Homework 2

● Graded

Student

Giancarlos Marte

Total Points

91.5 / 107.5 pts

Question 1

Performance with scaling up JDK Collections

3.5 / 7.5 pts

– 0 pts Correct

– 7.5 pts not answered

– 1 pt part a not complete

✓ **– 2 pts** part b not correct

✓ **– 2 pts** part c not correct

– 2 pts part b not complete

– 2 pts part c not complete

– 2 pts part a not correct

– 1 pt part b not complete

– 2.5 pts part c not answered

– 7.5 pts CHEATING!

Question 2

O(1) Methods

5 / 5 pts

✓ **– 0 pts** Correct

– 5 pts not answered

– 4 pts not complete

– 0.5 pts not complete

– 2.5 pts part b

– 2 pts not complete

– 5 pts CHEATING!

Question 3

Choosing appropriate Collection classes to use

6 / 18 pts

- **– 0 pts** Correct
- **– 18 pts** not answered
- **– 1.5 pts** part 3 not complete
- **– 5 pts** part 1 not correct
- **– 3 pts** part 2 not complete
- **– 3 pts** part 3 not complete
- **– 7 pts** not complete
- **– 3 pts** pseudocode or describe
- **– 4.5 pts** time complexity
- **– 2 pts** not complete
- **– 2 pts** part 2 not complete
- ✓ **– 6 pts** part 1
- ✓ **– 6 pts** part 2
- **– 6 pts** part 3
- **– 1 pt** part 1 time complexity
- **– 1 pt** part 2 time complexity
- **– 1 pt** part 3 time complexity
- **– 5.5 pts** not complete
- **– 1 pt** part 1 not complete
- **– 12 pts** not complete
- **– 1 pt** part 2 not complete
- **– 2 pts** part 3 not complete
- **– 2.5 pts** part 2 not complete
- **– 2.5 pts** part 3 not complete
- **– 1 pt** part 3 not complete
- **– 15 pts** Can't read it!
- **– 18 pts** CHEATING!

Question 4

Map vs. ST

28 / 28 pts

✓ **– 0 pts** Correct

– 28 pts not answered

– 4 pts part 1 not complete

– 1 pt part 2 not complete

– 1 pt part 1 not complete

– 7 pts part 1 not complete

– 5 pts part 2

– 2 pts part 1 not complete

– 2 pts part 2 not complete

– 23 pts part 1

– 28 pts CHEATING!

Question 5

Equals/hashCode/compareTo for elements

28 / 28 pts

✓ **– 0 pts** Correct

– 28 pts not answered

– 2 pts part a, Point

– 2 pts part a, Date

– 2 pts part a, String

– 2 pts part a, Counter

– 8 pts part b, Point

– 4 pts part b, Date

– 8 pts part b, Counter

– 28 pts CHEATING!

Question 6

Set equality across HashSet and TreeSet

5 / 5 pts

✓ **– 0 pts** Correct

– 5 pts not answered

– 2 pts Incomplete / Has errors

– 5 pts CHEATING!

Question 7

Using LinkedList to implement Bag

10 / 10 pts

✓ **– 0 pts** Correct

– 5 pts Bag not answered

– 5 pts TestBag not answered

– 2 pts Bag incorrect/incomplete

– 2 pts TestBag incorrect/incomplete

– 10 pts CHEATING!

Question 8

More on Interfaces

6 / 6 pts

✓ **– 0 pts** Correct

– 6 pts not answered

– 2 pts a not correct

– 2 pts b not correct

– 2 pts c not correct

– 6 pts CHEATING!

Question assigned to the following page: 1

Giancarlos Marte
Homework 2 (2/24/21)

1) Performance with scaling up JDK Collections

a. hashmap search operation is O(N) for size N. This means it is constant, which then means that a search operation on a size of 2000 will also take 1ms on average.

b. Treemap has O(logN) for it's search operation on size N.
N = 1000
log(1000) //base 2
= 9.958 or 10ms
log(2000) = 10.9658 or 11ms
Treemap will take 11ms on size N = 2000.

c. LinkedList has O(N) for it's search operation on size N.
1 search for N = 1000 is 1000ms
1 search for N = 2000 is 2000ms
LinkedList will take 2000ms on size N.

Question assigned to the following page: 2

2) O(1) Methods

Map:

- size()
- isEmpty()
- containsKey(KeyType key)
- get(Ketype key),
- put(Ketype keyValueType value)
- remove(Keytype key)

Set:

- add(AnyType x)
- contains(Object x)
- size()
- isEmpty()
- remove(Object x)

Question assigned to the following page: 3

3) Choosing the appropriate Collections classes to use

1. I would use a List API and a LinkedList implementation. There is no need to search or find lines, just print the lines in reverse order. This means only the insert and descendingIterator methods are needed.

<u>Steps:</u>

1. create an empty linkedList<String> object.
2. read the file line by line using a scanner object.
    a. add each line into the linkedList as you read them.
3. make an iterator object, then set it equal to linkedList.descendingIterator().
4. print the iterator by looping through it.
    a. make sure to add \n.

- *This would be an O(N) time complexity where N is the number of lines in the file.*

2. I would use a hashset because it makes sure that there will be no repetition of words.

<u>Steps:</u>

1. create an empty hashSet<String> object.
2. read the file word by word using a scanner object.
    a. add each word into the hashSet as you read them.
3. make an iterator object and set it equal to hashSet.iterator().
4. loop through the iterator and print each element (with \n).

- *This would be an O(N) time complexity where N is the number of words.*

3. I would use a hashMap because you need to keep track of words(keys) and their counts(values). Order is also not important.

<u>Steps:</u>

1. create an empty hashMap<String, Integer> object.
2. read the word by word using a scanner object.
    a. put each word into the hashMap as you read them
        i. increase count by 1 if already seen (basically update key's value)
        ii. else make count = 1
3. make a set object equal to hashMap.keySet().
4. make an iterator object equal to that set object
5. while looping through the iterator, get each key's value using hashMap.get() method.
    a. print the key and it's value.

- *This would be O(N) time complexity where N is the number of words.*

4) Map vs. ST

a. (i) basic map:

- public int size()
- public boolean isEmpty()
- public boolean containsKey(KeyType key)
- public ValueType get(KeyType key)
- public ValueType put(KeyType key, ValueType value)
- public ValueType remove(Keytype key)
- public void clear()
- public Set<KeyType> keySet()
- public Collection<ValueType> values()

(ii) basic ST equivalents:

- public int size()
    - same
- public boolean isEmpty()
    - same
- public boolean containsKey(KeyType key)
    - public boolean contains(Key key)
        - It is just different in the name of the function and the name of the input, but everything else is the same.
- public ValueType get(KeyType key)
    - public Value get(Key key)
        - It is just different in the name of the function and the name of the input, but everything else is the same.
- public ValueType put(KeyType key, ValueType value)
    - public void put(Key key, Value value)
        - It has no return type and has a different name for the method and it's inputs.
- public ValueType remove(Keytype key)
    - public void delete(Key key)
        - It has no return type and has a different name for the method and it's inputs.
- public void clear()
    - No equivalent
- public Set<KeyType> keySet()
    - public Iterable<Key> keys()
        - This has a different return type and has a different name for the method and it's inputs.
- public Collection<ValueType> values()
    - No equivalent

Question assigned to the following page: 4

b. methods that don’t match up:

- public void clear()
    - workaround: you can make a loop of the ST size and then use the delete function to delete every key value pair.
- public Collection<ValueType> values()
    - workaround: you can use the iterable keys() method and then loop through the iterator. While doing so, call the get() method to get the value of each key. You can store these values in some type of list like a linkedList or an arrayList.

Question assigned to the following page: 5

5) Equals/hashCode/compareTo for HashSet and TreeSet elements.

a.

- point2D
    - HashSet: It cannot be used because there is no equal or hashcode method for this type of custom object.
    - TreeSet: It cannot be used because of the same reasons and it has no compareTo implementation.
- Date
    - HashSet: It cannot be used because there is no hashcode method for this type of custom object. However, if most of the dates are not repeatable then you might be able to use some parts of the dates themselves as hashcodes.
    - TreeSet: It cannot be used because of the same reasons and it has no compareTo or comparator implementation.
- String
    - HashSet: It can be used because it has both an equals and hashcode method.
    - TreeSet: It can be used because there is a compareTo method.
- Counter
    - HashSet: It cannot be used because there is no equals or hashcode method.
    - TreeSet: It cannot be used because there is no compareTo method or comparator.

b.

point2D

```
public boolean equals(Object x) {
    if (this == x) return true;
    if (x == null) return false;
    if (this.getClass() != x.getClass()) return false;
    Point2D that = (Point2D) x;
    if (this.x() != that.x()) return false;
    if (this.y() != that.y()) return false;
    if (this.r() != that.r()) return false;
    if (this.theta() != that.theta()) return false;
    return true;
}
```

Question assigned to the following page: 5

```
public int hashCode() {
        int prime = 7;
        int x = this.x().hashCode;
        int y = this.y().hashCode;
        return prime + (x * y);
```

<u>Date</u>

```
public int hashCode() {
        int prime = 7;
        int m = this.month().hashCode;
        int d = this.day().hashCode;
        int y = this.year().hashCode;
        return prime + (m * d * y);
```

<u>Counter</u>

```
public boolean equals(Object x) {
        if (this == x) return true;
        if (x == null) return false;
        if (this.getClass() != x.getClass()) return false;
        Counter that = (Counter) x;
        if (this.name != that.name) return false;
        if (this.tally() != that.tally()) return false;
        return true;
}

public int hashCode() {
        int prime = 7;
        int c = this.tally();
        if (this.name != null && c != 0) {
                return prime + (name.hashCode * c);
        }
        return prime + (c * 31);
}
```

Question assigned to the following page: 6

6) Set Equality across HashSet and TreeSet

```
public static void main(String[] args) {
    Set<String> setH = new HashSet<>();
    Set<String> setT = new TreeSet<>();

    String a = "apple";
    String p = "pear";

    setH.add(a);
    setH.add(p);
    setT.add(a);
    setT.add(p);

    System.out.println(setH.equals(setT));
}
```

7) Using LinkedList to implement a specialized list

TestBag.java

```java
import java.util.Iterator;

public class TestBag {
   public static void main(String[] args) {
       Bag<String> bag = new Bag<>();
       String a = "apple";
       String p = "pear";
       bag.add(a);
       bag.add(p);
       bag.add(a);
       int count = 0;
       Iterator<String> iterator = bag.iterator();
       while (iterator.hasNext()) {
           if (iterator.next().equals(a)) {
               count++;
           }
       }
       System.out.println("number of apples in bag: " + count);
   }
}
```

Question assigned to the following page: 7

Bag.java

```java
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

public class Bag<Item> {
   private List<Item> list;

   public Bag() {
       list = new LinkedList<>();
   }

   public void add(Item x) {
       list.add(x);
   }

   public Iterator<Item> iterator() {
       return list.iterator();
   }
}
```

Question assigned to the following page: 8

8) More on Interfaces

a. public void trimToSize()

b. The new version has an error while compiling:

```
java: cannot find symbol
  symbol:   method trimToSize()
  location: variable array of type java.util.List<java.lang.String>
```

c. It failed because ArrayList is a subtype of List. Since List does not support trimToSize() method, trying to access it will give you an error.

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