finish playing this hand:
.
End of hand. Total score: 39 points

Enter n to deal a new hand, or e to end the game:
n
Current Hand: e a p h d o x q l e
Enter a word, or '.' to finish playing this hand:
eagle
Invalid word. Try again.
Current Hand: e a p h d o x q l e
```

```
Enter a word, or '.' to finish playing this hand:
leap
leap earned 24 points. Score: 24 points

Current Hand: h d o x q e
Enter a word, or '.' to finish playing this hand:
hed
No such word in the dictionary. Try again.

Current Hand: h d o x q e
Enter a word, or '.' to finish playing this hand:
ho
ho earned 10 points. Score: 34 points

Current Hand: d x q e
Enter a word, or '.' to finish playing this hand:
.
End of hand. Total score: 34 points

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

Notice that the more vowels a hand has, the easier it is to construct words from it. With that in mind, here is how we create hands in this game: We construct a string consisting of 8 random letters, and then we insert, randomly, the two letters 'a' and 'e' into the string. More about this in the `Scrabble` class documentation.

Implementation Plan

0. The `MyString` class is used extensively by the `Scrabble` class. Therefore you must first fully implement the `MyString` class.

1. Read and understand the class variable declarations, and the `init` function. Run and understand the `testBuildingTheDictionary` method.

2. Implement and test the functions `wordScore` and `createHand`. Notice that we provide standard tests that you are welcome to use and extend.

3. Complete the implementation of the `playHand` function, and test it.

4. Complete the implementation of the `playGame` function, and test it.

The implementation plan just described is a good example of an important software engineering practice called *Unit-testing*. The basic idea is designing the system as a modular set of functions that can be implemented and tested separately from the rest of the system.

Extension: In the game described above, hands are constructed by drawing letters from the English alphabet *with equal probabilities*, and then throwing in the vowels 'a' and 'e'.

A more natural algorithm for constructing hands is drawing letters randomly from the English alphabet according to their relative frequencies (probability of occurrence) in the English language.

Implementing this extension is a good exercise.