CMSC 350 Homework 1

Create a class that can be used to test data structure - similar to the StudentA.java example found shown below:

StudentA.java

```
import java.util.Scanner;
import java.util.ArrayList;

class StudentA implements Comparable <StudentA> {
   static java.util.Random rn = new java.util.Random ();
   static ArrayList <String> firstNames = new ArrayList <>();
   static ArrayList <String>  lastNames = new ArrayList <>();
   static SORTBY sortBy = SORTBY.LAST;
   static int nextUID = 1;

   String first, last;
   double gpa  = 0;
   int credits = 0;
   int uid     = 0;

   static {
      try {
         java.util.Scanner scFirst = new java.util.Scanner (new
java.io.File("firstNames.txt"));
         java.util.Scanner scLast  = new java.util.Scanner (new
java.io.File("lastNames.txt"));
         while (scFirst.hasNext()) firstNames.add (scFirst.next());
         while ( scLast.hasNext())  lastNames.add ( scLast.next());
      }
      catch (java.io.FileNotFoundException e) {
         System.out.println (e);
      } // end try catch
   } // end static intializer

   enum SORTBY {LAST, FIRST, CREDITS, GPA}

   public StudentA (String st) {this (new Scanner (st)) }

   public StudentA (Scanner sc) {
      uid     = nextUID++;
      first   = sc.next();
      last    = sc.next();
      credits = sc.nextInt();
      gpa     = sc.nextDouble();
   } // end Scanner constructor

   public StudentA () {uid = nextUID++;} // no parameter constructor

   public int compareTo (StudentA x) {
     switch (sortBy) {
       case LAST   : return last.compareTo (x.last);
```

```
        case FIRST  : return first.compareTo (x.first);
        case CREDITS: return credits - x.credits;
        case GPA    : return (gpa > x.gpa)? 1 : -1;
      } // end switch
      return 0;
   } // end compareTo for Comparable interface

   public String toString () {
      return String.format ("%5d %15s, %15s: %5d %10.2f", uid, last, first,
credits, gpa);
   } // end method toString

   public static StudentA [] makeRandom (int m) {
      StudentA [] sa = new StudentA [m];
      for (int i = 0; i < sa.length; i++) {
        sa[i]         = new StudentA ();
        sa[i].first   = firstNames.get (rn.nextInt (firstNames.size()));
        sa[i].last    =  lastNames.get (rn.nextInt ( lastNames.size()));
        sa[i].credits = rn.nextInt (120);
        sa[i].gpa     = rn.nextDouble () * 4.0;
      } // end for each student to instantiate
      return sa;
   } // end method makeRanom

   public static void main (String args []) {
      System.out.println (new StudentA ("james robinson 35 3.98"));
      StudentA [] x = makeRandom (10);
      for (StudentA m: x)
        System.out.println (m);
      java.util.Arrays.sort (x);
      System.out.println ("---- SORTED Last -----");
      for (StudentA m: x)
        System.out.println (m);
      System.out.println ("---- SORTED First -----");
      StudentA.sortBy = SORTBY.FIRST;
      java.util.Arrays.sort (x);
      for (StudentA m: x)
        System.out.println (m);
      System.out.println ("---- SORTED Credits -----");
      StudentA.sortBy = SORTBY.CREDITS;
      java.util.Arrays.sort (x);
      for (StudentA m: x)
        System.out.println (m);
      System.out.println ("---- SORTED GPA -----");
      StudentA.sortBy = SORTBY.GPA;
      java.util.Arrays.sort (x);
      for (StudentA m: x)
        System.out.println (m);
   } // end main
} // end class StudentA
```

Note that this class will be used in the projects in the rest of this course to test various data structures and algorithms

The following are requirements for this homework:

- Unique index integer fields
- Use at least 5 other fields of various, including at least one of each of String, int and double types.
    - assume that the String's have no spaces within them
- Write a method that will create an array of N random instances of this class, where N is an integer parameter to this method
- Provide a toString method that will format an instance of this class nicely
- Provide appropriate constructors
- Write methods that will write and read text files of this class
    - to make this easy, assume that the first line of the file is the number of elements in the file
    - make the file one line per entry
    - use spaces as field delimiters in the files
    - use the toString method of the class to write to the file
- Provide a good GUI interface to control and test this class, at a minimum:
    - use buttons to control the functions available
    - use JFileChoose for selecting input and output files
    - use a JTextArea and JScrollPane to create a scrolling text area for output
    - the window should resize nicely - use BorderLayout with the text area in the center

Submit your java file(s) along with your test plan and demonstration of successful compilation and running for all test cases to the homework 1 submission area.