**Reviewing Assignment**

Lab Assignment 9

|  |  |
| --- | --- |
| Started: | Nov 24, 2014 9:48 PM |
| Finished: | Nov 28, 2014 1:46 PM - late |

 1 of 1

**Lab Assignment 9     Total Grade: 17.25   (of possible 20 points)**

**Score: 17.25   (of possible 20 points)**

**Assignment 9 - Changing Sort Keys**

There is only one option this week.  Make sure you have read and understood

* both ***modules A*** and ***B*** this week, and
* ***module 2R - Lab Homework Requirements***

before submitting this assignment. Hand in only one program, please.

*While it may appear long at first, this lab is easier than it looks, so stay calm. It is easier because:*

1. *it uses material and code that you learned, compiled and ran over a week ago when you studied****Module 8,****and*
2. *below, I give you very explicit directions on what to do for each of the very short methods you need to write.*

*You will start with my existing****Student****and****StudentArrayUtilities****classes that you have already compiled and run when you read the modules. Then you will modify each of these classes as described below and provide a new test client.*

OPTION A (the only one)

Understand the Application

Sort Flexibility (Student class)

In ***week 8***, we saw an example of a **Student** class that provided a **compareTwoStudents()** method. We needed to define such a method because our ***sort algorithm***(which was in the **StudentArrayUtilities** (**SAU**) class) had to have a basis for comparing two **Student** objects, the foundation of **SAU**s sorting algorithm.  Since a **Student**is a compound data type, not a **double** or a **String**, there was no pre-defined **compareTo()** available, so we created our own **compareTwoStudents()**, which  was based on the spelling of the ***last name***.  You can review those notes to see its definition.

We want to make the sort more flexible in this assignment.  Sometimes we want to sort on ***last name***, as we did in the lectures, but other times we may choose to sort on the***first name*** (useful for informal communication) or ***total points*** (useful for analyzing statistics).  We could go into the **compareTwoStudents()** method and change its definition to work on ***first name*** or ***total points***, but all that does is replace the inflexibility of***last name*** with the inflexibility of some other criteria.  So here's the  plan:  we will provide a static **Student** class method called **setSortKey()** which will establish a new sort criteria.  A client will invoke it whenever it wants to switch to a new basis for comparison.  The client will pass a parameter to **setSortKey()**telling it which one of the three **Student** fields it wants future sort algorithms to use.

Console Output (SAU class)

Another change we'll make affects the output modality.  Rather than using **JOptionPane**, we want to make the class "U.I. neutral." So we will replace **SAU**'s **printArray()**method with a**toString()** method, and let the client choose how to use that.  In our example **main()**we will be sending the String array to the ***console***, only.

Median (SAU class)

We'll add one more method to our **SAU** class:**double getMedianDestructive( Student[] someArray )**.  This method will return the ***median*** of the **totalPoints** values in  an array.  Look up ***median***.  It is defined as the "middle-value" and is easily computed, but you first have to sort the array in order to find it.  Fortunately, you already have the sort method.

Client

Our client will declare *four****Student****arrays* to make sure our median works:  one array that had an odd number of students (***15***), one that has an even number of students (***16***), one that has a ***single student*** and one that has ***no students*** (we will define the median of an array with no elements to be 0.0).

We'll test the **sortKey** and sort algorithm only on the even numbered array, and then we will test our median computation on all four arrays.

The Program Spec

These changes should not be complicated as long as you read carefully and follow directions. New and modified members/methods are all very short, so stay focused and apply what you learned back in ***week 8***.

*Additions* to the Student Class

We will add the following members to the class in the modules.

Public static int constants (finals):

* **SORT\_BY\_FIRST** = 88
* **SORT\_BY\_LAST**= 98
* **SORT\_BY\_POINTS**= 108

These are the three ***sort keys*** that will be used by the client and the class, alike, to keep track of, or set, the ***sort key***.   If the client wants to establish a new ***sort key***, it will pass one of these tokens (say **Student.SORT\_BY\_FIRST**) to the setter described next.    I have intentionally made the literal values non-contiguous so you would not rely on their particular values in your logic.   You should be able to change the values without breaking your program (but you don't have to change them;  use the three values above).

Private static int:

* **sortKey** - this will always have one of the three constants above as its value. Make sure it initially has **SORT\_BY\_LAST** in it, but after the client changes it, it could be any of the above constants.

You should supply the following simple ***public static methods***:

* **boolean setSortKey( int key )** -- a ***mutator*** for the member **sortKey**.
* **int getSortKey()** -- an ***accessor*** for **sortKey**.

*Modification* to the Student Class

* **compareTwoStudents( ... )** -- same signatures as in the modules, but now this method has to look at the **sortKey** and compare the two **Students** based on the currently active **sortKey**.  A **switch** statement with three different expressions is all you need, and each expression will be very similar to the one already in the modules (in fact one will be identical).  As you saw in the modules, it needs to return an **int**, which is ***positive***, if the first student is ***greater*** than the second,***negative*** if***less than***, and ***zero*** if they are the ***same***, *based on the current value of****sortKey****, of course*.

*Change* to the StudentArrayUtilities Class

* **Replace printArray() with toString().**  Generate the same kind of **String**, but instead of sending it to the screen, return it to the client.
* Add a ***public*** static method **double getMedianDestructive(Student[] array)** - This computes and returns the ***median*** of the total scores of all the students in the array  The details are simple, but you have to take them each carefully:
  + Dispose of the cases of an empty array (0 elements) and one-element array.  Empty arrays return 0.0, always, and one-element array returns its one and only**Student**'s **totalPoints**.  (This second case can actually be skipped if you handle the next cases correctly, but it doesn't hurt to do it separately, here.)
  + **Even-numbered arrays >= 2** elements:  find the two middle elements and return their average of their total points.
  + **Odd-numbered arrays >= 3** elements:  return the total points of the exact middle element.
  + **Special Note**:  This method has to do the following.  It must ***sort the array*** according to **totalScore** in order to get the medians, and that's easy since we already have the sort method.  Then it has to find the middle-student's score (e.g., if the array is size **21**, the middle element is the score in**array[10]**, *after the sort*).  But, before doing the sort, it also has to change the **sortKey** of the **Student** class to **SORT\_BY\_POINTS**.  One detail, that you may not have thought of, is that, at the very start of the method, it needs to save the***client's sort key***.  Then, before returning, restore the client's sort key.  This method doesn't know what that sort key might be, but there is an accessor **getSortKey()** that will answer that question.
  + This method has the word "**Destructive**" in its name to remind the client that it may (and usually will) modify the order of the array, since it is going to sort the array by ***total points*** in the process of computing the ***median***.  However, it will not destroy or modify the client's **sortKey**when the method returns to client (see previous bullet).

The Foothill main()

Our client will declare *four****Student****arrays*:  using direct initialization, as in the modules: no user input.  The array sizes should be 15, 16, 1 and 0.  The second array can be the same as the first with one extra **Student** tagged onto the end.  Each array should be initialized in no particular order:  unsorted in all fields.

Using the largest, even numbered, array:

Using the largest, even numbered, array:

1. display the array immediately before calling a sort method,
2. sort the array using the default (initial) ***sort key*** and display,
3. change the ***sort key*** to ***first name***, sort and display,
4. change the ***sort key*** to***total score***, sort and display,
5. **setSortKey()** to***first name***, call the **getMedianDestructive()** method and display the***median score***. and finally
6. call **getSortKey()** to make sure that the **getMedianDestructive()** method preserved the client's **sortKey** value of***first name*** that was just set prior to the**getMedianDestructive()** call.

Using each of the other three arrays:

* get the median of each array and display.  No other testing needed in this part.

Here's a sample output, but you must not use my arrays.  Make your own as per the spec above.

Before:

smith, fred points: 95

bauer, jack points: 123

jacobs, carrie points: 195

renquist, abe points: 148

zz-error, trevor points: 108

perry, fred points: 225

loceff, fred points: 44

stollings, pamela points: 452

charters, rodney points: 295

cassar, john points: 321

Sorting by default ---------------

After:

bauer, jack points: 123

cassar, john points: 321

charters, rodney points: 295

jacobs, carrie points: 195

loceff, fred points: 44

perry, fred points: 225

renquist, abe points: 148

smith, fred points: 95

stollings, pamela points: 452

zz-error, trevor points: 108

Sorting by first name ---------------

After:

renquist, abe points: 148

jacobs, carrie points: 195

loceff, fred points: 44

perry, fred points: 225

smith, fred points: 95

bauer, jack points: 123

cassar, john points: 321

stollings, pamela points: 452

charters, rodney points: 295

zz-error, trevor points: 108

Sorting by total points ---------------

After:

loceff, fred points: 44

smith, fred points: 95

zz-error, trevor points: 108

bauer, jack points: 123

renquist, abe points: 148

jacobs, carrie points: 195

perry, fred points: 225

charters, rodney points: 295

cassar, john points: 321

stollings, pamela points: 452

Median of evenClass = 171.5

Successfully preserved sort key.

Median of oddClass = 148.0

Median of smallClass = 95.0

Median of noClass = 0.0

**Answer**

* text/plain[foothillAssignment9.txt](https://myetudes.org/access/mneme/content/private/mneme/cff3240c-b51c-41f6-80dc-4db4530bdd05/submissions/15321977/4317663f-9f86-4b0e-80eb-38840c8c228f/foothillAssignment9.txt)

[[https://myetudes.org/ambrosia_library/icons/collapse.gif](https://myetudes.org/portal/tool/09d2d876-2329-4a14-000d-b3da1e731165/review/15321977/list) Model Answer](https://myetudes.org/portal/tool/09d2d876-2329-4a14-000d-b3da1e731165/review/15321977/list)

//CS 1A LOCEFF

// LAB 9 INSTRUCTOR SOLUTION

public class Foothill

{

   public static void main (String[] args)

   {

      Student[] evenClass = { new Student("smith","fred", 95),

         new Student("bauer","jack",123),

         new Student("jacobs","carrie", 195),

         new Student("renquist","abe",148),

         new Student("3ackson","trevor", 108),

         new Student("perry","fred",225),

         new Student("loceff","fred", 44),

         new Student("stollings","pamela",452),

         new Student("charters","rodney", 295),

         new Student("cassar","john",321),

      };

      Student[] oddClass = { new Student("smith","fred", 95),

         new Student("bauer","jack",123),

         new Student("jacobs","carrie", 195),

         new Student("renquist","abe",148),

         new Student("3ackson","trevor", 108),

         new Student("perry","fred",225),

         new Student("loceff","fred", 44),

         new Student("stollings","pamela",452),

         new Student("charters","rodney", 295),

         new Student("cassar","john",321),

         new Student("oddman","out",100),

      };

      Student[] smallClass = { new Student("smith","fred", 95) };

      Student[] noClass = { };

      System.out.println( StudentArrayUtilities.toString("Before: ", evenClass));

      System.out.println("Sorting by default ---------------");

      StudentArrayUtilities.arraySort(evenClass);

      System.out.println( StudentArrayUtilities.toString("After: ", evenClass));

      System.out.println("Sorting by first name ---------------");

      Student.setSortKey(Student.SORT\_BY\_FIRST);

      StudentArrayUtilities.arraySort(evenClass);

      System.out.println( StudentArrayUtilities.toString("After: ", evenClass));

      System.out.println("Sorting by total points ---------------");

      Student.setSortKey(Student.SORT\_BY\_POINTS);

      StudentArrayUtilities.arraySort(evenClass);

      System.out.println( StudentArrayUtilities.toString("After: ", evenClass));

      // test median

      Student.setSortKey(Student.SORT\_BY\_LAST);

      System.out.println( "Median of evenClass = "

         + StudentArrayUtilities.getMedianDestructive(evenClass));

      if ( Student.getSortKey() == Student.SORT\_BY\_LAST )

         System.out.println("Successfully preserved sort key.");

      else

         System.out.println("Problem.");

      // test odd class

      System.out.println( "Median of oddClass = "

         + StudentArrayUtilities.getMedianDestructive(oddClass));

      // test one-student class

      System.out.println( "Median of smallClass = "

         + StudentArrayUtilities.getMedianDestructive(smallClass));

      // test no-student class

      System.out.println( "Median of noClass = "

         + StudentArrayUtilities.getMedianDestructive(noClass));

   }

}

class Student

{

   private String lastName;

   private String firstName;

   private int totalPoints;

   public static final String DEFAULT\_NAME = "zz-error";

   public static final int DEFAULT\_POINTS = 0;

   public static final int MAX\_POINTS = 1000;

   // constructor requires parameters - no default supplied

   public Student( String last, String first, int points)

   {

      if ( !setLastName(last) )

         lastName = DEFAULT\_NAME;

      if ( !setFirstName(first) )

         firstName = DEFAULT\_NAME;

      if ( !setPoints(points) )

         totalPoints = DEFAULT\_POINTS;

   }

   public String getLastName() { return lastName; }

   public String getFirstName() { return firstName; }

   public int getTotalPoints() { return totalPoints; }

   public boolean setLastName(String last)

   {

      if ( !validString(last) )

         return false;

      lastName = last;

      return true;

   }

   public boolean setFirstName(String first)

   {

      if ( !validString(first) )

         return false;

      firstName = first;

      return true;

   }

   public boolean setPoints(int pts)

   {

      if ( !validPoints(pts) )

         return false;

      totalPoints = pts;

      return true;

   }

   // sort and ordering support

   static public final int SORT\_BY\_FIRST = 88;

   static public final int SORT\_BY\_LAST = 98;

   static public final int SORT\_BY\_POINTS = 108;

   static private int sortKey = SORT\_BY\_LAST;

   static boolean setSortKey( int key )

   {

      switch (key)

      {

      case SORT\_BY\_FIRST:

      case SORT\_BY\_LAST:

      case SORT\_BY\_POINTS:

         sortKey = key;

         return true;

      default:

        // should never get here, but compiler wants it here or after switch

         return false;

      }

   }

   static public int getSortKey()

   {

      return sortKey;

   }

   // could be an instance method and, if so, would take one parameter

   public static int compareTwoStudents( Student firstStud, Student secondStud )

   {

      switch (sortKey)

      {

      case SORT\_BY\_FIRST:

         return firstStud.firstName.compareToIgnoreCase(secondStud.firstName);

      case SORT\_BY\_LAST:

         return firstStud.lastName.compareToIgnoreCase(secondStud.lastName);

      case SORT\_BY\_POINTS:

         return firstStud.totalPoints - secondStud.totalPoints;

      default:

         return 0;

      }

   }

   public String toString()

   {

      String resultString;

      resultString = " " + lastName

         + ", " + firstName

         + " points: " + totalPoints

         + "\n";

      return resultString;

   }

   private static boolean validString( String testStr )

   {

      if (testStr != null && Character.isLetter(testStr.charAt(0)))

         return true;

      return false;

   }

   private static boolean validPoints( int testPoints )

   {

      if (testPoints >= 0 && testPoints <= MAX\_POINTS)

         return true;

      return false;

   }

}

class StudentArrayUtilities

{

   // return a string of entire array with a title for the console

   public static String toString(String title, Student[] data)

   {

      String output = title + "\n";

      // build the output string from the individual Students:

      for (int k = 0; k < data.length; k++)

         output += " " + data[k].toString();

      // now put it in a JOptionPane

      return output;

   }

   // returns true if a modification was made to the array

   private static boolean floatLargestToTop(Student[] data, int top)

   {

      boolean changed = false;

      Student temp;

      // compare with client call to see where the loop stops

      for (int k = 0; k < top; k++)

         if ( Student.compareTwoStudents(data[k], data[k+1]) > 0 )

         {

            temp = data[k];

            data[k] = data[k+1];

            data[k+1] = temp;

            changed = true;

         }

      return changed;

   }

   // public callable arraySort() - assumes Student class has a compareTo()

   public static void arraySort(Student[] array)

   {

      for (int k = 0; k < array.length; k++)

         // compare with method def to see where inner loop stops

         if ( !floatLargestToTop(array, array.length - 1 - k) )

            return;

   }

   // median -- this method is allowed to return with array in new order

   public static double getMedianDestructive(Student[] array)

   {

      int saveSortKey, arrLen, justBelow, justAbove;

      double retVal;

      arrLen = array.length;

      retVal = 0.;

      if (arrLen ==  0)

         return 0;

      if (arrLen ==  1)

         return array[0].getTotalPoints();

      // preserve the client's sortKey

      saveSortKey = Student.getSortKey();

      // sort by points to prep for computing median

      Student.setSortKey( Student.SORT\_BY\_POINTS );

      arraySort( array );

      if ( arrLen % 2 == 0 )

      {

         justBelow = (arrLen/2 - 1);

         justAbove = arrLen/2;

         retVal = ( array[justBelow].getTotalPoints()

            +  array[justAbove].getTotalPoints() ) / 2.0;  //need double

      }

      else

         retVal = array[arrLen/2].getTotalPoints();

      // restore the sort key to what it was before the call

      Student.setSortKey( saveSortKey );

      return retVal;

   }

}

/\* ------------------ SAMPLE RUN ------------------

Before:

  smith, fred points: 95

  bauer, jack points: 123

  jacobs, carrie points: 195

  renquist, abe points: 148

  zz-error, trevor points: 108

  perry, fred points: 225

  loceff, fred points: 44

  stollings, pamela points: 452

  charters, rodney points: 295

  cassar, john points: 321

Sorting by default ---------------

After:

  bauer, jack points: 123

  cassar, john points: 321

  charters, rodney points: 295

  jacobs, carrie points: 195

  loceff, fred points: 44

  perry, fred points: 225

  renquist, abe points: 148

  smith, fred points: 95

  stollings, pamela points: 452

  zz-error, trevor points: 108

Sorting by first name ---------------

After:

  renquist, abe points: 148

  jacobs, carrie points: 195

  loceff, fred points: 44

  perry, fred points: 225

  smith, fred points: 95

  bauer, jack points: 123

  cassar, john points: 321

  stollings, pamela points: 452

  charters, rodney points: 295

  zz-error, trevor points: 108

Sorting by total points ---------------

After:

  loceff, fred points: 44

  smith, fred points: 95

  zz-error, trevor points: 108

  bauer, jack points: 123

  renquist, abe points: 148

  jacobs, carrie points: 195

  perry, fred points: 225

  charters, rodney points: 295

  cassar, john points: 321

  stollings, pamela points: 452

Median of evenClass = 171.5

Successfully preserved sort key.

Median of oddClass = 148.0

Median of smallClass = 95.0

Median of noClass = 0.0

---------------------------------------------- \*/

**Comments**

So, we're nearing the end of the course.  I know you put a lot of work in, so let's review the last regular assignment here.  If you want to also do #10, that's optional.  No penalty for not doing it.

- The declarations of your arrays are inconsistent. They should look identical. (-.5)  
  
class Student ================================  
- look at a slightly more informative and efficient way to do the totalPoints case.  
\* You correctly added the static consts SORT\_BY\_FIRST, etc.  Good.  
\* The mutable static sortKey was there. Excellent.  
\* setSortKey() was provided as a static, exactly as required.  
\* setSortKey() provided a mechanism for setting sortKey and also not doing so in case of a bad parameter.  Excellent.  
\* You have the getSortKey() accessor, as required.  Great.  
\* compareTwoStudents() now uses the sortKey and compares based on whatever that value was.  This was the main task and you did it well.  
  
getMedianDestructive() -----------------------------------------  
- Should handle the array size = 1 first so as not to take certain unnecessary actions (like saving and restoring the sort key, calling a sort() method on one element, etc.).  This is a general concept:  avoid doing certain calls or parts of the algorithm if you don't need them.  (-.25)  
\* saves the old sort key.  
\* does the sort on total points prior to the computation of median.  
\* differentiates between even and odd array sizes.  Good.  
\* gets the median properly as the middle number (or avg of 2 middle numbers).  Excellent  
\* restores the client sort key.  Very nice.  
  
MAIN AND RUN ================================  
\* followed the steps of the main()  
\* run looked good  
  
Great final project submission. Dmitri

2 days late (-2)

 1 of 1

public class Foothill

{

public static void main (String[] args)

{

Student[] myClass = { new Student("smith","fred", 95),

new Student("bauer","jack",123),

new Student("jacobs","carrie", 195),

new Student("renquist","abe",148),

new Student("3ackson","trevor", 108),

new Student("perry","fred",225),

new Student("loceff","fred", 44),

new Student("stollings","pamela",452),

new Student("charters","rodney", 295),

new Student("cassar","john",321),

new Student("smith","bob",228),

new Student("ivanov","ivan", 55),

new Student("petrov","petr",542),

new Student("sidorov","sidr", 324),

new Student("marisko","marusya",32),

new Student("ivanko","vasya",32),

};

Student[] myClass1 = {new Student("smith","fred", 95),

new Student("bauer","jack",123),

new Student("jacobs","carrie", 195),

new Student("renquist","abe",148),

new Student("3ackson","trevor", 108),

new Student("perry","fred",225),

new Student("loceff","fred", 44),

new Student("stollings","pamela",452),

new Student("charters","rodney", 295),

new Student("cassar","john",321),

new Student("smith","bob",228),

new Student("ivanov","ivan", 55),

new Student("petrov","petr",542),

new Student("sidorov","sidr", 324),

new Student("marisko","marusya",33),

};

Student[] myClass2 = {new Student("smith","fred", 95)};

Student[] myClass3 = null;

System.out.println(StudentArrayUtilities.toString("Before:", myClass));

StudentArrayUtilities.arraySort(myClass);

System.out.println("Sorting by default ---------------");

System.out.println(StudentArrayUtilities.toString("After:", myClass));

Student.setSortKey(Student.SORT\_BY\_FIRST);

StudentArrayUtilities.arraySort(myClass);

System.out.println("Sorting by first name ---------------");

System.out.println(StudentArrayUtilities.toString("After:", myClass));

Student.setSortKey(Student.SORT\_BY\_POINTS);

StudentArrayUtilities.arraySort(myClass);

System.out.println("Sorting by total points -------------");

System.out.println(StudentArrayUtilities.toString("After:", myClass));

Student.setSortKey(Student.SORT\_BY\_FIRST);

double median = StudentArrayUtilities.getMedianDestructive(myClass);

System.out.println("Median of evenClass = " + median);

if (Student.getSortKey() == Student.SORT\_BY\_FIRST)

System.out.println("Successfully preserved sort key.");

median = StudentArrayUtilities.getMedianDestructive(myClass1);

System.out.println("Median of oddClass = " + median);

median = StudentArrayUtilities.getMedianDestructive(myClass2);

System.out.println("Median of smallClass = " + median);

median = StudentArrayUtilities.getMedianDestructive(myClass3);

System.out.println("Median of noClass = " + median);

}

}

class Student

{

private String lastName;

private String firstName;

private int totalPoints;

public static final int SORT\_BY\_FIRST = 88;

public static final int SORT\_BY\_LAST = 98;

public static final int SORT\_BY\_POINTS = 108;

public static final String DEFAULT\_NAME = "zz-error";

public static final int DEFAULT\_POINTS = 0;

public static final int MAX\_POINTS = 1000;

private static int sortKey = SORT\_BY\_LAST;

public Student( String last, String first, int points)

{

if ( !setLastName(last) )

lastName = DEFAULT\_NAME;

if ( !setFirstName(first) )

firstName = DEFAULT\_NAME;

if ( !setPoints(points) )

totalPoints = DEFAULT\_POINTS;

}

public String getLastName()

{

return lastName;

}

public String getFirstName()

{

return firstName;

}

public int getTotalPoints()

{

return totalPoints;

}

public static int getSortKey()

{

return sortKey;

}

public static boolean setSortKey(int key)

{

if (!validSortKey(key))

return false;

sortKey = key;

return true;

}

public boolean setLastName(String last)

{

if ( !validString(last) )

return false;

lastName = last;

return true;

}

public boolean setFirstName(String first)

{

if ( !validString(first) )

return false;

firstName = first;

return true;

}

public boolean setPoints(int pts)

{

if ( !validPoints(pts) )

return false;

totalPoints = pts;

return true;

}

static int compareTwoStudents( Student frstStud, Student scndStud )

{

int sortKey = getSortKey();

int result = 0;

switch(sortKey)

{

case SORT\_BY\_FIRST:

result = frstStud.firstName.compareToIgnoreCase(scndStud.firstName);

break;

case SORT\_BY\_LAST:

result = frstStud.lastName.compareToIgnoreCase(scndStud.lastName);

break;

case SORT\_BY\_POINTS:

if (frstStud.getTotalPoints() > scndStud.getTotalPoints())

result = 1;

else if (frstStud.getTotalPoints() == scndStud.getTotalPoints())

result = 0;

else

result = -1;

break;

default:

result = 0;

break;

}

return result;

}

public String toString()

{

String resultString;

resultString = " "+ lastName

+ ", " + firstName

+ " points: " + totalPoints

+ "\n";

return resultString;

}

private static boolean validSortKey(int key)

{

if(key == SORT\_BY\_FIRST || key == SORT\_BY\_LAST || key == SORT\_BY\_POINTS)

return true;

return false;

}

private static boolean validString(String testStr)

{

if (testStr != null && Character.isLetter(testStr.charAt(0)))

return true;

return false;

}

private static boolean validPoints(int testPoints)

{

if (testPoints >= 0 && testPoints <= MAX\_POINTS)

return true;

return false;

}

}

class StudentArrayUtilities

{

public static String toString(String title, Student[] data)

{

String output = title + "\n";

for (int k = 0; k < data.length; k++)

output += " "+ data[k].toString();

return output;

}

public static double getMedianDestructive(Student[] array)

{

int clientSortKey = Student.getSortKey();

Student.setSortKey(Student.SORT\_BY\_POINTS);

double median = 0;

if (array != null)

{

arraySort(array);

if (array.length == 1)

median = (double) array[0].getTotalPoints();

else

{

if (array.length % 2 == 0)

median = ((double)array[array.length/2].getTotalPoints()

+ (double)array[array.length/2 - 1].getTotalPoints())/2;

else

median = (double) array[array.length/2].getTotalPoints();

}

}

Student.setSortKey(clientSortKey);

return median;

}

private static boolean floatLargestToTop(Student[] data, int top)

{

boolean changed = false;

Student temp;

for (int k = 0; k < top; k++)

if (Student.compareTwoStudents(data[k], data[k+1]) > 0)

{

temp = data[k];

data[k] = data[k+1];

data[k+1] = temp;

changed = true;

}

return changed;

}

public static void arraySort(Student[] array)

{

for (int k = 0; k < array.length; k++)

if (!floatLargestToTop(array, array.length-1-k))

return;

}

}

/\*----------paste of run from console window------------

Before:

smith, fred points: 95

bauer, jack points: 123

jacobs, carrie points: 195

renquist, abe points: 148

zz-error, trevor points: 108

perry, fred points: 225

loceff, fred points: 44

stollings, pamela points: 452

charters, rodney points: 295

cassar, john points: 321

smith, bob points: 228

ivanov, ivan points: 55

petrov, petr points: 542

sidorov, sidr points: 324

marisko, marusya points: 32

ivanko, vasya points: 32

Sorting by default ---------------

After:

bauer, jack points: 123

cassar, john points: 321

charters, rodney points: 295

ivanko, vasya points: 32

ivanov, ivan points: 55

jacobs, carrie points: 195

loceff, fred points: 44

marisko, marusya points: 32

perry, fred points: 225

petrov, petr points: 542

renquist, abe points: 148

sidorov, sidr points: 324

smith, fred points: 95

smith, bob points: 228

stollings, pamela points: 452

zz-error, trevor points: 108

Sorting by first name ---------------

After:

renquist, abe points: 148

smith, bob points: 228

jacobs, carrie points: 195

loceff, fred points: 44

perry, fred points: 225

smith, fred points: 95

ivanov, ivan points: 55

bauer, jack points: 123

cassar, john points: 321

marisko, marusya points: 32

stollings, pamela points: 452

petrov, petr points: 542

charters, rodney points: 295

sidorov, sidr points: 324

zz-error, trevor points: 108

ivanko, vasya points: 32

Sorting by total points -------------

After:

marisko, marusya points: 32

ivanko, vasya points: 32

loceff, fred points: 44

ivanov, ivan points: 55

smith, fred points: 95

zz-error, trevor points: 108

bauer, jack points: 123

renquist, abe points: 148

jacobs, carrie points: 195

perry, fred points: 225

smith, bob points: 228

charters, rodney points: 295

cassar, john points: 321

sidorov, sidr points: 324

stollings, pamela points: 452

petrov, petr points: 542

Median of evenClass = 171.5

Successfully preserved sort key.

Median of oddClass = 195.0

Median of smallClass = 95.0

Median of noClass = 0.0

----------------------------------------------------- \*/