PROBLEM 1 : (*Blasting out DNA (24 points)*)

In this problem, we use a single strand of DNA to generate fragments of DNA. A DNA strand is made up of the nucleotides A, C, G and T. For example, a DNA strand might be "CGTA". Each symbol has a complement. A and T are complements of each other, and C and G are complements of each other. The complement of "CGTA" is the complement of each symbol, resulting in "GCAT".

We want to generate DNA fragments from a DNA strand and a reactant (one of the four nucleotides). A DNA fragment is generated by generating a string of complements, stopping randomly at the chosen reactant.

For example, suppose the DNA strand is "CGATACTCTGT" and the reactant is "A". Shown below are four copies of this DNA strand on the first line with the four possible fragments below it. Each symbol in the fragment is the complement of the corresponding symbol directly above it and each fragment has to end in the reactant A.

```
CGATACTCTGT   CGATACTCTGT   CGATACTCTGT   CGATACTCTGT
GCTA          GCTATGA       GCTATGAGA     GCTATGAGACA
```

In this problem, given a DNA strand and a reactant we will randomly generate an array of possible fragments. We will write this in three parts.

First we give you the method complement which given a string of one letter (C,G,T or A) gives you the complement letter. You may want to use this method.

```
public String complement(String str)
{
   if (str.equals("A"))
      return "T";
   if (str.equals("T"))
      return "A";
   if (str.equals("C"))
      return "G";
   return "C";
}
```

PART A. (8) Complete the method *findPositionOfAllOccurences* which has two String parameters: dna is a string of nucleotides and nuc is one nucleotide. This method returns an `ArrayList` of the positions of nuc in dna. For example, if dna is AGCATA and nuc is A, then return the ArrayList of integers 0, 3, 5, which are the positions of the A in the dna strand.

```
public ArrayList<Integer> findPositionOfAllOccurences(String dna, String nuc)
{
```

PART B. (8) Complete the method *createOneFragment* that is given a dna String and a position in the String and returns one fragment that ends at that position. The fragment is created by generating a new String in which each character is the complement of the character in the dna String. The fragment may be shorter then the dna String as it must stop at the lastposition parameter.

For example, suppose the dna string is "CGTAG" and the last position is 2, then the fragment returned is "GCA", since GCA is the complement of CGT and the fragment ends at position 2.

```
public String createOneFragment (String dna, int lastPosition)
{
```

PART C. (8) Complete the method *generateFragments* that has three parameters: dna is a strand of dna, reactant is a single nucleotide and number is the number of fragments to generate. This method generates fragments of different random lengths, all ending in the same reactant.

For example, suppose the dna string is "CGTATGT", the reactant is "A" and number is 4, then an array of four random fragments are generated that all end in A, which is the complement of "T". These four fragments might be "GCA", "GCATACA", "GCA", and "GCATA".

```
public  String [] generateFragments(String dna, String reactant, int number)
{
```

PROBLEM 2 : (*Extra curricular activities (28 points)*)

You are given the data from Dook University of the clubs and members of each club in the following format. The data for each club is on two lines. The first line is the name of the club. The next line is the name of all the members in the club, with members separated by a colon.

The sample data below shows the members of three clubs.

```
Indoor and Outdoor Tennis
Jeff Forbes:Hillary Rodham Clinton:Mary Lou Retton
Gourmet Cooking
Susan Rodger:Oprah Winfrey:Cay Horstmann:Mary Lou Retton:Owen Astrachan
Photography
Owen Astrachan:Oprah Winfrey:Mary Lou Retton
```

We will create the Club class to store and manipulate the data for one club and the DookClubs class to store and manipulate data on all the clubs.

```
public class Club
{
    private String myName;
    private Set<String> myMembers;

    public String getName() { return myName; }
    public Set<String> getMembers() { return myMembers; }
    // constructor not shown

}

public class DookClubs
{
   private ArrayList<Club> myClubs;

   // constructor and member functions not shown

}
```

PART A. (6) Complete the Club class constructor which is passed in the name of the club and the members in the club, in the format described earlier with members separated by colons.

```
public  Club(String name, String members)
{




}
```

PART B. (8) Complete the DookClubs class constructor which is passed in a Scanner that is ready to read from a file. The constructor reads all the club data from the file and stores the information in `myClubs`.

```
public  DookClubs (Scanner input)
{


}
```

PART C. (8) Complete the DookClubs member function *allMembers* that returns an ArrayList of all the people from Dook University in a club.

Using the datafile given earlier, the `ArrayList` would contain: Jeff Forbes, Hillary Rodham Clinton, Mary Lou Retton, Susan Rodger, Oprah Winfrey, Cay Horstmann, and Owen Astrachan.

```
public ArrayList<String> allMembers ()
{




}
```

PART D. (6) Complete the DookClubs member function peopleNotInClubs that takes an `ArrayList` of names and returns an `Arraylist` of all the people from students who are *not* in a club.

For example, if the ArrayList students contained the names: Michael Jackson, Owen Astrachan, Oprah Winfrey, and Peter Lange, then `peopleNotInClubs(students)` would return the `ArrayList` containing Michael Jackson and Peter Lange, as they are not in any clubs.

```
public ArrayList<String> peopleNotInClubs (ArrayList<String> students)
{

}
```

PROBLEM 3 : (*Sorted Pictures (18 points)*)

In this problem, you will create a Pixmap *Command* that changes the target Pixmap by sorting each column of Colors by *luminance*. Luminance is a measure of the intensity of light as perceived by the human eye. White would have the most luminance while black would have the least. By sorting each column by luminance, the darkest colors will be on the top and brightest colors will be on the bottom. You should not change the value of any individual color, just its position in its column.

You computed luminance for the `WeightedGrayScale` filter as part of the Pixmap classwork. If you have a color (R, G, B) for the red, green, and blue values, the luminance is defined as $0.3R + 0.59G + 0.11B$.

PART A. (8) Complete the *LumComparator* class below. You will need to write:

1. the `luminance` method that returns the luminance of a `Color`
2. the `compare` method that should use the `luminance` method to return a negative integer, zero, or a positive integer if the first color is less than, equal to, or greater than the second in terms of luminance.

```
public class LumComparator implements Comparator<Color>
{
    // Computes the luminance of c where luminance is defined as
    // 0.3R + 0.59G + 0.11B
    private int luminance(Color c)
    {
```

```
    }

    // returns:
    //  < 0 if the luminance of c1 is less than c2
    //   0  if the luminance of c1 is equal to c2
    //  > 0 if the luminance  of c1 is greater than c2
    public int compare (Color c1, Color c2)
    {
```

PART B: (10) Complete the execute method below so that the resulting image has each column sorted by luminance. You can put the elements of each column in an `ArrayList` and sort that list using using `Collections.sort()` with the `LumComparator`.

```
public class Student extends Command
{
    public Student ()
    {
        super("Sort Colors");
    }


   public void execute (Pixmap target)
    {
        Dimension size = target.getSize();
```

Throughout this test, assume that the following classes and methods are available. These classes are taken directly from the material used in class.

```
public class String {
    public int length ()
    // Returns a substring of this string that
    // begins at the specified beginIndex and
    // extends to the character at index
    // endIndex - 1.
    public String substring (int beginIndex,
                             int endIndex)
    // Returns a substring of this string that
    // begins at the specified beginIndex
    public String substring (int beginIndex)
    // Returns position of the first
    // occurrence  of str, -1 if not found
    public int indexOf (String str)
    // Returns the position of the first
    // occurrence of str after index start
    public int indexOf (String str, int start)
    // returns character at position index
    public char charAt(int index)
    // returns true if str has the exact
    // same characters in the same order
    public boolean equals(String str)
}

public class Collections {
   // Sorts the specified list according
   // to the order induced by thecomparator
   public static void sort(List list,
                        Comparator c)
}

public class Integer {
    // Returns the argument as a signed integer
    public int parseInt(String s)
}

public class Pixmap {
    public Dimension getSize()
    // Get the pixel at (row,col)
    public Color getColor (int row, int col)
    // Set the pixel at (row,col)
    public void setColor (int row, int col, Color value)
}

pubclic class Dimension {
    public int width
    public int height
}

public class Color {
    // Creates a color with the specified red,
    // green, and blue values in the range
    // (0 - 255)
    public Color(int r,int g,int b)
    // Returns color components
    public int getRed()
    public int getGreen()
    public int getBlue()
}

public class Random {
    // Create a new random number generator
    public Random()
    // Returns a pseudorandom, uniformly
    // distributed value in [0,n)
    public int nextInt(int n)
}

public class ArrayList {
    // Constructs an empty list
    public ArrayList ()
    // Returns the number of elements
    public int size ()
    // Returns element at position index
    public Object get (int index)
    // Replaces the item at position index
    // with element.
    public Object set (int index, Object element)
    // Appends specified element
    public boolean add (Object o)
}

public class Scanner {
    // Create Scanner that reads data from a file.
    public Scanner (File file)
    // Create Scanner that reads data from a string.
    public Scanner (String str)
    // Change delimiters used to separate items
    public void useDelimiter (String characters)
    // Check if more items are available
    public boolean hasNext ()
    // Get next delimited item as a string
    public String next ()
    // Get next line as a string
    public String nextLine ()
    // Get next delimited item as an integer value
    public int nextInt ()
    // Get next delimited item as a Double value
    public double nextDouble ()
}
```

```
public class TreeSet
{
   // creates an empty TreeSet
   public TreeSet()

   // adds object e to the TreeSet
   boolean add(Object e)

   // adds all elements from a collection to this set
   boolean addAll(Collection c)

   // Retains only the elements in this set that are contained
   // in the specified collection - returns true if set changed
   boolean retainAll(Collection c)

   // Removes from this set all of its elements that are contained
   // in the specified collection
   boolean removeAll(Collection c)

   // removes all objects from the TreeSet
   void clear()

   // returns true if e is in the set, othewise returns false
   boolean contains(Object e)

  // returns true if set is empty, otherwise returns false
  boolean isempty()

  // returns an Iterator for the set
  Iterator<Object> iterator()

  // removes the object e from the set
  boolean remove(Object e)

  // returns the number of elements in the set
  int size()

}
```