CS310: Advanced Data Structures and Algorithmns

Assignment 2 – Solution

1. Performance with scaling up JDK Collections. See slides at end of class 2 for examples

   (a) Since a search in a Hash table is constant in the size of the table, the search would still be (approximately) 1ms.

   (b) In a logarithmic search doubling the input adds c to the runtime, i.e. some constant not huge, hard to determine. If we follow the formula $f(n) = c * log(n), where f(1000) = 1ms$ then c log (1000) = 1, so c = 1/log(1000) and f(2000) = c log(2000) = log(2000)/log(1000) = 11/10 approx., = 1.1 ms, approximately.

   (c) A search in a linked list is $O(n)$, therefore the search time approximately doubles to 2ms.

2. O(1) Methods Map's get(), and put() methods can be implemented in O(1) time. Also, size(), containsKey() and remove() would be O(1), too. The Set's add(), remove(), contains(), and remove() methods are O(1) time, if we use HashSet. Also size(), isEmpty(). The map operations that can be implemented with O(1) are get, put and remove. The set operations are add, remove and contains.

3. Choosing the appropriate Collections classes to use

   a. The simplest implementation is a linked list of the lines. Implementation: Read buffer line by line; add each line to a list (add appends the line in the end of the list); when the file is done traverse the list backwards and print the lines. This is an O(N) implementation (where N is the number of lines), since add is O(1) if we have a pointer to the end of the list. A TreeMap can also be used with the line number as key, but this is less efficient since add is O(logN).

   b. Use a HashMap<String,Integer> where the keys are words and the values are their occurrence. Implementation: Read file word by word; each word – if it exists in the map, increment its counter. Otherwise, insert with counter value 1. After the file is read – traverse the map key set and print all the key values. Order is not important and therefore HashMap should be used. Complexity: O(N), where N is the number of words.

   c. Exactly like b above, only print also the value in addition to the key. This is also $O(N)$.

4. Using LinkedList to implement a specialized list

```
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
// implement the API for Bag on pg. 121 including the iterator promised by "implements
public class Bag<Item> implements Iterable<Item> {
    private List<Item> list;

    public Bag() { list = new LinkedList<Item>();}
    public int size() { return list.size();}
    public boolean isEmpty() { return list.isEmpty(); }
    public void add(Item item) { list.add(item); }
    public Iterator<Item> iterator() { return list.iterator(); }
}

public class TestBag {
```

```
    public static void main(String[] args) {
            Bag<String> fruit = new Bag<String>();
            fruit.add("apple");
            fruit.add("pear");
            fruit.add("apple");

            int count = 0;
            for (String f : fruit) {
                    if (f.equals("apple"))
                            count++;
            }
            System.out.println("count: " + count);
    }
}
```

Note: for this assignment, it's OK to have the test code in main. But it's better to have it in another class. Methods available on a Bag: constructor, size, isEmpty, add, iterator, equals, hashCode, toString, getClass and some more obscure Object methods.

5. Map vs. ST

   (a) i. size: int size(), ST size, int size(), same
   ii. isEmpty: boolean isEmpty(), ST: boolean isEmpty(), same
   iii. containsKey: boolean containsKey(KeyType key), ST: boolean contains(Key key), same except for names (or "same")
   iv. get: ValueType get(KeyType key), ST: Value get(Key key), same
   v. put: ValueType put(KeyType key), ST: void put(Key key, Value val), different: JDK put returns old value, ST returns void
   vi. remove: ValueType remove(Object key), ST: void delete(Key key), different: JDK remove accepts Object, not just KeyType, returns old value, ST delete accepts only Key type, returns void
   vii. clear: void clear(), ST: none
   viii. keySet: Set<KeyType>keySet() ST: Iterable<Key>keys(), different: JDK returns a Set, ST only an Iterable (might not be a set by type).
   ix. values: Collection<ValueType>values() ST: none

   (b) clear and values are missing in ST. For clear, we can iterate through the keys returned by keys(), and call delete on each. For values, we similarly iterate through the keys calling get for each, and putting each resulting value in a list.

   (c) i. Point2D: has consistent equals and hashCode from Object, so can use HashSet, but has no compareTo, so cannot use TreeSet.
   ii. Date, pg 103: has own equals and hashCode from Object, but these are inconsistent, so HashMap cannot be used. Has no compareTo, so TreeSet cannot be used.
   iii. String, pg 80: has own equals and hashCode, consistent, so HashMap can be used. Has equals and compareTo, so TreeSet can also be used.
   iv. Counter, pg 85: has consistent equals and hashCode from Object, so can use HashSet, Has no compareTo, so TreeSet cannot be used.

6. Equals/hashCode/compareTo for HashSet and TreeSet elements.

   (a) Point2D: identified by (x,y), so should use both in equals and hashcode if using a set of Point2Ds. This code is modified from PhoneNumber in slides. Note: OK to leave out this code for Point2D out on the argument that it already has consistent equals and hashCode, but it's much better to have field-based equals, to avoid allowing two points in the set with same coordinates.

```
public boolean equals(Object other) {
      // can skip the following line, just faster sometimes
```

```
        if (other == this) return true;
        if (other == null) return false;
        if (other.getClass() != this.getClass()) return false;
        Point2D that = ( Point2D) other;
        return (this.x == that.x) && (this.y == that.yh);
    }
    public int hashCode() {
       // or use another prime here, or just xor them, or ...
        return 31*x + y;
    }
```

(b) Date: has equals based on month, day, year, needs consisten hashCode;

```
    public int hashCode() {
       // or use another prime here, or just xor them, or ...
        return 31*31*month+ 31*day + year;
    }
```

(c) String: nothing needed

(d) Counter: As with Point2D, it's just a good idea to implement equals and hashCode here:

```
  public boolean equals(Object other) {
        if (other == this) return true;  // or skip this line
        if (other == null) return false;
        if (other.getClass() != this.getClass()) return false;
        Counter that = (Counter) other;
        return this.name == that.name;
    }
    public int hashCode() {
        return name.hasCode();  // Use String.hashCode
    }
```

7. Set Equality across HashSet and TreeSet

```
import java.util.HashSet;
import java.util.Set;
import java.util.TreeSet;

public class SetEquals {
    public static void main(String[] args) {
        HashSet<String> a = new HashSet<String>();
        Set<String> b = new TreeSet<String>();
        a.add("banana");
        b.add("banana");
        a.add("apple");
        b.add("apple");
        System.out.println("set =: "+ a.equals(b));
    }
}
```

8. More on Interfaces

(a) void trimToSize()

(b) Command line:

```
cs310Ex>javac TestArrayList.java
TestArrayList.java:8: error: cannot find symbol
     array.trimToSize();
           ^
```

```
  symbol:   method trimToSize()
  location: variable array of type List<String>
1 error

Alternatively, in eclipse: Red marks with message: The method
    trimToSize() is undefined for type List<String>
Alternatively, error messages from other IDEs.
```

(c) This method is not in the List interface, which specifies (along with supertypes Collection and Object) all methods available to variables of type List. It doesn't matter that the referenced object has this method. To call this method, you need a ref of type ArrayList. (The first sentence is sufficient as a hw answer.)

(d) Line of code to downcast from List¡String¿ to ArrayList¡String¿ to be able to call trimToSize: Here the List¡String¿ is held in variable array of TestArrayList.java (provided): Note the needed parentheses to require that the casted type be used with the method.

```
((ArrayList<String>)array).trimToSize();
```

Also acceptable, using raw type:

```
((ArrayList)array).trimToSize();      // This will cause a warning, but will work.
```