Set

1 Introduction

This homework will give you practice with using Collections by creating a Set. A Set, as you have learned, contains no duplicate and has no distinct ordering.

2 Solution Description

You are required to complete all the methods provided in `Set.java`. You will also write a class called `Student.java` which represents a student. Once you finish
tt Set.java and `Student.java`, use the provided `Driver.java` to test your set. Detailed descriptions of these classes follow.

2.1 Set

The Set class represents a Collection which cannot contain duplicates. By default, it has no ordering, and should use an array (NOT an `ArrayList`) to store its data. `Set`'s public API is:

- `public Set(Type<T> type, int initialSize)`
- `public Set(Type<T> type)`
- `public boolean add(T entry)`
- `public T remove(Object entry)`
- `public T remove(int index)`
- `public boolean contains(T entry)`
- `public T[] toArray()`
- `public boolean isEmpty()`
- `public void clear()`
- `public int size()`
- `public String toString()`

- `public T[] sort(int start, int end)`

We have provided some of the more difficult parts of the code as private helper methods, as well as a fun sorting algorithm to make the output look nice.

- `private T[] createArray(Class<T> type, int size).` Creates an array of generics of type T.
  - Since a generics type cannot be initiated as an array, we have to initiate it as an Object array then cast the array to the generics type. Ex: T[] myArray = (T[]) new Object[10].
  - The problem is myArray is still an Object array, therefore the elements in the array cannot extend Comparable, which is a built-in Java class that indicates whether an object can implement the method compareTo(Object other).
  - Thats why you need createArray to create an array of generics type T, not of Object. FYI, the method uses reflection which is not in the scope of this class and youll not be tested on it. If you have time this week, do some research on it. The first person to successfully explain to me what it is and how it works in the backend, I (*Son Tran. Not Aaron. Definitely not Aaron.*) will buy that person lunch.

2.2 Student

The `Student` class represents a student that has a first name, last name, and grade in a course.

- Instance variables
  - `private String firstName`
  - `private String lastName`
  - `private double grade`
- Constructor
  - `public Student(String firstName, String lastName).` The constructor should throw `IllegalArgumentException` if the first name and the last name are invalidated by being null, the empty string, or any number of spaces. You can create a helper method to check String validity if you so choose.
- Public methods
  - `public void setGrade(double grade)`
  - `public String toString().` Should return a String representation of the student in the following format: ”firstName lastName: grade”, i.e. (John Smith: 96.00).
  - `public boolean equals(Object other).` Two students are the same if they have the same first name and last name (you don’t have to compare their grades).

- `public int compareTo(Student other)`. This method returns a positive integer value if this object is greater than the other, a negative value if this object is less than the other, and zero if they are the same. Compare in order of grades -¿ firstName -¿ lastName. For example, if two students grades are the same, compare their first names for alphabetical ordering, then last names. (Hint: there is a method of the String class that makes this easy!)
- `public int hashcode()`. This method returns an integer which corresponds to this Student. It should follow the pattern shown in lecture.

2.3 Driver

This file is to help you test your Set. It is not meant to be comprehensive, so there are some edge cases that you may have to test yourself. Good luck!

3 Tips

- You may create helper methods in both `Set` and `Student`. Remember that your helper methods must not be public — you'll receive points off for cluttering up your public API with helper methods!
- You can comment out any parts of the driver class that you are not testing. The best strategy is to test and fix a method right after you finish it. You don't want to write all the code and end up with so many bugs that you don't even know where to start.
- Before you start any methods, write down all the edge cases and where the program may break. This is a very important practice, especially if/when you have an internship or job as a software engineer.
- Remember to throw Exceptions when the javadocs tell you to do so!

4 Checkstyle

Review the CS 1331 Code Conventions and download the Checkstyle jar file and the configuration file to the directory that contains your Java source files. Run Checkstyle on your code like this (in the directory containing all your Java source files):

```
$ java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. You can easily count the errors by piping the output of Checkstyle through `grep`.

```
$ java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java | grep -cEv "(Starting
    audit...|Audit done)"
0
$
```

The `-c` option tells `grep` to count matching lines instead of printing them, `E` means use `egrep` (extended `grep`) syntax, and `v` means invert the match. Here we use an inverted match to discard the two non-error lines of Checkstyle's output. If you have `egrep` you could leave off the `-E` and just use `egrep`.

```
$ java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java | egrep -cv "(Starting
    audit...|Audit done)"
0
$
```

Alternatively, if you are Windows, you can use this command to count the number of style errors by piping output through the `findstr` program.

```
C:/> java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java |findstr /v "Starting
    audit..." | findstr /v "Audit done" | find /c /v "!!!!!"
```

The Java source files we provide contain no Checkstyle errors. You are responsible for any Checkstyle errors you introduce when modifying these files. For this assignment, there will be a maximum of 75 points lost due to Checkstyle errors. In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

5 Javadoc

Add Javadoc comments to all classes and methods except `main`. The tags we are looking for are: **@author, @version, @param, @return**.

6 Turn-in Procedure

Submit all of the Java source files you create to T-Square. Do not submit the classes provided to you, any compiled bytecode (`.class` files), the Checkstyle jar file, or the `cs1331-checkstyle.xml` file. When you're ready, double-check that you have submitted and not just saved a draft. Theses files include, but are not limited to:

- `Set.java`
- `Student.java`

7 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 ensure 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 Wednesday. Do not wait until the last minute!