"Please start working on this assignment as early as possible!"

CS2400 Project 1

Total points: 100

Purpose:

1. Warm up your programming skills.
2. Master the interface and implementations of bags.

Task Description:

1. (8 pts) **Add** three methods: ***union***, ***intersection***, and ***difference*** to the interface BagInterface for the ADT bag. The detailed description of the three methods could be found on pages 2 and 3 of this assignment.
2. (10 pts) **Write** a client program, "ArrayBagTest.java", which contains a main method to **test the three methods** (*union*, *intersection*, and *difference*) you will implement for the class **ResizeableArrayBag**.
3. (30 pts) **Implement** the three methods, ***union***, ***intersection***, and ***difference,*** for the class **ResizeableArrayBag**. Note: the class ResizeableArrayBag presents an implementation of the ADT bag using a resizable array.
4. (10 pts) **Write** a client program, "LinkedBagTest.java", which contains a main method to **test the three methods** (*union*, *intersection*, and *difference*) you will implement for the class **LinkedBag**.
5. (30 pts) **Implement** the three methods, ***union***, ***intersection***, and ***difference,*** for the class **LinkedBag**. Note: the class LinkedBag presents an implementation of the ADT bag using a linked chain.
6. (12 pts) **Use Big O notation** to indicate the time complexity of each method defined (the three methods implemented in ResizeableArrayBag and LinkedBag **in the best case and the worst case**, respectively. Please also **provide** explanations for your answers.

Table 1. The time complexities of this assignment

	ResizeableArrayBag			**LinkedBag**		
	union	**intersection**	**Difference**	**Union**	**intersection**	**difference**
Time Complexity in the Best Case						
Time Complexity in the Worst Case						

What to Submit?

1. In your shared (with "2404s21") GitHub repo:

a) Source codes for Tasks 1-5, which are *"BagInterface.java", "ResizeableArrayBag.java", "LinkedBag.java", "ArrayBagTest.java", "LinkedBagTest.java".*
Note:

- The *class Node* could be used as a member inner class of the class *LinkedBag*. Alternatively, it is fine to make *class Node<T>* an independent class and place in an additional java file “Node.java”.
- Please properly comment java code to improve the readability. Use Javadoc to produce clear and neat documentations.
- Write test cases first. For this project, using a unit test framework (google “Java unit test frameworks”) is a plus and gives your extra credit (10 pts). The tester class should be a separate class than that containing main() and in a separate file. It will be **required** for all the projects after this one.

b) The generated Javadoc folder (and subfolders).

c) A README.md file containing:

  i) Your full names

  ii) Each member’s contribution

  iii) Whether you have implemented extra features (eg. unit test class using some framework)

  iv) A link to your screencast on some video streaming website. The screen cast should introduce your names and demo run your code, and each person should talk about their code briefly. Please make it <5 minutes. (Hint: you can edit the video – concatenate/delete video sections.)

  v) A link to the Javadoc’s start page (index.html).

2. Submit on Canvas: An Excel file, *“efficiency.xlsx” (or Word file “efficiency.docx”),* which includes

a) Your full names

b) Link to your GitHub repo

c) **Table 1** (as given on Page 1 of this assignment) showing the time complexities for Task 6 and **Explanations** for each of your answers.

Note: You will be graded based on the quality of your program and documentation, as well as the correctness of your algorithm analysis.

- **Union**

The *union* of two collections consists of their contents combined into a new collection. Add a method union to the interface BagInterface for the ADT bag that returns as a new bag the union of the bag receiving the call to the method and the bag that is the method’s one argument. Include sufficient comments to fully specify the method.

Note that the union of two bags might contain duplicate items. For example, if object *x* occurs five times in one bag and twice in another, the union of these bags contains *x* seven times. Specifically, suppose that bag1 and bag2 are Bag objects, where Bag implements BagInterface; bag1 contains the String objects a, b, and c; and bag2 contains the String objects b, b, d, and e. After the statement

BagInterface<String> everything = bag1.union(bag2);

executes, the bag everything contains the strings a, b, b, b, c, d, and e. Note that union does not affect the contents of bag1 and bag2, and the order of data items in the resulting bag everything doesn’t matter.

- **Intersection**

The *intersection* of two collections is a new collection of the entries that occur in both collections. That is, it contains the overlapping entries. Add a method intersection to the interface BagInterface for the ADT bag that returns as a new bag the intersection of the bag receiving the call to the method and the bag that is the method’s one argument. Include sufficient comments to fully specify the method.

Note that the intersection of two bags might contain duplicate items. For example, if object *x* occurs five times in one bag and twice in another, the intersection of these bags contains *x* twice. Specifically, suppose that bag1 and bag2 are Bag objects, where Bag implements BagInterface; bag1 contains the String objects a, b, and c; and bag2 contains the String objects b, b, d, and e. After the statement

BagInterface<String> commonItems = bag1.intersection(bag2);

executes, the bag commonItems contains only the string b. If b had occurred in bag1 twice, commonItems would have contained two occurrences of b, since bag2 also contains two occurrences of b. Note that intersection does not affect the contents of bag1 and bag2.

- **Difference**

The *difference* of two collections is a new collection of the entries that would be left in one collection after removing those that also occur in the second. Add a method difference to the interface BagInterface for the ADT bag that returns as a new bag the difference of the bag receiving the call to the method and the bag that is the method’s one argument. Include sufficient comments to fully specify the method.

Note that the difference of two bags might contain duplicate items. For example, if object *x* occurs five times in one bag and twice in another, the difference of these bags contains *x* three times. Specifically, suppose that bag1 and bag2 are Bag objects, where Bag implements BagInterface; bag1 contains the String objects a, b, and c; and bag2 contains the String objects b, b, d, and e. After the statement

BagInterface leftOver1 = bag1.difference(bag2);

executes, the bag leftOver1 contains the strings a and c. After the statement

BagInterface leftOver2 = bag2.difference(bag1);

executes, the bag leftOver2 contains the strings b, d, and e. Note that difference does not affect the contents of bag1 and bag2.