Homework 10

GSquare Gradebook

1 Introduction

For a previous homework you implemented a user database for GSquare. Georgia Tech liked your user database so much that they decided to hire you to make the gradebook portion of GSquare.

Be sure to read the **entire document** as there are key details throughout that will help you succeed on this assignment.

2 Problem Description

This assignment asks you to write a backend that handles adding students, viewing student grades, and changing student grades. A backend is a set of methods that a UI may call to handle core operations of a program. We have provided a GUI, `GSquare.java`, as a front-end for you to test your program. To store your students, you are required to write a class that implements `SimpleSortedSet.java`, an interface provided to you.

The code for this homework is independent of Homework 7, meaning that it doesn't interact with anything you have written previously for this class.

3 Provided Files

3.1 SimpleSortedSet

A Set is a Collection which makes sure never to have duplicate elements. Any attempt to add an element which is already in the Set is ignored and the Set is unchanged. SortedSet is an interface in the Java API which describes a Set that is always sorted. This means that, even when items are added and removed, the Set makes sure that it stays sorted.

`SimpleSortedSet` is an interface which follows the same rules as SortedSet but has drastically less methods. Additionally, a `get(E e)` method has been added to allow you to use it more easily. **Do not modify this file.**

3.2 Student

Student is a class that represents a student using GSquare. It has a `name`, `testGrades`, and `homeworkGrades` fields and getters and setters where appropriate. Note that its `compareTo` and `equals` methods only check based on its name. **Do not modify this file.**

3.3 GSquare

GSquare is the JavaFX application that uses GSquareGradebook. You should be able to add students, select students and change their grades. Students should show up alphabetized. **Do not modify this file.**

4 Solution Description

4.1 MySortedSet

Write a class MySortedSet which implements SimpleSortedSet. Be sure to use generics properly. You must use an array to hold all of the elements in the set. Descriptions for all methods are in the javadocs of the SimpleSortedSet.

4.2 GSquareGradebook

Fill in GSquareGradebook with implementations for each method in it. For convenience, GSquareGradebook.java has been provided to you with undocumented empty methods. Feel free to add instance variables and more methods. You should use your MySortedSet to hold all of the students, that is, this class should have an object of that class as instance data.

- `addStudent` should attempt to add a new Student with the given name, homework grade, and test grade.
- `setTestGrade` and `setHomeworkGrade` should set the proper grade of the Student with the same name.
- `getStudents` should return a trimmed array of all Students.

5 Helpful Hints

- Make sure to implement and test all methods in MySortedSet, even if you don't use them in GSquareGradebook, they will be tested separately.
- Don't use anything which trivializes the assignment. This includes but is not limited to the any existing Collections, the class java.util.Arrays and the class java.util.Collections. If in doubt, ask on Piazza.
- You cannot create an array of generic type. Instead you must create a Object[] and cast it to a generic array (i.e. `(E[])new Object[size]`).

6 Javadocs

For this assignment you will be commenting your code with Javadocs. Javadocs are a clean and useful way to document your code's functionality. For more information on what Javadocs are and why they are awesome, the online documentation for them is very detailed and helpful.

You can generate the javadocs for your code using the command below, which will put all the files into a folder called javadoc:

```
$ javadoc *.java -d javadoc
```

The relevant tags that you need to have are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```
import java.util.Scanner;

/**
 * This class represents a Dog object.
 * @author George P. Burdell
 * @version 13.31
 */
public class Dog {

    /**
     * Creates an awesome dog (NOT a dawg!)
     */
    public Dog(){
        ...
    }

    /**
     * This method takes in two ints and returns their sum
     * @param a first number
     * @param b second number
     * @return sum of a and b
     */
    public int add(int a, int b){
        ...
    }
}
```

Take note of a few things:

1. Javadocs are begun with `/**` and ended with `*/`.
2. Every class you write must be Javadoc'd and the `@author` and `@version` tag included. The comments for a class start with a brief description of the role of the class in your program.
3. Every non-private method you write must be Javadoc'd and the `@param` tag included for every method parameter. The format for an `@param` tag is `@param <name of parameter as written in method header> <description of parameter>`. If the method has a non-void return type, include the `@return` tag which should have a simple description of what the method returns, semantically.

6.1 Javadoc and Checkstyle

You can use the Checkstyle jar mentioned in the following section to test your javadocs for completeness. Simply add -j to the checkstyle command, like this:

```
$ java -jar checkstyle-6.2.1.jar -j *.java
Audit done. Errors (potential points off):
0
```

7 Checkstyle

You must run checkstyle on your submission. The checkstyle cap for this assignment is **20** points.
Review the Style Guide and download the Checkstyle jar. Run Checkstyle on your code like so:

```
$ java -jar checkstyle-6.2.2.jar -a *.java
Audit done. Errors (potential points off):
0
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off.

The Java source files we provide contain no Checkstyle errors. For this assignment, there will be a maximum of **20** points lost due to Checkstyle errors (1 point per error). In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

8 Turn-in Procedure

Non-compiling or missing submissions will receive a zero. NO exceptions

Submit all of the Java source files you modified and resources your program requires to run to T-Square. **Do not submit** any compiled bytecode (`.class` files) or the Checkstyle jar file. When you're ready, double-check that you have submitted and not just saved a draft.

Please remember to run your code through Checkstyle!

8.1 Verify the Success of Your Submission to T-Square

Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.

2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.

3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.

4. Recompile and test those exact files.

5. This helps guard against a few things.

   (a) It helps insure that you turn in the correct files.

   (b) It helps you realize if you omit a file or files. [1] (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)

   (c) Helps find last minute causes of files not compiling and/or running.

[1] Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework will not be accepted regardless of excuse. Treat the due date with respect. The real due date is midnight. Do not wait until the last minute!