Lab 02: Efficiently Enumerating Subsets, Part 1
COSC 290 - Fall ’22

<table>
<thead>
<tr>
<th>Starter File(s)</th>
<th>L02_starter.zip (2 .java files)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submission</td>
<td>Upload only the following file(s) to Moodle:</td>
</tr>
<tr>
<td></td>
<td>• SetsPractice.java</td>
</tr>
<tr>
<td></td>
<td>• Lab02Tester.java</td>
</tr>
<tr>
<td>Due Date</td>
<td>Mon, Sept 26th at 10PM for all lab sections</td>
</tr>
</tbody>
</table>

1 Overview

In this lab, you will continue to work with sets, this time using functionality implemented by the Java API. You will implement six functions in the provided SetsPractice class. This is a warm up to the more complex (and recursive!) functions you will implement next week.

If you finish early, there are some practice problems to get you thinking recursively in preparation for next week’s lab.

2 Your Task

Your task is to implement the five functions outlined below:

- **generateUnion** and **generateIntersection**: accepts two Set arguments and performs the respective operation. Both functions return a brand new Set that is the result, without modifying either argument.
- **isEitherSubset**: accepts two Set arguments, and returns a boolean indicating whether either Set is a subset of the other. Also returns true if both Sets are subsets of each other. Does not modify either argument Set.
  Note: this function only checks to see if either Set is a subset of the other, but not necessarily a proper subset.
- **allSubsContain**: returns a boolean indicating if the argument element exists in all subsets contained in the argument Set of Sets. Does not modify the argument Set.
- **removeAnyElement**: removes and returns any one element from the argument Set
- **removeFromAllSubs**: creates and returns a copy of the argument Set of Sets, but with the argument element removed from any subsets which contain it. Does not modify the argument Set.

Read the docstrings included with both of these functions in SetsPractice for more details on how they work. As always, don’t forget to test a variety of cases in your Lab02Tester.java which you will submit. You can assume no arguments will be null or contains null values in them.

3 Pre-Lab Questions

After reading this document and provided code, answer the following before we meet (we will discuss in-lab):

1. Remember that non-primitive (ie. Object) arguments in Java are always passed by reference. Review the comments and docstrings for the two functions you must implement. One of these functions will modify its argument and one will not – which is which?
2. Review the Java API documentation for Set: https://docs.oracle.com/javase/7/docs/api/java/util/Set.html
   Fill in the blank: Set is not a class, it is a: __________. Refresh yourself – how is this different than a class?
3. The above API page indicates that Collection and Iterable are superinterfaces of Set. What do you think "superinterface" means? What assumptions can you therefore make about a Set object?
4. **Lab02Tester** makes use of HashSet objects – review the Java API documentation for HashSet:
https://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html

HashSet implements *four* constructors. Ignore the two constructors which accept an `int initialCapacity` argument – what is the difference between the two remaining constructors?

### 4 Submission

See the top of this document for this assignment’s due date/time per your lab section. As always, you will be evaluated on the quality of the test cases provided in your `Lab02Tester.java`.

When uploading your submission to Moodle submit **only** the files listed below – please **do not** upload a .zip or any .class/.java~ files:

- SetsPractice.java
- Lab02Tester.java

### 5 Post-Lab Questions

Here’s a sneak preview of next week’s Pre-Lab questions! If you finish the above early, answer the following questions and document your answers – we will discuss these next week:

1. Sets can be defined via *enumeration* or *abstraction*.
   
   Let’s define `S` by enumeration: `S := {1, 2, 3}`
   
   and let’s define `T` be abstraction: `T := {(x + 1) mod 4 : x ∈ S}`
   
   What are the elements of `T`?

2. Suppose `S := ∅`. What is `P{S}`?

3. Consider `A := {a, b, c}`
   
   What is `P(A − {a})`? See if you can observe a relationship between `P(A)` and `P(A − {a})`

4. Given a set `S` of arbitrary size and let `x` be some element in `S`.
   
   Let `P_s := P(S)`
   
   Define `T := S − {x}`
   
   and define `P_t := P(T)`
   
   Express `P_s` in terms of `P_t` and `x` using set operations (union, intersection, difference) and *set abstraction* (see first question).

   Your answer to the last question provides a hint on how you might implement power set generation using recursion (think of `P_t` as the result of a recursive call on a smaller input).