Project 8: Sport Stacker App

Due: Monday, October 15, 2012 by 11:59 PM

Deliverables

The following project files must be submitted to the "graded" assignment by the due date and time specified above (see the Lab Guidelines for information on submitting project files). You should plan to start on the project not later than Wednesday in lab. To ensure that your classes and methods are named correctly, you should make sure that your file is successfully submitted to the "ungraded" assignment in Web-CAT during lab on Wednesday. **Projects sent via e-mail past the deadline at 11:59 PM will not be accepted without a university-approved excuse.**

Files to submit to Web-CAT:

- SportStackerApp.java
- SportStacker.java

Specifications - Use arrays in this project; ArrayLists are not allowed!

Overview: Sport Stacking consists of an individual or team stacking and un-stacking cups in pre-determined sequences, competing against the clock or another player. This project will create two classes: (1) SportStacker is a class representing a competition sport stacker and (2) SportStackerApp is a driver class with a main method that reads input from a file specified as a command line argument, creates a SportStacker object, then provides options for printing the report, adding a new time, deleting the worst time; finding the fastest time, and quitting the application. **I strongly recommend that you create a new folder for this project.** Here's a video that shows a sport stacker in action: http://www.speedstacks.com/videos/steven_purugganan_cycle_video-4.

- **SportStacker.java**

  Requirements: Create SportStacker class that stores the competitor's name, an array of competition times, and the number of times that are stored in the array. It also includes methods to get name, times, and number or time, and methods to print a report, add a time, delete the worst time, and find the fasted time.

  Design: The SportStacker class has fields, a constructor, and methods as outlined below.

  (1) **Fields** (or instance variables): competitor's name which is a String, an array of competition times of type double, and the number of times that are stored in the array. These instance variables should be private so that they are not directly accessible from outside of the class. These are the only instance variables this class should have.

  (2) **Constructor**: Your SportStacker class must contain a constructor that accepts three parameters representing the name, number of times, and an array of times. Below is an example of how the constructor could be used in main to create a SportStacker object:

```
SportStacker stacker = new SportStacker (name, numOfTimes, times);
```

(3) **Methods**: Usually a class provides methods to access (or read) and modify each of its instance variables (known as get and set methods) along with any other required methods. The methods for StudentInvoice are described below.

- `getName`: Accepts no parameters and returns a String representing the name.
- `getTimes`: Accepts no parameters and returns a double array representing times field.
- `getNumTimesRecorded`: Accepts no parameters and returns an int representing the number of times that are stored in the array.
- `toString`: Returns a String containing the information in the SportStacker object as shown below, including newline escape sequences and formatting for the average time.. Note that no lines are skipped.

```
Sport Stacker Name: Sami Smith
Times: 7.41 8.86 5.9 7.78 6.45 5.88 12.5 9.0
Average Time: 7.97
```

- `addTime`: Accepts a double parameter representing a new time to add to the times array; and returns nothing. Note that when an array element is added, it should be placed at the next available index indicated by the field for number of times recorded. If this will exceed the capacity of the array, you must call the increaseSize method described below before you add the new time to the times array.
- `increaseSize`: Accepts no parameters and has no return value, increases the capacity of the times array by one. You must create a new temp array that is one element larger, copy the old array to the temp array, and then assign the temp array to the old array. See the example on pages 398-401 in your text. Specifically, see the increaseSize() method on page 399. Note that this method doubles the size of the array; whereas your method should only increase the size by one.
- `removeSlowestTime`: Accepts no parameters, removes the slowest time (i.e., the largest value) in the times array, and returns a double representing the time that was removed. Note that when an array element is removed, the remaining elements must be shifted to the left as appropriate and the now unoccupied location at the end should be set to zero.
- `findFastestTime`: Accepts no parameters and returns a double representing the fastest time in the times array.
- `computeAvgTimes`: Accepts no parameters and returns a double representing the average of recorded values in the times array. If there are no times in array, 0.0 should be returned as the average.

Code and Test: As you implement your SportStacker class, you should compile it and then test it using interactions. For example, as soon you have implemented and successfully compiled the constructor, you should create an instance of SportStacker in interactions . Remember that when you have an instance on the workbench, you can unfold it to see its values. After you have implemented and compiled one or more methods, create a SportStacker object and invoke each of your methods on the object to make sure the methods is working as intended.

- **SportStackerApp.java**

Requirements: Create a SportStackerApp class with a main method that gets a file name as a run argument. To use run arguments, click the Build menu, then check "Run Arguments". The Run Arguments text box should open above your code. The program reads in the file and creates a SportStacker object. The program then presents a menu to the user with five options, each of which is implemented: (1) print a report, (2) add a new time, (3) delete the worst time, (4) find the fastest time, or (5) quit the program.

Design: If no file name is provided as a run argument, an appropriate message should be printed and the program should end (e.g., you can use a *return* statement with no value to exit the main method).

Line #	Program output
1	`File name expected as a run argument.`
2	`Program ending.`

One example input file is provided (*stacker.dat*), but you may want to create additional input files perhaps with fewer times for testing. After reading in the input file, the main method should print an appropriate success message followed by a list of options with the action code and a short description followed by a line with just the action codes prompting the user to select an action. After the user enters an action code, the action is performed, including output if any. Then the line with just the action codes prompting the user to select an action is printed again to accept the next code. Since the input is read in and the SportStacker is created prior to the menu, the first action a user should normally perform is 'P' to print the report and verify that the input file was read and processed correctly. The other actions can then be used as appropriate. This continues until the user enters 'Q' to quit. Note that your program should accept both uppercase and lowercase action codes.

Below is output produced after printing the action codes and short descriptions followed by the prompt with the action codes waiting for the user to make a selection.

Line #	Program output
1	`File read in and SportStacker created.`
2	`StudentInvoice System Menu`
3	`P - Print Report`
4	`A - Add A New Time`
5	`D - Delete Worst Time`
6	`F - Find Fastest Time`
7	`Q - Quit`
8	
9	`Enter Code [P, A, D, F, or Q]:`

The result of the user selecting 'p' to print a report is shown below. This is followed by the prompt with the action codes waiting for the user to make the next selection.

Line #	Program output
	File read in and SportStacker created. StudentInvoice System Menu P - Print Report A - Add A New Time D - Delete Worst Time F - Find Fastest Time Q - Quit Enter Code [P, A, D, F, or Q]: p Sport Stacker Name: Sami Smith Times: 7.41 8.86 5.9 7.78 6.45 5.88 12.5 9.0 Average Time: 7.97 Enter Code [P, A, D, F, or Q]:

The result of the user selecting 'a' to add a new time followed by 'p' to print the updated report is shown below. Note that after 'a' was entered, the user was prompted for the time to add, which is then added to the times array. This is followed by the prompt for the next action where p was entered. After printing the report, the prompt for the next action is displayed again.

Line #	Program output
	Enter Code [P, A, D, F, or Q]: a Time To Add: 6.99 Enter Code [P, A, D, F, or Q]: p Sport Stacker Name: Sami Smith Times: 7.41 8.86 5.9 7.78 6.45 5.88 12.5 9.0 6.99 Average Time: 7.86 Enter Code [P, A, D, F, or Q]:

Here is an example of deleting the three worst times by entering ‘d’ as three separate actions followed by ‘p’ to print the updated report.

Line #	Program output
	Enter Code [P, A, D, F, or Q]: d Slowest Time Deleted: 12.5 Enter Code [P, A, D, F, or Q]: d Slowest Time Deleted: 9.0 Enter Code [P, A, D, F, or Q]: d Slowest Time Deleted: 8.86 Enter Code [P, A, D, F, or Q]: p Sport Stacker Name: Sami Smith Times: 7.41 5.9 7.78 6.45 5.88 6.99 Average Time: 6.74 Enter Code [P, A, D, F, or Q]:

Here is an example of a ‘p’ to print the report followed by ‘f’ to find the fastest time.

Line #	Program output
	Enter Code [P, A, D, F, or Q]: p Sport Stacker Name: Sami Smith Times: 7.41 5.9 7.78 6.45 5.88 6.99 Average Time: 6.74 Enter Code [P, A, D, F, or Q]: f Fastest Time: 5.88 Enter Code [P, A, D, F, or Q]:

Code and Test: After reading in the file and printing the menu of actions and descriptions, you should have a **do-while loop** that prints the prompt with just the action codes followed by a **switch statement** that performs the indicated action. The do-while loop ends when the user enters ‘q’ to quit. You should be able to test your program by exercising each of the action codes. You may also want to run in debug mode with a breakpoint set at the switch statement. Although your program may not use all of the methods in SportStacker and SportStackerApp, you should ensure that all of your methods work according to the specification. You can either user interactions in jGRASP or you can write another class and main method to exercise the methods. Web-CAT will test all methods to determine your project grade.