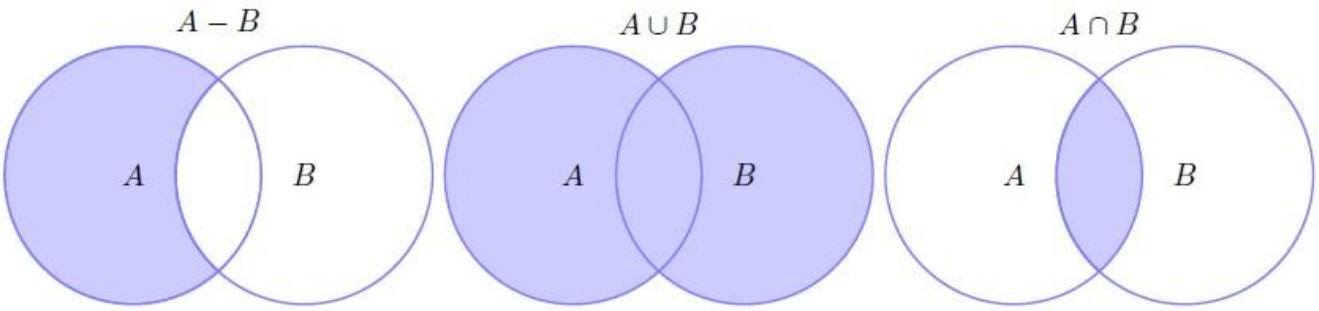
CompSci 251

Assignment 7

Due Nov. 12th, 2018

Introduction

This assignment will focus on Array Lists, Linked Lists, the List interface, and sets. A set is a collection of elements where duplicates are not allowed and elements can appear in any order. Below is a diagram showing two sets, A and B, where each image shows a different set operation performed on A and B. The three operations are subtraction, union, and intersection.



Assume set A and B hold Integers.
Set A = {1,2,3,4}
Set B = {4,5,6}

Subtraction

The left most diagram shows subtraction. The result is everything in set A minus anything in Set B.

A – B = {1,2,3}

Union

The middle diagram show union. The result is everything in set A and everything in Set B. Remember, no duplicates are allowed.

A ∪ B = {1,2,3,4,5,6}

Intersection

The right most diagram shows intersection. The result is only elements that occurs in both sets.

A ∩ B = {4}

Program Specification

For this assignment, we will use Array Lists and Linked Lists that store Integer objects to represent our sets. We will assume that the sets created are well behaved, meaning no duplicates will occur. This means you will not have to check for this. But remember, sets can be still entered out of order.

What you need to do

I have provided a template for the Sets class, which will be a class of nothing but static methods and all parameters will be List objects. This means any class that implements the List interface can be passed to these methods. The methods you will have to write are:

- public static boolean isElement(i , list)
    - Search the passed in list for the passed in Integer i. If the Integer occurs in the list, return true, else return false.

- Public static boolean isSubSet(list1, list2)
    - Determine is list1 is a subset of list2 (list1 ⊆ list2). For list1 to be a subset of list2, all elements that occur in list1 must occur in list2.

      list1 = { 1 , 5 , 9 }
      list2 = { 1 , 4 , 5 , 15 , 9 , 8 }

      list1 is a subset of list2 because all of its elements occur in list2.

      list1 = { 1 , 5 , 9 }
      list2 = { 1 , 4 , 5 , 15 , 8 }

      list1 is not a subset of list2 because 9 does not occur in list2.

- public static boolean isSuperSet(list1, list2)
    - Determine if list1 is a super set of list2 (list1 ⊇ list2). For list1 to be a super set of list2, all elements in list2 must occur in list1.

      list1 = { 1 , 4 , 5 , 15 , 9 , 8 }
      list2 = { 1 , 5 , 9 }

      list1 is a super set of list2 because all of list2's elements occur in list1.

      list1 = { 1 , 4 , 15 , 9 , 8 }
      list2 = { 1 , 5 , 9 }

      list1 is not a superset because 5 occurs in list2 but not in list1.

- public static List<Integer> subtract(list1, list2)
    - Perform the subtraction of the two lists as described above.
    - Returns the result as a new List object in increasing sorted order.

- public static List<Integer> union(list1 , list2)
    - Perform the union of the two lists as described above.
    - Returns the result as a new List object in increasing sorted order.

- public static List<Integer> intersection(list1, list2)
    - Perform the intersection of the two lists described above.
    - Returns the result as a new List object in increasing sorted order.

- Public static boolean equals(list1, list2)
    - Determine if two sets are equal by checking if all elements in list1 occur in list2, and all elements in list2 occur in list1.
    - Do not confuse this with the equals method of the Object class. This is not related an we are not overriding anything. Notice the parameters.

        list1 = { 5 , 1 , 9 }
        list2 = { 1 , 5 , 9 }

        list1 and list2 are equal since both share the exact same elements.

- public static List<String> cartesianProduct(list1, list2)
    - Perform the cartesian product of two lists. The cartesian product is creating all pairs between all elements in list1 and list2. The first element in each pair will be from list1.

        list1 = { 1 , 4 }
        list2 = { 2 , -6 , 4}

        result = { (1,-6) , (1,2) , (1,4) , (4-6) , (4,2) , (4,4) }

    - Before performing the Cartesian Prodcut, make sure each list is sorted.
    - Include duplicate elements from opposite set. In the example, 4 occurs twice.
    - Notice this method returns a List<String>. Since Java does not support Tuples, we will mimic a pair by converting the pair of Integers to a String. The format is
        - "(<list1 element>,<list2 element>)"

- Public static void sort(list)
    - Sort the passed in list from smallest to largest values. You may use any sorting technique you wish.
    - This method does not return anything, so the list passed in gets sorted.

Graduate Students

Add the extra method specified below.

- public static List<Integer> complement(List<Integer> list, int lowerBound, int upperBound)
    - The complement of a set is all numbers not in the set.

        list = { 1 , 2 , 4 }

        complement(list) = all numbers from $\pm\infty$ not including 1, 2, and 4

    - Since computers have their limitations, we cannot represent $\pm\infty$, instead you will use the lowerBound and upperBound to represent all possible real numbers.

        list = { 1 , 2 , 4 } , lowerBound = -4, upperBound = 8

        complement(list) = { -4 , -3 , -2 , -1 , 0 , 3 , 5 , 6 , 7 , 8 }
- Write a Junit test case that tests this new method. You must create a new test method in the Junit file, do not add these tests to another test method.

Keep in mind, Lists are interfaces, so they cannot be instantiated (ie. List<Integer> list = new List<>() will not work). You must pick a concrete class that implements the List interface and return that. You may choose which concrete class to return for all methods.

I have provided a template you should use for the class Sets. There are also some Junit tests written to help with debugging. You must write a driver though to further test your code. Create a few Array Lists, create a few Linked Lists, and add different numbers to each type. Then, test **ALL** your methods to make sure each work correctly. My recommendation is to keep your lists short, this will make writing and understanding your results easier.

You may ask why do I want you to write a driver when I provided Junit tests? First, my Junit tests are not extensive, so your program may still have errors that I did not think about testing for. This means your driver may catch something I did not. Second, it will be good practice. Learning to write your own driver and tests is a valuable skill in programming. If you write some code and never test it, you will never know how good your code actually is. You must know what to test, and how to test it so you can show that your program functions correctly.