Lab Assignment 02, Data Abstractions and Structures, CSE 274, Fall 2018
Department of Computer Science and Engineering, Miami University

Bags and JUnit Test

In this lab, we will implement the Bag ADT and test our code using the JUnit library. More specifically, we will use an array to implement our Bag. It will be a bag of Point objects (import the **Point** class from java.awt, i.e. java.awt.Point). Highlights of the Bag ADT:

- Order is unimportant (and unpredictable)
- We want to be able to:
    - Add items
    - Remove items (2 ways...remove a specific item, and remove any item)
    - Find the size
    - Check if an item is in the bag
    - Count the number of times an item is in the bag
- We will use bags with a limited size (you can use a constant greater than or equal to 25).

Today we will use arrays to implement the Bag ADT. But there are other good ways to do it. We will look at another later (using linked data). We are implementing a bag that specifically holds Point objects. But it would be even better if we implemented bag so that it could hold any objects. We'll look at that later, using generics.

Some things that you may not have seen before:

- Partially filled arrays. We will create an array that is bigger than we need, and fill it in as we go. This means keeping track of how much of the array is filled. Doing it this way avoids the somewhat costly operation of creating a new array every time we run out of room in our array.
- When we write the method that adds an item to the bag, we could decide to write it as a void method. However, it is often useful to return a boolean. The boolean is a way for the method to report whether it completed successfully. If the method returns true, we will know that the item was successfully added. If it returns false, we will know that the item was not added.
- Since order is unimportant, we are free to change the order of the items if it makes our code more efficient. You will notice this when we write the method that removes a particular element.

Things to consider when implementing any ADT:

- Efficiency: When faced with more than one way to implement an ADT, we have to stop and ask "Which one is faster? Which one uses less memory?" (mostly we will be thinking about speed rather than memory) Sometimes there is a tradeoff. A solution might be fast in some ways, and slow in other ways. Always be thinking about efficiency. **Couldn't we just use an ArrayList to implement Bag?** We could, but it is terribly inefficient compared to using an array. ArrayList objects have a lot of unneeded functionality too. We are trying to build a data structure that exactly meets our needs. No more and no less.

- Edge cases: We want our data structures to work 100% of the time. Often, we forget about what happens in the "extreme" cases: what happens if we try to add one more item to a bag but the bag is full? What happens if we try to remove the only item in the bag? Should we throw an exception?

Lab Assignment 02, Data Abstractions and Structures, CSE 274, Fall 2018
Department of Computer Science and Engineering, Miami University

Bags and JUnit Test

Before getting started:

- Create a new project named Lab2ArrayBag
- Download the BagInterface class and add it to your project.
- Create a new class in your project named PointBag. This class will implement the BagInterface. Define all the unimplemented methods along with a class constructor.
- Now create a JUnit tester class, PointBagTester in your project. Test all the methods of the PointBag class.

What you need to do:

- You are responsible for implementing and testing (using the JUnit library) all the methods.

What to do when you are done:

- Submit all java files to the appropriate submission folder on Canvas.

Grading Rubric:

Implements BagInterface	2
Well commented	5
The following methods work correctly (Partial grading for incomplete methods)	
public int getCurrentSize();	5
public boolean isEmpty();	5
public boolean add(T newEntry);	8
public T remove();	8
public boolean remove(T anEntry);	8
public void clear();	8
public int getFrequencyOf(T anEntry);	8
public boolean contains(T anEntry);	8
public T[] toArray();	8
JUnit test for each of the above 9 methods (3 points each)	27
Total	**100**