**Reviewing Assignment**

Lab Assignment 10

|  |  |
| --- | --- |
| Started: | Dec 2, 2014 6:18 PM |
| Finished: | Dec 4, 2014 11:32 PM - late |

 1 of 1

**Lab Assignment 10     Total Grade: 10   (of possible 0.1 points)**

**Score: 10   (of possible 0.1 points)**

**Assignment 10 - A StudentArrayUtilities Class with an Internal Array**

*This extra credit assignment is worth 10 points.  Don't worry about the fact that it may say " .1 points possible."*

OPTION A (The Only Option):  A List of Points

Understand the Problem

Our previous **StudentArrayUtilities** (**SAU**) class consisted of a set of static class methods that acted on arrays of students.  We take this a step further.  Rather than having our client define and manage the array, let's give **SAU** the ability to do that.  There will be two main changes required:

* **Adding and Removing Students** - We will provide methods **addStudent()** and **removeStudent()** as***instance methods*** in our **SAU** class.  Our clients can use these to add or remove individual students to an**SAU** object.  The client may not see where or how the students are stored, internally.  The extent to which we allow the client to gain access to the internal data is up to us, and in this assignment, we will give the client very limited access:  our implementation will behave a bit like a stack data structure, so that adding is like "pushing" and removing is like (but not exactly the same as) "popping".
* **Earlier Methods Become Instance Methods** - Once we have the data ***internal*** to our **SAU** object, there is no need to pass arrays;  the data is already present as***this*** data.  So older methods, like **sort()** and **toString()**, are converted to***instance methods***, and their signatures are changed to reflect this fact.  When a sort is requested, the client only has to call **someUtilObj.sort()**, and it is understood that the **Students** to be sorted are the ones in the ***this***  object's internal array created by the many previous calls to **addStudent()** and, perhaps, trimmed by some calls to **removeStudent**().

StudentArrayUtilities Spec

Summary

The class will store its ***private data*** as***an array of Student*** references.   The natural -- and in a sense *universal* -- solution to this arrangement is to have two private data members: one for the ***array***, and a second for the **int** which holds us the number of actual students being stored in the array at any point in time:

Student[] theArray;

int numStudents;

**theArray** will be a fixed-size array, that size being the maximum number of **Students**we expect to ever manage.  It will hold many more elements than we need, typically.  **numStudents** would initially be **0** (during an object instantiation of this class), would grow as students are added via **addStudent()** and shrink as **Students**are removed via **removeStudent()**.  A common use for **numStudents**, besides telling us how many actual **Students**are in the  object, is to tell any method that cares where the last (i.e., *highest* in the array) **Student**is stored.  So, if **numStudents** is **12**, it means there are **12** **Students**in the array, even if the **theArray** happens to have, say,**1000 = MAX\_STUDENTS** positions of available capacity.   The active **Students** are stored in locations**0 - 11**, **theArray[11]** is the location of the **Student**in the highest occupied position, while **theArray[12]** is where the next **Student**would be added if and when a subsequent call was made to **addStudent()**.

The client may not need to know all this, but that's what's going on, internally.

An**SAU** object is instantiated using a default constructor and, once created, will use the **addStudent()** mutator to build a **Student**roster for the object.  If the client wants to reduce the population of that object it would call **removeStudent()**.  Here is a typical instantiation followed by the addition of a couple **Student**s:

StudentArrayUtilities myStuds = new StudentArrayUtilities();

myStuds.addStudent( new Student("bartman", "petra", 102) );

myStuds.addStudent( new Student("charters","rodney", 295));

As you see, there is no need for the client to create any arrays.  However, if the client happens to have an array of **Students**, it can use that array along with a loop to add many students efficiently, as in:

for (k = 0; k < myClass.length; k++)

myStuds.addStudent( myClass[k] );

Once there are some **Student**s in the **SAU** object, we can display them with the help of the instance method**toString()**:

System.out.println( myStuds.toString("Here are the students currently being stored: "));

This is a method similar to the**toString()** of our older **SAU** class, but now we see no array need be passed, since this is an instance method which carries the full data of the object wherever it goes (via the ***this*** data).  Likewise, the sort is handled without a parameter:

myStuds.arraySort();

That method call would result in our internal array being re-ordered.  As before, and without any modifications needed, that sort would be based on the underlying **Student**class's ***sort key***.

Calling **toString()** after an**arraySort()** is invoked would naturally show the new order of the internal array.

Finally, the**removeStudent()** method would remove (from the internal array inside its **SAU** object) one of the **Student**s.  Which **Student**?  That's up to this spec and we could have many different answers.  The answer for *us* (and for *you*) is that it will remove and return the student in the***highest occupied array location*** of the internal array.  So a call to:

student = myStuds.removeStudent();

Would remove the **Student** in location which is determined by the current value of the private member (but not necessarily that exact location -- this is something you will determine based on all the information just given.  It copies that removed object to the **Student**variable on the LHS for use by the client.

Static (Public) Members

* A final int **MAX\_STUDENTS** which you can set to 20 for testing, but would be larger in general.  This is used to instantiate the internal array, whose capacity (physical array size) never changes from this one value.

Instance (Private) Members

* **Student[] theArray** - our internal array whose size is always **MAX\_STUDENTS**, but whose actual data is stored in elements **0** through **numStudents - 1**.
* **int numStudents** - the current number of actual students stored in the array.  This can never be > **MAX\_STUDENTS**, and you have to make sure that it isn't.

Instance (Public) Methods

* **boolean addStudent( Student stud )**- This method will place the passed-parameter into the next available location (highest) of our internal array.  It has to test**stud** for **null** and also make sure not to overrun the internal array by breaching its capacity, **MAX\_STUDENTS**.  Returns false if error.
* **Student removeStudent()**- Removes and returns the Student in the highest occupied position in the array.  Note, depending on whether or not the internal array has been modified by a sort, this may -- but does not have to be -- the most recently added **Student**.  If there are no students in the array, it returns **null**.  
    
  The following methods are still here, but are now changed to instance methods.  The information that was supplied by their array parameters is now provided as part of the ***this*** object in the form of **numStudents** (for the array size) and **theArray** (for the hitherto passed array).  You would now use those members in their definitions.
* **String toString( String title )**- This returns, usually for display by client, our the entire array in a single **String**, just like the static version did, but without the need for an array parameter.
* **void arraySort()**- Just like the old static ***sort***, but no need for an array parameter.   It works on the internal array.
* **double getMedianDestructive()** - as above.

Helper (Private) Methods

* **boolean floatLargestToTop( int top )** - Same as our old version, but now ***instance***.  Use this data in place of parameters lost from prior version.

Sample Client

Here is some client code to use while debugging.  You should be able to determine the correct run that results.  You should provide some code that is more complete than this in your testing.   For this test (and your own testing) set **MAX\_STUDENTS** to **20**.

public static void main (String[] args)

{

int k;

Student student;

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),

};

// instantiate a StudArrUtilObject

StudentArrayUtilities myStuds = new StudentArrayUtilities();

// we can add stdunts manually and individually

myStuds.addStudent( new Student("bartman", "petra", 102) );

myStuds.addStudent( new Student("charters","rodney", 295));

// if we happen to have an array available, we can add students in loop.

for (k = 0; k < myClass.length; k++)

myStuds.addStudent( myClass[k] );

System.out.println( myStuds.toString("Before: "));

myStuds.arraySort();

System.out.println( myStuds.toString("Sorting by default: "));

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

myStuds.arraySort();

System.out.println( myStuds.toString("Sorting by first name: "));

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

myStuds.arraySort();

System.out.println( myStuds.toString("Sorting by total points: "));

// test median

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

+ myStuds.getMedianDestructive() + "\n");

// various tests of removing and adding too many students

for (k = 0; k < 100; k++)

{

if ( (student = myStuds.removeStudent()) != null)

System.out.println("Removed " + student);

else

{

System.out.println("Empty after " + k + " removes.");

break;

}

}

for (k = 0; k < 100; k++)

{

if (!myStuds.addStudent(new Student("first", "last", 22)))

{

System.out.println("Full after " + k + " adds.");

break;

}

}

}

*That's all.  See you at the Final Exam, and have a wonderful break, next quarter, and life.*

**Answer**

* text/plain[foothillAssignment10.txt](https://myetudes.org/access/mneme/content/private/mneme/cff3240c-b51c-41f6-80dc-4db4530bdd05/submissions/15397178/e3beecba-4a2e-4e72-804b-7cde40096b8d/foothillAssignment10.txt)

[[https://myetudes.org/ambrosia_library/icons/collapse.gif](https://myetudes.org/portal/tool/09d2d876-2329-4a14-000d-b3da1e731165/review/15397178/list) Model Answer](https://myetudes.org/portal/tool/09d2d876-2329-4a14-000d-b3da1e731165/review/15397178/list)

//CS 1A LOCEFF

// LAB 10 INSTRUCTOR SOLUTION

public class Foothill

{

   public static void main (String[] args)

   {

      int k;

      Student student;

      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),

      };

      // instantiate a StudArrUtilObject

      StudentArrayUtilities myStuds = new StudentArrayUtilities();

      // we can add stdunts manually and individually

      myStuds.addStudent( new Student("bartman", "petra", 102) );

      myStuds.addStudent( new Student("charters","rodney", 295));

      // if we happen to have an array available, we can add students in loop.

      for (k = 0; k < myClass.length; k++)

         myStuds.addStudent( myClass[k] );

      System.out.println( myStuds.toString("Before: "));

      myStuds.arraySort();

      System.out.println( myStuds.toString("Sorting by default: "));

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

      myStuds.arraySort();

      System.out.println( myStuds.toString("Sorting by first name: "));

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

      myStuds.arraySort();

      System.out.println( myStuds.toString("Sorting by total points: "));

      // test median

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

         +  myStuds.getMedianDestructive() + "\n");

      // various tests of removing and adding too many students

      for (k = 0; k < 100; k++)

      {

         if ( (student = myStuds.removeStudent()) != null)

            System.out.println("Removed " + student);

         else

         {

            System.out.println("Empty after " + k + " removes.");

            break;

         }

      }

      for (k = 0; k < 100; k++)

      {

         if (!myStuds.addStudent(new Student("first", "last", 22)))

         {

            System.out.println("Full after " + k + " adds.");

            break;

         }

      }

   }

}

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:

         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

{

   public final int MAX\_STUDENTS = 20;

   private Student[] theArray;

   private int numStudents = 0;

   // one and only constructor

   StudentArrayUtilities()

   {

      theArray = new Student[MAX\_STUDENTS];

      numStudents = 0;

   }

   public boolean addStudent( Student stud )

   {

      if (stud == null || numStudents >= MAX\_STUDENTS )

         return false;

      theArray[ numStudents++ ] = stud;

      return true;

   }

   public Student removeStudent()

   {

      if (numStudents == 0)

         return null;

      return  theArray[--numStudents] ;

   }

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

   public String toString(String title)

   {

      String output = title + "\n";

      // build the output string from the individual Students:

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

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

      // now put it in a JOptionPane

      return output;

   }

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

   private  boolean floatLargestToTop(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(theArray[k], theArray[k+1]) > 0 )

         {

            temp = theArray[k];

            theArray[k] = theArray[k+1];

            theArray[k+1] = temp;

            changed = true;

         }

      return changed;

   }

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

   public  void arraySort()

   {

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

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

         if ( !floatLargestToTop( numStudents - 1 - k) )

            return;

   }

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

   public double getMedianDestructive()

   {

      int saveSortKey, justBelow, justAbove;

      double retVal;

      retVal = 0.;

      if (numStudents ==  0)

         return 0;

      if (numStudents ==  1)

         return theArray[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();

      if ( numStudents % 2 == 0 )

      {

         justBelow = (numStudents/2 - 1);

         justAbove = numStudents/2;

         retVal = ( theArray[justBelow].getTotalPoints()

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

      }

      else

         retVal = theArray[numStudents/2].getTotalPoints();

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

      Student.setSortKey( saveSortKey );

      return retVal;

   }

}

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

Before:

  bartman, petra points: 102

  charters, rodney points: 295

  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:

  bartman, petra points: 102

  bauer, jack points: 123

  cassar, john points: 321

  charters, rodney points: 295

  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:

  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

  bartman, petra points: 102

  charters, rodney points: 295

  charters, rodney points: 295

  zz-error, trevor points: 108

Sorting by total points:

  loceff, fred points: 44

  smith, fred points: 95

  bartman, petra points: 102

  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

  charters, rodney points: 295

  cassar, john points: 321

  stollings, pamela points: 452

Median of evenClass = 171.5

Removed  stollings, pamela points: 452

Removed  cassar, john points: 321

Removed  charters, rodney points: 295

Removed  charters, rodney points: 295

Removed  perry, fred points: 225

Removed  jacobs, carrie points: 195

Removed  renquist, abe points: 148

Removed  bauer, jack points: 123

Removed  zz-error, trevor points: 108

Removed  bartman, petra points: 102

Removed  smith, fred points: 95

Removed  loceff, fred points: 44

Empty after 12 removes.

Full after 20 adds.

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

**Comments**

I'm really glad to see that you did the bonus -- it's a good way to practice for the final.  Please compare with my instructor solution to complete the process of understanding this useful SAU class.  
  
class SAU ================================  
\* You added the static const MAX\_STUDENTS, as needed.  
\* Yours had the instance members theArray[] and numStudents.  Great.  
\* Your correctly modified the old static methods like sort() and toString() and made them instance methods.  
\* You properly removed the parameters to the old static methods and now used the implied "this" object in their place. Great job there.  
\* addStudent() tested to make sure numStudents was not too large (i.e., was not >= MAX\_STUDENTS, the size of the member array).  Great work there.  
\* removeStudent() - I'll let you self-grade that by comparing to my solution.

Fantastic! Enjoy your holidays, Dmitri

 1 of 1

public class Foothill

{

public static void main(String[] args)

{

int k;

Student student = null;

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),

};

StudentArrayUtilities myStuds = new StudentArrayUtilities();

myStuds.addStudent(new Student("bartman", "petra", 102));

myStuds.addStudent(new Student("charters", "rodney", 295));

for (k = 0; k < myClass.length; k++)

myStuds.addStudent(myClass[k]);

System.out.println(myStuds.toString("Before: "));

myStuds.arraySort();

System.out.println(myStuds.toString("Sorting by default: "));

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

myStuds.arraySort();

System.out.println(myStuds.toString("Sorting by first name: "));

Student Removed = myStuds.removeStudent();

System.out.println("Removed " + Removed.toString());

System.out.println( myStuds.toString("After Removal"));

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

myStuds.arraySort();

System.out.println(myStuds.toString("Sorting by total points: "));

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

+ myStuds.getMedianDestructive() + "\n");

for (k = 0; k < 100; k++)

{

if ((student = myStuds.removeStudent()) != null)

System.out.println("Removed " + student.toString());

else

{

System.out.println("Empty after " + k + " removes.");

break;

}

}

for (k = 0; k < 100; k++)

{

if (!myStuds.addStudent(new Student("first", "last", 22)))

{

System.out.println("Full after " + k + " adds.");

break;

}

}

}

}

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:

result = frstStud.totalPoints - scndStud.totalPoints;

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

{

private static int MAX\_STUDENTS = 20;

private Student[] theArray = null;

private int numStudents = 0;

public StudentArrayUtilities()

{

theArray = new Student[MAX\_STUDENTS];

}

public boolean addStudent(Student stud)

{

if (stud != null && numStudents < MAX\_STUDENTS)

{

theArray[numStudents] = stud;

numStudents++;

return true;

}

return false;

}

public Student removeStudent()

{

Student stud = null;

if (numStudents >= 1)

{

stud = theArray[numStudents - 1];

numStudents--;

}

return stud;

}

public String toString(String title)

{

String output = title + "\n";

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

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

return output;

}

public double getMedianDestructive()

{

double median = 0;

if (theArray != null && numStudents > 0)

{

if (numStudents == 1)

median = theArray[0].getTotalPoints();

else

{

int clientSortKey = Student.getSortKey();

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

arraySort();

if (numStudents % 2 == 0)

median = ((double) theArray[numStudents / 2].getTotalPoints()

+ (double) theArray[numStudents / 2 - 1]

.getTotalPoints()) / 2;

else

median = theArray[numStudents / 2].getTotalPoints();

Student.setSortKey(clientSortKey);

}

}

return median;

}

private boolean floatLargestToTop(int top)

{

boolean changed = false;

Student temp;

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

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

{

temp = theArray[k];

theArray[k] = theArray[k + 1];

theArray[k + 1] = temp;

changed = true;

}

return changed;

}

public void arraySort()

{

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

if (!floatLargestToTop(numStudents - 1 - k))

return;

}

}

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

Before:

bartman, petra points: 102

charters, rodney points: 295

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:

bartman, petra points: 102

bauer, jack points: 123

cassar, john points: 321

charters, rodney points: 295

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:

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

bartman, petra points: 102

charters, rodney points: 295

charters, rodney points: 295

zz-error, trevor points: 108

Removed zz-error, trevor points: 108

After Removal

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

bartman, petra points: 102

charters, rodney points: 295

charters, rodney points: 295

Sorting by total points:

loceff, fred points: 44

smith, fred points: 95

bartman, petra points: 102

bauer, jack points: 123

renquist, abe points: 148

jacobs, carrie points: 195

perry, fred points: 225

charters, rodney points: 295

charters, rodney points: 295

cassar, john points: 321

stollings, pamela points: 452

Median of evenClass = 195.0

Removed stollings, pamela points: 452

Removed cassar, john points: 321

Removed charters, rodney points: 295

Removed charters, rodney points: 295

Removed perry, fred points: 225

Removed jacobs, carrie points: 195

Removed renquist, abe points: 148

Removed bauer, jack points: 123

Removed bartman, petra points: 102

Removed smith, fred points: 95

Removed loceff, fred points: 44

Empty after 11 removes.

Full after 20 adds.

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