Homework 9

1 Introduction

Hello. To get use to the Collections in Java, we will be implementing our own version of Set but with additional features. Some of the operations also require a Tuple class, so you will need to also implement a Tuple class. A Tuple is an ordered pair of elements like: (a, b). You should be familiar with most of the math required for this assignment, but we have included the (incredibly helpful) Wikipedia pages for each mathmatical function you need to implement.

2 Problem Description

The goal of this homework is to get you familiar with collections. You will be given the following files:

- ExtendedSet.java - The interface that your set class will be implementing
- Tuple.java - The interface that your tuple class will be implementing
- LFilter.java - The functional interface representing the filter operation
- LMap.java - The functional interface representing the map operation
- LReduce.java - The functional interface representing the reduce operation

You will create two classes: GenericSet which implements ExtendedSet, and GenericTuple which implements Tuple. Both of these classes will be generic. Check the included Javadocs/api for descriptions of each interface.

3 Solution Description

You will be writing two classes that implement two interfaces: ExtendedSet and Tuple.

Note: You must implement all methods specified in the interface and API (including optional methods).

You should be able to read the Javadocs we have provided you ("allclasses-frame.html" should work, or you can just look at the code) and get the jist of what you need to write. Don't forget you will need something to actually hold the data of your Set (a backing structure). We've used plain old arrays in the past for this purpose, but now you are allowed (and should) use ArrayList.

3.1 Testing

We have included a driver that will test various capabilities of your solution with Strings. Note that this is purposefully *NOT* a complete testing structure and you are encouraged to write tests of your own for the various operations. You can take a look at the testing code to get some examples of lambdas in action.

3.2 Map, Reduce and Filter

The trickiest to understand functions will be, map, reduce and filter. Turns out these are three, *incredibly* useful concepts in computer science so we want you to at least try and figure out what they do. However, they are a little abstract and tricky to visualize so I'll provide a short description of how each work below.

`http://redsymbol.net/articles/crash-course-applied-functional-programming/` provides a more complete tutorial on the subject including the theoretical basis for the functions.

Do keep in mind this course is on *Object Oriented Programming*, NOT *Functional Programming* so you definitely don't need to understand anything beyond what is necessary for this assignment about Map, Reduce and Filter for this course. The focus on this assignment will not be on these methods and you should *definitely* do the other ones first and make sure you have them right before attempting Map, Reduce and Filter.

3.3 Map

Map effectively takes elements of one type and associates or *maps* them to another. This is different than casting. We are associating or making a connection, not converting or doing anything about the original object. Think about it like we can map Pokemon to integers by using the numbers that are associated with each Pokemon (Bulbasaur = 1, Ivysaur = 2, etc...). When we talk about hashing in a few weeks, you'll see another use of the idea of mapping.

The function you need to write for this assignment will essentially walk through your entire backing structure, and map things in the structure to some other type. It then puts the results of this mapping into a *new* ExtendedSet (do not change the original data) of the type we mapped to and returns those results. Think about it like you have a set of Pokemon, and I call map with "instructions" on how to map them to numbers. You will return the set of numbers that are associated with those Pokemon. The instructions I give you will be contained within the parameter passed into `map(LMap<T, E> map)`; you don't need to worry about how to specifically map things, just assume it is given to you. The parameter has one method: `E map(T element)` which can be called on the items in your array (passed in as parameters) and returns the object we map each of those items to. You'll notice filter and reduce have similar themes to this.

3.4 Filter

Filter is similar to map in that it walks through your backing structure and does something to each element based on the instructions we pass in via the parameter (`LFilter<T> filter`). This time, instead of taking an object in your array and associating it to another type, we are checking to see if each object in the array meets some criteria (based on the instructions in the parameter) and if it doesn't meet those criteria we "filter" it out. You don't need to worry about how to specifically filter things, just assume it is given to you. The parameter has one method: `boolean filter(T a)` which can be called on the items in your array (passed in as parameters) and returns true or false depending on whether or not the item needs to be filtered. Do not change the original data, but instead make a new ExtendedSet<T> that has only the elements that have not been filtered out.

3.5 Reduce

Reduce walks through your backing structure and does something to each element based on the instructions we pass in via the parameter (`LReduce<T> reduce`). Do you sense a theme here? Reduce simply reduces the amount of items in your structure by providing "instructions" on how to take two elements and making them one element. You use these instructions over and over again on your backing structure until eventually you have reduced everything to one element. Think about it like you have a backing array full of strings like: ["My", "name", "is", "Ethan"]. You walk through the array and keep accumulating until you are only left with one string. The loop over the array would make calls that look like this:

$reduce("My","name") -> \text{“}Myname"$
$reduce("Myname","is") -> \text{“}Mynameis"$
$reduce("Mynameis","Ethan") -> \text{“}MynameisEthan"$

The result of calling reduce on your entire array is then "MynameisEthan". You don't need to worry about how we actually reduced the two items at each step, assume it is given to you. The parameter to reduce has one method: `T reduce(T a, T b)` which can be called on the items in your array (passed in as parameters) and returns the reduced item. You should use that reduced item as the first parameter to reduce on the next iteration of your walk through your array. Don't forget not to alter the original data! You simply return the result of your reduction.

4 Tips

- Definitely come to a TA (EARLY) if you are struggling with any of the math, a TA should be able to assist you if you don't understand the operations
- Partial credit is absolutely enforced on this assignment. If you are struggling on one piece, work on the others assuming that piece works, we will grade each method independently

5 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.

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 begin 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.

5.1 Javadoc and Checkstyle

Be sure you download the updated Checkstyle file (6.2.1) from T-Square along with this assignment. It will be used to verify both your Checkstyle and Javadocs. Javadocs will count towards your grade on this assignment.

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
```

6 Checkstyle

Checkstyle counts for this homework. You may be deducted up to 100 points for having Checkstyle errors in your assignment. Each error found by Checkstyle is worth one point. This cap will be raised next assignment. Again, the full style guide for this course that you must adhere to can be found by clicking **here**.

Come to us in office hours or post on Piazza if you have specific questions about what Checkstyle is looking for and how to fix Checkstyle errors.

First, make sure you download the `checkstyle-6.2.1-all.jar` from the T-Square assignment page. Then, make sure you put this file in the same directory (folder) as the `.java` files you want to run Checkstyle on. Finally, to run Checkstyle, type the first line into your terminal while in the directory of your Java files and press enter.

```
$ java -jar checkstyle-6.2.1.jar *.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. You can easily count Checkstyle errors by piping the output of Checkstyle through `wc -l` and subtracting 2 for the two non-error lines printed above (which is how we will deduct points). For example:

```
$ java -jar checkstyle-6.2.1.jar *.java | wc -l
     2
```

Alternatively, if you are on Windows, you can use the following instead:

```
C:\> java -jar checkstyle-6.2.1.jar *.java | findstr /v "Starting audit..." | findstr /v "Audit
    done" | find /c /v "hashcode()"
0
```

7 Turn-in Procedure

Submit all `.java` files you wrote/changed on T-Square as an attachment. Do not submit 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.

8 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 8PM Thursday. Do not wait until the last minute!