Code Project 1: Bag

Project Description

Prompt: Implement a “Bag” (aka MultiSet: a set that may contain duplicates, also described as an unordered list) of Strings.

Spec: your Bag class must implement the Collectible interface (provided in the zip file), using whatever implementation strategy you choose.

Input: some amount of text

Output: information about the uniqueness of the words given in the input. Specifically, you will need to print the total size of the collection (how many words there are, including duplicates), and a list of all the unique words and how many times they appear. For example, given a list of words “apple banana apple grape watermelon watermelon grape apple”, your Bag program should print out:

Total number of words: 8
Unique number of words: 4
watermelon 2
grape 2
banana 1
apple 3

Notes on the output:

- the list of words can appear in any order (since MultiSets are unordered lists)
- any amount of whitespace can be between the distinct word and its count, but each set{word, count} must be separated by one newline (you can just rely on the main method to print the collection out in the correct format)

Already implemented:

- public static void main(String [] args): the main method is already implemented for you, including reading words from standard input and printing the contents of your Bag to standard output in the correct format (in order for this to work, your Bag must implement the Collectible interface)
- the BagIterator class and the iterator() method: this implementation of the BagIterator class assumes that you are using the LinkedList implementation of a Bag. If you are doing an ArrayList implementation of a Bag, you will need to rewrite the BagIterator to work with your implementation strategy.

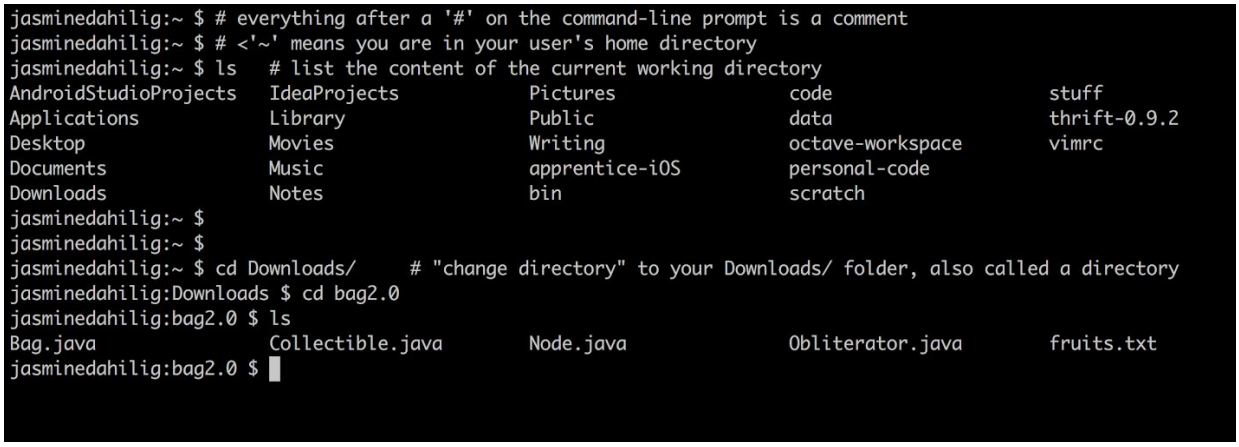
To run and test your program:
(from a command-line prompt, such as Terminal or Cygwin)

javac Bag.java
java Bag < fruits.txt

Walkthrough

Search for your command-line prompt program and open it. (If you have an Apple computer, press the keys CMD (apple symbol) + SPACE, to get to the Finder search bar and search for the “Terminal” program.)

Once at the command-line prompt, change directories ("cd") to wherever you downloaded the bag2.0.zip file. I assume you have already unzipped the file, so there should be a bag2.0/ directory (the ending slash indicates it's a directory or folder). Use cd to navigate into the bag2.0/ directory. Remember you can use "ls" to list the contents of the directory you are currently in.



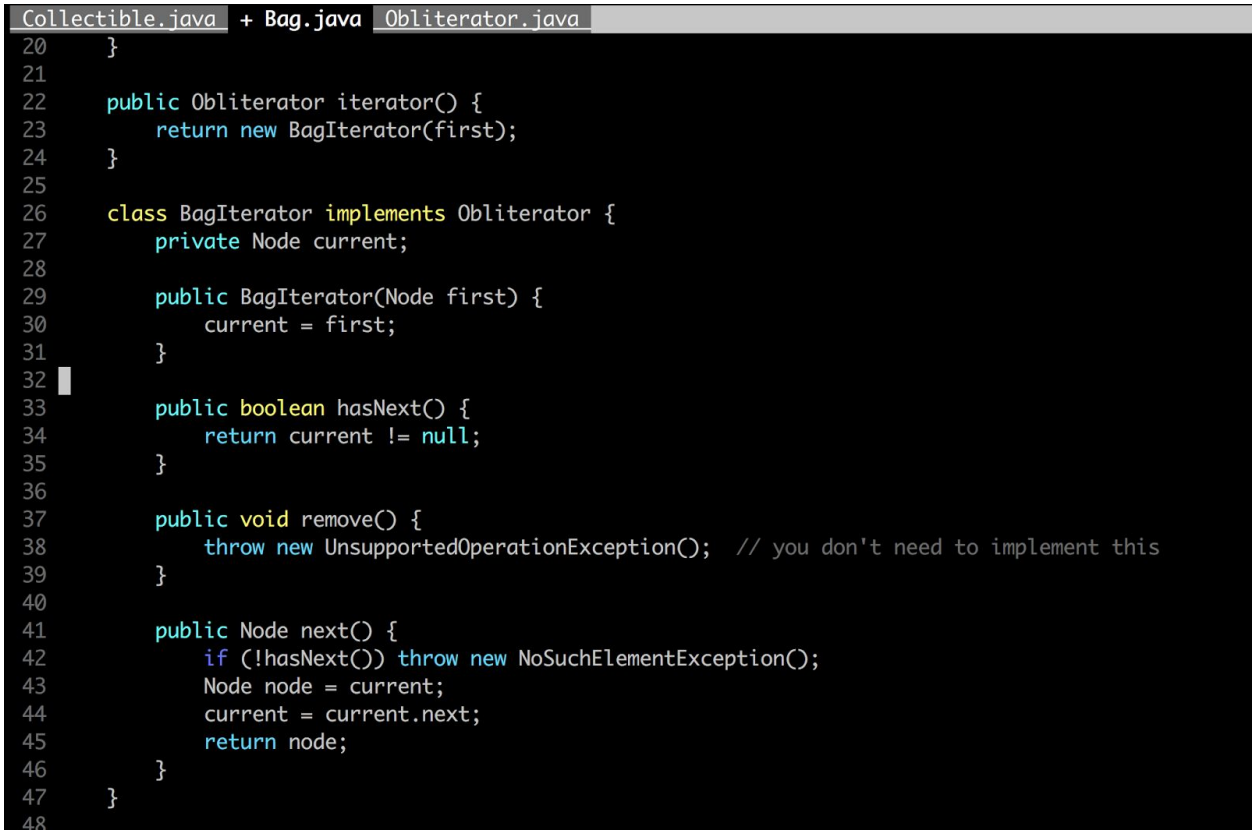
Now to compile the program. Compiling the program checks if your syntax is valid and, if valid, compiles or transforms your Java code into Java bytecode. Compile your application by running "javac Bag.java". You’ll see when you run this command right out of the box, you generate an error. Read the error and identify the problem.

```
jasminedahilig:bag2.0 $ javac Bag.java
Bag.java:5: error: Bag is not abstract and does not override abstract method add(String) in Collectible
class Bag implements Collectible {
^
Bag.java:57: error: cannot find symbol
               bag.add(item);
                   ^
  symbol:   method add(String)
  location: variable bag of type Bag
Bag.java:66: error: cannot find symbol
       System.out.format("Total number of words: %d\n", bag.size());
                                                          ^
  symbol:   method size()
  location: variable bag of type Bag
3 errors
jasminedahilig:bag2.0 $
[algo] 0:vim- 1:bash* 2:bash                                    "Jasmines-MacBook-Pro.l" 15:36 17-Sep-16
```

You need to implement the add(String), isEmpty(), and size() methods to satisfy the constraint of implementing the Collectible interface. [Not shown in the error above] You will also need to implement the method uniqueSize(), which will currently throw an exception at runtime (you'll need to remove that exception and put your code there instead).

```
 1 import java.io.*;
 2 import java.util.NoSuchElementException;
 3 import java.util.Scanner;
 4
 5 class Bag implements Collectible {
 6     private Node first;
 7     private int n; // number of nodes
 8     private int total; // total number of words in this bag
 9
10
11     // Implement a constructor
12     // public Bag() { ...
13
14
15     // Implement Collectible interface methods here
16
17
18     public int uniqueSize() {
19       throw new UnsupportedOperationException(); // not a thing, this throws an exception
20     }
21
22     public Obliterator iterator() {
23         return new BagIterator(first);
```

If you look further down the file at the rest of the Bag class code, you'll see that there is already an implemented BagIterator class for your Bag. This includes implemented hasNext(), next() methods, and an unimplemented remove() method. You can leave the remove() method alone for now because we won't use it for this project and it's not required in the Obliterator interface.



An “Iterator” is an object that knows how to iterate, or traverse, through your data structure element by element. Iterators are important because they allow client code (i.e. any other code that may use your Bag class) to traverse your data structure without knowing the implementation details. In order to allow this, iterators are usually expected to have three methods: hasNext() returns false if you’re at the end of the collection, next() returns the next element of the list, and remove() allows you to remove items while iterating through the collection. This project requires your iterator to implement the Obliterator interface, which only asks for hasNext() and next().

You can see from the BagIterator’s next() method that it relies on a LinkedList implementation of your Bag data structure (the “node.next” code indicates that it relies on the “linking” pattern of a LinkedList structure). If you are not using a LinkedList to implement your bag, you will have to change BagIterator to fit your implementation, and you will need to make sure it still implements the Obliterator interface (specified in Obliterator.java).

Collectible.java + Bag.java **Obliterator.java**

```
1 interface Obliterator {
2
3   boolean hasNext();
4
5   Node next();
6
7 }
```

You can see an example of the BagIterator being used in the main method below.

```
68
69          // Print distinct words in bag and their frequency
70          for (Obliterator i = bag.iterator(); i.hasNext(); ) {
71              Node node = i.next();
72              System.out.format("%s %d\n", node.item, node.count);
73          }
74      }
```

When you've filled in the add(), size(), uniqueSize() and isEmpty() methods, you can compile your class with "javac Bag.java". Once your code is compiled to java bytecode (which is what is stored in the Bag.class and other .class files that show up after compilation), you can run it. Run your program with the command "java Bag < fruits.txt". The "<" left arrow is a special operator that means "pass this file as input to this program", and in this case we are passing the fruits.txt file as input to your Bag program.

```
jasminedahilig:bag2.0 $ ls
Bag.java              Collectible.java       Node.java              Obliterator.java       fruits.txt
jasminedahilig:bag2.0 $ javac Bag.java
jasminedahilig:bag2.0 $ ls
Bag$BagIterator.class   Bag.java              Collectible.java       Node.java              Obliterator.java
Bag.class               Collectible.class     Node.class             Obliterator.class      fruits.txt
jasminedahilig:bag2.0 $
jasminedahilig:bag2.0 $ java Bag < fruits.txt
Total number of words: 8
Unique number of words: 7
strawberry 1
grape 1
peach 1
carrot 1
pomegranate 1
banana 1
apple 2
jasminedahilig:bag2.0 $
```