Assignment Prefix: lab02

Due Date: Wednesday, May 24th @ 11:59pm

Points: 100

This is an individual programming assignment.

**Restrictions:**

You cannot use any methods from the Java Arrays class to copy an array, check for equality, or otherwise manipulate an array. You must write the Java code to perform these functions. Note that this does not mean that you cannot use an Array, merely that you cannot call any of the static methods of the Arrays class.

**In this project you will be doing the following:**

Create a NetBeans project named lab02 and ensure it is imported into Subversion.

In this assignment you will be creating a generic interface *GenericBag* which would be a collection of items that the client would define. The *GenericBag* interface will be a generic interface with a Type Parameter. Just like in lab01, Bag will not order the items in any particular order nor does it prevent any duplicates. You may reuse the code from the lab01 assignment for the *Bag* interface if you'd like as it is the same. Provide the following methods in the interface:

Develop a Generic interface named ***GenericBag*** that can store certain number of items (type will be specified by the client). Provide the following methods in the interface. You can copy your code from the previous assignment.

* ***public int getCurrentSize( )***
  + returns a count as of numbers in the bag
* ***public boolean isEmpty( )***
  + checks if the bag is empty, returns true when empty
* ***public void add (T value)***
  + adds a new number ***value*** to the bag
* ***public boolean remove (T value)***
  + removes the first occurrence of the number ***value*** from the bag
* ***public void remove( )***
  + removes a randomly selected entry from the bag
* ***public void clear( )***
  + removes all the numbers from the bag
* ***public int getFrequencyOf(T value)***
  + returns a count the number of times the number ***value***  exists in the bag
* ***public boolean contains(T value)***
  + returns whether the bag contains the number ***value***

Implement the generic ***SinglyLinkedList*** class from chapter 3.2.1. You may copy the code from the book when doing so, but you may want to attempt to implement it yourself first. We will be inheriting from this class later so you should make the field values protected instead of private.   
  
Design a generic class called ***LinkedBag***which extends the generic ***SinglyLinkedList*** class and implements the ***GenericBag*** interface.

* Because we are inheriting from another class, it is not necessary to declare additional fields.
* Implement the methods received from the interface. Some of these methods may not be difficult to implement or may be exactly the same functional as methods already implemented in the ***LinkedBag***superclass, in which case you can call those methods as appropriate.
* The method that adds to the Bag should behave similar to the ***ArrayBag*** class from lab01 in that it adds elements at the end of the Bag.
* The method that removes a specific item which is passed as a parameter should use the ***equals( )***method to compare the contents of objects in the bag with the contents of the parameter. If there is an object in the bag with the same contents then, it removes that item from the bag and returns true. If there is no item with the same contents, then the method returns false.
* Implement the following additional methods
  + A method that returns an item at a specific index position in the bag. As with the ***ArrayBag*** class (and arrays in general) you should consider index 0 the first item in the bag.
  + A method that returns an array that stores all of the values in the ***LinkedBag*** with the same ordering. The array should be of the same generic type. As with other cases, you will need to write this code as follows:
    - T[] array = (T[]) new Object[size];

Create a user-defined class called ***Student***. Each Player object will have the following attributes (instance variables): ***name, ID***, and ***GPA***. Use appropriate data types to define these instance variables and use recommended naming conventions for the variable names.

Provide a constructor, implement the accessor and mutator methods for each of the instance variables, and include the toString( ) and equals( ) methods. For the equals method, you can assume that two students are the same if they have the same name and ID.

Also include a static factory method that can be used to generate instances of the Student class. This method should create a student with a random name, ID, and GPA and return that student object.

Finally, create a Client Program ***BagClient*** with the ***main( )*** method. Inside the main method do the following:

* Create an object of ***LinkedBag*** called course to store information about 14 different students. For practice, when you declare the object reference for variable course, use the interface type ***Bag***.
* Run a ***for loop*** to call the static factory method of the Student class to generate student objects to store in the bag you created.
* Remove a random student from the course.
* Add a new Student with some made up information to the course. You should read this in using the Scanner class.
* Display the current count of students in each course.
* Remove the Student that you just added earlier with made up information from each course using appropriate method.
* Display the current count of students in each course.
* Use a for loop (or implement the toString method for the ***LinkedBag***) to print the information of the Students in each course.

Comment your ***GenericBag*** interface, ***LinkedBag*** and ***Student*** classes with Java Doc commenting style. Use single line or block style comments as necessary or appropriate for the ***BagClient*** program.

Using Microsoft Visio draw a UML class diagram displaying all the classes and the relationships between them.

For this diagram **include the client class**. The contents of the client class can just be the ***main( )*** method. This class will have association relationships between ***SinglyLinkedList***, ***LinkedBag****,*and ***Student*** classes. You should include your interface in the diagram as well.

An association relationship simply means that a class uses another class – In this project, Client class is using ***LinkedBag*** and ***Student***. This is depicted by an undirected solid line connecting the two classes.

**Things to turn in:**

* Ensure that your source code is committed to the repository .
* Attach the ***UML class diagram*** and generated ***Javadoc*** comments (you will need to create a .zip file for them) as part of the Blackboard submission.