Lab 6: String and Text I/O

Objectives

1. To be able to create and manipulate nonmodifiable string objects of class ***String***.
2. To be able to create and manipulate modifiable string objects of class ***StringBuffer/ StringBuilder*** class.
3. To write data into a file using ***PrintWriter*** class.
4. To print data from a file using ***Scanner*** class.

Syntax

Strings are Objects in Java and Strings can be declared and created using one of the following:

```
String stringName = new String("String Litral");

String stringName = new String(char [] arrayOfCharacters);

String stringName = "String Litral";
```

File Input and Output

To write data into a text file the **`java.io.PrintWriter`** class can be used. A ***PrintWriter*** object can be created as follows:

```
PrintWriter out = new PrintWrite(fileName);
```

To read data from a file **`java.util.Scanner`** class can be used in which the physical name of the file can be used instead of System.in that deals with consol to read from the keyboard.

```
Scanner in = new Scanner(fileName);
```

PrintWriter and Scanner classes UML

java.io.PrintWriter
+PrintWriter(filename: String) +print(s: String): void +print(c: char): void +print(cArray: char []): void +print(i: int): void +print(l: long): void +print(f: float): void +print(d: double): void +print(b: boolean: void Also contains the overloaded printf and println methods

Java.util.Scanner
+Scanner(source: String) +Scanner(source: File) +close(): void +hasNext(): boolean +nextBoolean() :boolean +nextInt() : int +nextByte(): byte +nextShort(): short +nextLong() :long +nextDouble() :double +nextString() : String +next(): String +nextLine() :String +useDelimiter(pattern: String): Scanner

String class UML

The following is the **java.lang.String** class containing the methods to deal with Strings:

Java.lang.String
+String() +String(String value) +String(char[] text) +String(char[] text, int offset, int count) +String(byte[] data, int offset, int length, String encoding) + String(byte[] data, String encoding) +length(): int +indexOf(int ch): int +indexOf(int ch, int fromIndex): int +lastIndexOf(int ch): int +lastIndexOf(int ch, int fromIndex): int +indexOf(String str): int +indexOf(String str, int fromIndex): int +(String str): int +lastIndexOf(String str, int fromIndex): int +valueOf(char[] text): String + valueOf(char[] text, int offset, int count): String +valueOf(boolean b): String +valueOf(char c): String + valueOf(int i): String +valueOf(long l): String +valueOf(float f): String +valueOf(double d): String + charAt(int index): char +getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin): void +getBytes(String encoding): byte[] +getBytes():byte[] +substring(int beginIndex): String +substring(int beginIndex, int endIndex): String +concat(String str): String +toCharArray(): char [] +equals(Object anObject): boolean +equalsIgnoreCase(String anotherString): boolean +compareTo(String anotherString): int +regionMatches(int toffset, String other, int ooffset, int len): boolean +regionMatches(boolean ignoreCase, int toffset String other, int ooffset, int len): boolean +startsWith(String prefix, int toffset): boolean +startsWith(String prefix): boolean +endsWith(String suffix): boolean + replace(char oldChar, char newChar): String +toLowerCase(Locale locale): String +String toLowerCase():String +String toUpperCase(Locale locale): String +String toUpperCase():String +trim():String

Exercises

1. You will create a class that will perform several different functions on Strings that are sent to it. All of the methods you create will be static, and the class should work in a similar manner to the Math class. Only the four methods listed below should be public.
   1) Create a method ***reverseString*** that receives a String and returns a String that is the exact reversal of the characters in the first String.

   2) Create a method ***isPalindrome*** that receives a String and returns a boolean value of true if the String is a Palindrome and false if it is not. A word is a palindrome if it reads the same forwards and backwards. For example, the word level is a palindrome. The idea of a palindrome can be extended to phrases or sentences if we ignore details like punctuation. Here are two familiar examples:

      Madam, I'm Adam
      A man, a plan, a canal: Panama

      We can recognize these more elaborate examples as palindromes by considering the text that is obtained by removing all spaces and punctuation marks and converting all letters to their lower-case form.

      Madam, I'm Adam → madamimadam
      A man, a plan, a canal: Panama → amanaplanacanalpanama

      If the "word" obtained from a phrase in this manner is a palindrome, then the phrase is a palindrome. Your method should ignore the case of the letters. A palindrome is determined by considering only alphabetic characters (a – z, A – Z) and numbers (0 – 9) as valid text.
      Use these sample phrases as inputs for your run outputs:
         radar
         Lewd did I live, & evil I did dwel.
         I like Java
         Straw? No, too stupid a fad, I put soot on warts.

   3) Create a method ***pigLatin*** that receives a String, converts the String to Pig Latin, and returns the new Pig Latinated word. There may be multiple words in your String, so you will need to have a recursive function that breaks down the String into single words and then reconstructs the sentence in Pig Latin. Here's how to translate the English word *englishWord* into the Pig Latin word *pigLatinWord*:

      1. If there are no vowels in *englishWord*, then *pigLatinWord* is just *englishWord* + "ay". (There are ten vowels: 'a', 'e', 'i', 'o', and 'u', and their uppercase counterparts. ‘y’ is not considered to be a vowel for the purposes of this exercise, i.e. my becomes myay, why becomes whyay, etc.)

      2. Else, if *englishWord* begins with a vowel, then *pigLatinWord* is just *englishWord* + "yay".
      3. Otherwise (if *englishWord* has a vowel in it and yet doesn't start with a vowel), then *pigLatinWord* is end + start + "ay", where end and start are defined as follows:

A) Let start be all of *englishWord* up to (but not including) its first vowel.
B) Let end be all of *englishWord* from its first vowel on.
C) But, if *englishWord* is capitalized, then capitalize end and "uncapitalize" start.

4) Create a method ***ShortHnaded*** that receives a String and returns the String converted into shorthand. The simplified shorthand form of a string is defined as follows:

    1. Replace these four words: "and" with "&", "to" with "2", "you" with "U", and "for" with "4".
    2. Remove all vowels ('a', 'e', 'i', 'o', 'u', whether lowercase or uppercase)

2. Create a text file ***myText.txt*** containing the following:

Tom Sawyer
Huck Finn
Luke Skywalker
Obiwan Kenobe
The quick brown fox jumps over the lazy dog

Write a Java program to read a text from ***myText.txt*** file. The program must read the text word by word, count the number of words in the file, and echo each (copy) word to the consol.

Note: Make sure not to miss any word, and not to repeat the last word by mistake.