THOMPSON RIVERS UNIVERSITY

COMP 1130
Computer Programming I
Midterm 2 Assignment A – *Solutions*

[10 marks] Question 1

Computers are a great help in science, especially with tedious activities. Write a program to determine differences in DNA sequences of base pairs, which are described with the letters: A, G, C, T (https://ghr.nlm.nih.gov/primer/basics/dna)

The program prompts for two strings, containing DNA sequences with at least 8 letters (upper or lowercase), for example:

```
Enter sequence 1: AaacaacttcGTAAGTATA
Enter sequence 2: AAGTTCCTtcgtaagTATA
```

Note: There is no scientific verification of the DNA base pairs, the program just treats them as a sequence of characters. 😉

The processing rules,

Part 1: If the two sequences have different lengths or less than 8 chars, the comparison is incomplete, and the program displays "Sequence must be longer" or "Sequences must be the same length" and stops immediately.
Assume all letters are correct (that users did not type anything by A,G,C,T.)

Otherwise, the program loops through the strings, comparing the DNA letter pairs at each *position* in the both sequences and displaying in a table with: **=** (letters are the same) or **!** (not the same) – *see example:*

```
Pos: S1 - S2
  1:  A = A
  2:  A = A
  3:  A ! G
  4:  C ! T
  5:  A ! T
  6:  A ! C
  7:  C = C
  8:  T = T
  ...
 18:  T = T
 19:  A = A
```

Part 2: Below the table, the program displays a summary,

- the length of the sequences:
```
DNA Sequence length: 19
```
- count of the number of differences:
```
There are 4 differences between the sequences.
```
or:
```
There are no differences, sequences are a match.
```

Part 3: Calculate and display the "weight" of each sequence, where A=1, G=2, C=3, T=4.
In the example, Sequence 1 has weight: 42

Part 4: *Implemented in parts above,*

- implement a method **inputStrLine()**, as seen in class, that both prompts the user with a string and returns the user's full input string
- implement a method **compareChar()** that compares two char parameters, returning a boolean value: *true* if the characters are the same, *false* if the characters are not. *Use this method when comparing the DNA symbols*
- use the technique of *concatenating all the output in a single output string*, that is displayed at the end with a single System.out.println()

HINTS:

- recall the String class methods: .**charAt(), .toUpperCase(), .substring(), .length()**, etc.
- recall the Scanner class methods: **.next()** and **.nextLine()**
- to help line up the data in the table, consider using **String.format()** or **System.out.printf()**,
  - example: `String.format ("%4d:  %c %c %c", pos, char1, symbol, char2)`
    which right-justifies **int** pos *in a width of 4, and displays* **chars** *char1, symbol, and char2*

Solution:

```
/* DNASequenceChecker.java - check two sequences of DNA symbols, detecting & counting differences.
 *                           only check when sequences are the same length.
 *
 *    (note: the program is not written as efficient as it could be, for the sake of being clear
 *           of the processing steps involved.)
 */

import java.util.Scanner;

public class DNASequenceChecker
{
     public static void main (String[] args)
     {
          String dnaStr1="", dnaStr2="";   // the two DNA strings provided by the user
          char dnaChar1=' ', dnaChar2=' '; // characters at position in the DNA strings
          int weight1=0, weight2=0;        // weights of sequences
          char compareSymbol=' ';          // comparison symbol: = or !
          int diff=0;                      // dna differences count

          String out = "";                 // output String, concatenating all the output

          //----------
          dnaStr1 = inputStrLine("Sequence 1: ").toUpperCase();  // get DNA string 1, in uppercase
          dnaStr2 = inputStrLine("Sequence 2: ").toUpperCase();  // get DNA string 2, in uppercase

          if ( dnaStr1.length() != dnaStr2.length() )   // if string lengths are different
          {
               out += "Sequences must be the same length";    // concat message
          }
          else if ( dnaStr1.length() < 8 || dnaStr2.length() < 8 )  // if at least one string is < 8 letters
          {
               out += "Sequences must be greater than 8 letters";    // concat message
          }
          else            // strings lengths are okay
          {
               out += " Pos: S1 - S2\n";    // concat table title

               for (int i=0; i<dnaStr1.length(); i++)   // loop through all characters in the strings
               {
                    dnaChar1 = dnaStr1.charAt(i);           // symbol in string 1
                    dnaChar2 = dnaStr2.charAt(i);           // symbol in string 2

                    // determine which comparison symbol: = or !
                    if (compareChar(dnaChar1, dnaChar2))     // if true (the same)
                    {
                         compareSymbol = '=';
                    }
                    else                                     // if false (not the same)
                    {
                         compareSymbol = '!';
                         diff++;                             // increment difference count
                    }

                    // calculate weights - using a new method to reduce redundant code
                    weight1 = weight1 + getWeightValue(dnaChar1);
                    weight2 = weight2 + getWeightValue(dnaChar2);

                    // concat formatted row in table
                    out += String.format ("%4d:  %c %c %c \n", (i+1), dnaChar1, compareSymbol, dnaChar2);
               }// end of loop processing each character in both strings
```

```
            out += String.format ("     %3d %3d  - weights\n", weight1, weight2);

            if ( diff == 0 )      // if there are no differences
            {
                out += "There are no differences between the sequences.";  // concat summary
            }
            else                  // there are differences
            {
                out += "There are "+diff+" differences between the sequences.";  // concat summary
            }
        }

        System.out.println (out);     // display results
    }// end of main()

    // inputStrLine() - display a prompt to the user, and return the user's entire input line
    public static String inputStrLine (String prompt)
    {
        Scanner scan = new Scanner (System.in);  // console input stream

        System.out.print (prompt);       // display prompt string to user

        return ( scan.nextLine() );      // return input
    }// end of inputStrLine()

    // compareChar() - compares to characters together, returning true if the same and false otherwise
    public static boolean compareChar (char ch1, char ch2)
    {
        return ( ch1 == ch2 );            // return if characters are equal (true), or not (false)
    }// end of compareChar()

    // getWeightValue() - determines the weight value of a letter: A=1, G=2, C=3, T=4
    public static int getWeightValue (char letter)
    {
        int value=0;       // weight value to return

        switch ( letter )       // could also be written as a shorter if_else_if, but that's no fun
        {
            case 'A':
                value = 1;
                break;
            case 'G':
                value = 2;
                break;
            case 'C':
                value = 3;
                break;
            case 'T':
                value = 4;
                break;
        }

        return ( value );  // return value
    }// end of getWeightValue()

}// end of class
```

[10 marks] Question 2

Write a program that tests the “fairness” of the random number generated in Java. The test is pretty simple, but will (or at least, will hopefully) show that all values within a range have the same probability of being selected (unbiased randomness).

Part 1: Generate 60 random values in the range 1..5, and count how many times each number is chosen (5 int counting variables). Once this is finished, a table is produced showing the counts of each value.
The expected outcome is that all values should have, relatively, the same counts—this is called a “Uniform Distribution” and is a fundamental aspect of unbiased randomness. For the range 1..5, after 60 random picks the expectation is 12 picks per value.

For example, the output of the program could be:

```
60 Random Values in range 1..5:
      1:  9
      2: 11
      3: 10
      4: 18
      5: 12
```

Part 2: To the right of each number, graph the “size” of each count, using '*' symbols.

For example, the output could be:

```
60 Random Values in range 1..5:
      1:  9 *********
      2: 11 ***********
      3: 10 **********
      4: 18 ******************
      5: 12 ************
```

Part 3: Surround the above operations with a loop that repeats 10 times, producing a larger sample set to verify the unbiased randomness, which is better than running the program 10 times. (Note: Don’t forget to reset the counting variables to zero.) AND send the output to a file **randomtext.txt**. *Attach this file to your assignment submission.*

For example, the output could be:

```
60 Random Values in range 1..5:
Iteration 1:
      1:  9 *********
      2: 11 ***********
      3: 10 **********
      4: 18 ******************
      5: 12 ************
Iteration 2:
      1: 12 ************
      2: 13 *************
      3: 17 *****************
      4:  7 *******
      5: 11 ***********

Iteration 3:
      1: 12 ************
      2: 14 *************
      3: 16 ***************
      4:  5 *****
      5: 13 *************
...

Iteration 10:
```

```
1: 17 *****************
2:  8 ********
3: 10 **********
4: 14 **************
5: 11 ***********
```

Part 4: *Implemented in parts above,*

- implement a method **stars()** that has one int parameter (**n**) and returns a String. The method returns a String composed with as many '*' chars as indicated by **n**. Use this method to help create the graph line of stars.
- use the technique of *concatenating all the output in a single output string*, that is displayed once everything is finished process, to make it easier to display BOTH to the screen and output file

HINTS:

- to format the output for an int in a specific width, with .printf() or String.format(),
    - example: `String.format ("%3d:  ", 8)` returns a string such that: " 8"

```
Solution:
/* RandomTester.java - check the fairness (unbiased randomness) of the Java random number generator.
 *    The program generates random numbers in the value range 1..5, to see if each value is picked
 *    approximately the same number of times.
 */

import java.util.Random;
import java.io.*;

public class RandomTester
{
     public static void main (String[] args)
          throws IOException         // to deal with any file I/O errors
     {
          String out="";      // common output string

          PrintWriter outFile = new PrintWriter ( "randomtext.txt" );   // output file

          Random gnr8 = new Random();   // random number generator
          int rand = 0;                 // value from randome number generator
          int MAXITER = 60;             // maximum number of iterations

          int r1=0, r2=0, r3=0, r4=0, r5=0;  // hold count of each value in range: 1..5

          out += MAXITER + " Random Values in range: 1..5:\n";

          for ( int n=1; n<=10; n++)    // generate test 10 times
          {
               r1=0; r2=0; r3=0; r4=0; r5=0;      // reset value counters each test loop
               for ( int i=1; i<=MAXITER; i++)    // to generate 60 random numbers
               {
                    rand = gnr8.nextInt(5)+1;         // generator random number (0..4)+1: 1..5

                    if (rand==1)                      // inc. count of appropriate variable
                         r1++;
                    else if (rand==2)
                         r2++;
                    else if (rand==3)
                         r3++;
                    else if (rand==4)
                         r4++;
                    else
                         r5++;
               }// end of random number generation loop
```

```
            // summary
            out += "Iteraton "+n+":\n";
            out += String.format ("\t%d: %3d: ", 1, r1) + stars(r1) +"\n";
            out += String.format ("\t%d: %3d: ", 2, r2) + stars(r2) +"\n";
            out += String.format ("\t%d: %3d: ", 3, r3) + stars(r3) +"\n";
            out += String.format ("\t%d: %3d: ", 4, r4) + stars(r4) +"\n";
            out += String.format ("\t%d: %3d: ", 5, r5) + stars(r5) +"\n";
        }// end of test count loop

        System.out.println (out);     // display results
        outFile.println (out);        // write results to file
        outFile.close();              // close output file
    }// end of main()


    // stars() - return a String of '*' chars, as indicated by n
    public static String stars (int n)
    {
        String s = "";               // hold string of stars

        for (int i=0; i<n; i++)     // count n times
        {
            s += "*";                  // concatenate a "*" to the string
        }

        return (s);
    }// end of stars()

}// end of class
```

[10 marks] Question 3

Assume the only console output method available is **display(c)**, shown below. *This means you cannot use System.out.print(), System.out.println(), nor System.out.printf().*

```
public static void display (char c)
{
     System.out.print (c);
}
```

Implement the following methods, and write a **main()** to test each. Apply the concept of *overloading*, and that the new methods call other new methods, for full marks.

Part 1: Common output methods:
- **display (String str)** – displays all the characters in the String **str**
- **displayln (String str)** – displays all the characters in the String **str**, following by newline character ('\n')
- **display (int val)** – displays the integer in variable **val**
- **display (double val)** – displays the double in variable **val**
- **displayln ()** – displays only a newline character

Part 2: Special output methods:
- **displayRepeat (String str, int rep)** – displays the String **str**, repeated as many times indicated in variable **rep**
- **displaySkip (String str, int j)** – displays every jth char in the String **str** (ex: "abcdefghi",2 → display: "bdfh")
- **displayTriangle(String str))** – displays all the characters in the String **str**, in a triangle form,
  ex: displayTriangle("pies") shows:

```
p
ii
eee
ssss
```

```
Solution:
/* DisplayMethods.java - a tester class to implement & verify the dislay() methods.
 */

public class DisplayMethods
{
     public static void main (String[] args)
     {
          // testing the display methods
          display ("Bumble Bees\n");         // display(String)
          displayln ("More bumble bees");    // displayln(String)
          displayInt (42);                   // displayInt(int)
          displayln ();                      // displayln() - newline only
          displayDouble (42.13);             // displayDouble(double)
          displayln ();
          displayRepeat ("Bob",4);           // displayRepeat(String,int)
          displayln ();
          displaySkip ("abcdefghi",2);       // displaySkip(String,int)
          displayln ();
          displayTriangle("pies");           // displayTriangle(String)

     }// end of main()

     // *** the display methods

     // display() - display a single character to the console
     public static void display (char c)
     {
          System.out.print (c);
     }// end of display()
```

```
    // display() - display a string to the console
    public static void display (String str)
    {
         for (int i=0; i<str.length(); i++)  // loop through all characters
         {
              display ( str.charAt(i) );        // display character
         }
    }// end of display()

    // displayln() - display a string to the console, with a newline
    public static void displayln (String str)
    {
         display (str + "\n");      // display string concat'd with newline
    }// end of displayln()

    // displayln()) - display newline only
    public static void displayln ()
    {
         displayln ("");            // display empty string with newline
    }// end of displayln()

    // displayInt() - display an int to the console
    public static void displayInt (int val)
    {
         display ( "" + val);       // display int concat'd with string
    }// end of displayInt()

    // displayDouble() - display a double to the console
    public static void displayDouble (double val)
    {
         display ( "" + val);       // display double concat'd with string
    }// end of displayDouble()

    // dsplayRepeat() - display a string repeated a specific number of times
    public static void displayRepeat (String str, int rep)
    {
         String res="";        // result string
         for (int i=0; i<rep; i++)
         {
              res = res + str;      // could also use: display(str), but this is actually faster
         }
         display (res);
    }// end of displayRepeat()

    // displaySkip() - display every jth character in string
    public static void displaySkip (String str, int j)
    {
         for (int i=j-1; i<str.length(); i=i+j)  // count i from j, skipping by j
         {
              display ( str.charAt(i) );         // display character; could also use a result string
         }
    }// end of displaySkip()

    // displayTriangle() - display all characters in str, in triangle form
    public static void displayTriangle (String str)
    {
         for (int i=0; i<str.length(); i++)    // i is the index position in the string
         {
              displayRepeat ( ""+str.charAt(i), i+1 );   // display character at i, i times
              displayln ();                              // display newline between each char at i
         }
    }// end of displayTriangle()
}
```