Purpose: This exercise is to familiarize you with some of the String methods in Java as well as the concepts of cascading and composition.

To Receive Credit: Show the TA your working program before leaving class.

Activities:
1. Following the same procedure from the first lab, create a lab project named with your Virginia Tech pid, with the console application named StringMethods.
2. In Java, the String class has several methods. The ones we will use in this exercise are:
   a. toUpperCase
   b. toLowerCase
   c. length
   d. trim
   e. concat
   f. substring

Type in the following simple program using Strings. Make sure you type it EXACTLY as shown: notice in particular the tab at the beginning of poem1 and the spaces at the end of poem1, poem2 and poem3.

```java
import java.io.*;
class StringMethods {
    public static void main(String[] arg) {
        String poem1 = "Tell me not, in mournful numbers"; //use a tab
        String poem2 = "Life is but an empty dream!";
        String poem3 = "For the soul is dead that slumbers,";
        String poem4 = "And things are not what they seem.";

        System.out.println (poem1);
    } // end main
} // end StringMethods
```

By modifying the above program, add the necessary code to accomplish the following. Make one change at a time and run the program after each modification. Keep adding modifications to the program for each task so that your final program implements all of the tasks below. Before running the program, don’t forget to modify the Java Target in the Edit->Java Application Release Settings to be the name of the file/class: StringMethods.

a. Print poem1 in all uppercase.
b. Print poem1 in all lowercase.
c. Print the length of poem1 by using the appropriate String method to calculate the length. Print the length in the following format: “The length of poem1 is x” where x is the length. (Hint: think of the difference between println and print.)
d. Use the appropriate String method to remove the tab from the beginning of poem1 and print the resulting String.
e. Using the appropriate String method, append poem2 on to the end of poem1 and print the result.
f. Using the appropriate String method (hint: use only 1 argument), print only the phrase “in mournful numbers” from poem1.
g. Using the appropriate String method, print only the phrase “me not,” from poem1.

h. Create a new String using part of poem1 that reads “Students sigh, in mournful numbers”. Print this new String. (Hint: You may need to create new variables to hold intermediate results, including possibly the phrase “Students sigh”).

3. There are two methods that can be used when building a new String from three or more Strings:

1. Cascading
2. Composition

**Cascading** is the process of sending a message to an object to create a new object, which in turn is sent a message to create another new object, which in turn is sent a message to create yet another new object, and so on. The results of messages are used as receivers of additional messages. An example can be found from the code in the book that creates a new String “JFK” from the Strings “John”, “Fitzgerald” and “Kennedy”. (See pages 43-45 of Arnow and Weiss).

```
    initials = firstInit.concat(middleInit).concat(lastInit);
```

In this example, a concat message (with middleInit as argument) is sent to firstInit. The firstInit object returns a reference to a new String object (“JF”). Another concat message (with lastInit) is sent to the new String object whose reference was just returned. A reference to yet another new String object is returned.

**Composition** is the process of sending a message to an object to create a new object whose reference is used as an argument in a message. In turn, this may yield a reference to another new object, which then could be used as an argument in yet another message, and so on. The results of messages are used as arguments in additional message. An example can be found from code in the book that performs the same function as the code in the cascading example. (See page 46 of Arnow and Weiss).

```
    initials = firstInit.concat(middleInit.concat(lastInit));
```

In this example, a concat message (with lastInit as argument) is sent to middleInit. The middleInit object returns a reference to a new String object (consisting of the middle and last initials). The reference to this new object is sent as an argument in another concat message. A reference to yet another new String object is returned.

- By modifying the StringMethods class you just created, use both the (a) cascade and (b) composition technique to concatenate the following segments of a poem into one String. Store the results in variables called *cascade* and *composition* respectively. Print the resulting Strings *cascade* and *composition*.

```
    String poem1 = "Tell me not, in mournful numbers ";
    String poem2 = "Life is but an empty dream!  ";
    String poem3 = "For the soul is dead that slumbers, ";
    String poem4 = "And things are not what they seem.";
```