Programming Assignment 9: Arrays
Total Points (50 pts) - Due Wednesday, March 22nd at 11:59 PM

This programming assignment is intended to demonstrate your knowledge of the following:

- declare a new class
- use calls to instance methods to access and change the state of an object
- declare and instantiate arrays
- access and process data in an array

A StudentArrayUtilities Class with an Internal Array

You will start with the existing **Student** class. You will modify the class as described below and provide a **StudentArrayUtilities** class.

Part 1: Additions to the Student Class [25 points]

Modification to the Student Class

We will add the following members to the Student class.

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 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. 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**.
- **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 like the one already in the modules (in fact one will be identical). 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**.

StudentArrayUtilities Class Spec

The class will store its **private data** as **an array of Student** references. The natural 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()**.

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. 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.

Part 2: StudentArrayUtilities Class [25 points]

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.
- **String toString(String title)** - Replace printArray() with toString(String title). This returns, usually for display by client, our the entire array in a single **String** 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()** - 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.
  - 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 totalPoints 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.

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 class StudentDriver{
   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 adding too many students
      for (k = 0; k < 100; k++)
      {
         if (!myStuds.addStudent(new Student("first", "last", 22)))
         {
            System.out.println("Full after " + k + " adds.");
            break;
         }
      }
   }
}
```

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
Full after 20 adds.
```

Submission Instructions

- Execute the program and copy/paste the output that is produced by your program into the bottom of the source code file, making it into a comment. I will run the programs myself to see the output.
- Make sure the run "matches" your source. If the run you submit could not have come from the source you submit, it will be graded as if you did not hand in a run.
- Use the Assignment Submission link to submit the source code file.
- Submit the following files:
  - Student.java
  - StudentArrayUtilities.java
  - StudentDriver.java
- Do not submit .class files.