CS310 MIDTERM EXAM, NOV. 2, 2020 NAME: **Jimmy Tran**

University ID: **01890657**

Open books, handouts, solutions, including online materials, JDK API. No person-to-person communication is allowed, except to ask me questions by email to elizabeth.oneil@umb.edu. Please attend class and have audio and video on. Submit this exam to Gradescope by 8:30pm. If you can't submit it for some reason, email it to me in pdf format. Please check your email occasionally in case I need to provide a correction. Show all work in the provided areas, expanding them if needed. Each problem is worth 20 points, for a total of 100 points.

1. Choosing the right JDK Collections class. For the following scenarios, choose the appropriate JDK Collections classes to use from those we studied in detail: List, Set, Map. Use the simplest class (first on the list here) that has the power you need.

a. You are a grader and decide to use a program to help you manipulate homework scores for students. Each student has a score (integer) for each of "hw1", "hw2", "pa1", etc. How can you hold all the scores for one student so it is easy (one method call) to find out the score for hw1 or pa1, or whatever?

JDK Collections API concrete class, including appropriate generic types:
Map<String, Integer>
- In this case the String type will contain the assignments "hw1", "hw2", and "pa1", and he integer type will contain the scores of those assignments

How to create one such (empty) container:
Map<String, Integer> studentGradeContainer = new HashMap<String, Integer>

b. In the same scenario, you realize you want to control the exact order of display of data, so that "hw1" is always first, "hw2" is second, etc. How can you hold just this particular ordering of these strings? (Don't worry about holding the scores here.)

JDK Collections API concrete class, including appropriate generic types:
List<String>
How to create one such (empty) container:
List<String> allTheAssignments = new ArrayList<String>();

c. The students themselves are ID'd by StudentID, an integer that fits in 32 bits. For the last programming assignment in the class, you have set up teams of students. How should you represent a single programming team of students?

JDK Collections API concrete class, including appropriate generic types:
Set<Integer>
How to create one such (empty) container:
Set<Integer> teamsOfStudents = new HashSet<Integer>();

d. Suppose each team has chosen a string name to identify it. How can you hold not only all these team names but also the membership of each named team?

JDK Collections API concrete class, including appropriate generic types:
Map<String, Set<Integer> >
How to create one such (empty) container:
Map<String, Set<Integer> > teamWithMemberships = new Map<String, Set<Integer>> ();

2. Big-Oh analysis. Consider this Java method:

```
public static int f(int n)
{
        int i, j, sum = 0;

        for (i=0; i < n; i++)  /* loop 1 */
            sum += i*i;
        for (i=0; i < n; i++)  /* loop 2 */
            for (j=0; j < 3; j++)  /* loop 3 */
                sum += i*j;
        return sum;
  }
```

Let T(n) be the running time of this method in terms of its argument n. Analyze the contributions to T(n) from:
a. Loop 1
For loop one the run time complexity of this would be in linear time. It is in linear time because we are iterating through the loop in O(n) times since it is iterating through it until "i" reaches the value of n. Therefore, the time that the iteration will go through is based on the size of n, so O(n) is the time complexity. [T(n) = O(n)]

b. Loop3, for a single value of i.
The value of "i" will be iterating through O(n) times. This is true because when we are iterating **i's** value is only changing whenever we iterate through the first outer loop, and incrementing the value of "i" by 1. So it depends on how many iterations there are, and that is dependent on the value of n, so it is iterating and changing the value n times. So O(n), and therefore linear runtime. [T(n) = O(n)], n being the value of the variable n

c. Loop2, the whole double loop.
The double loop will have a O(3n) complexity since the outer loop will be O(n), n being the size of the n. T(n) = O(n) * O(3) => T(n) = O(3n). Therefore this will also be a linear runtime. And O(3n) can be simplified to O(n), which is linear runtime, since 3n and n won't have a significant difference in runtime.

d. The whole function.
The whole function will be T(N) = O(N) + O(N) * O(3) ⇒ O(3N) + O(N) ⇒ O(3N) ⇒ O(N)
Overall, the whole function will run in O(N) / linear time because if we calculate the first loop, it is O(N) and the second double loop will also be O(N) simplified. If we combine both O(N) and O(N) it will still be O(N), since those two operations combined won't make any significant operations. Therefore it is linear runtime overall.

3. What JDK Collections objects can and can't do. Which of the following things can be done by pure use of the API as defined in the Java Collections? If you can do it, outline how, else say that it is impossible.

a. The program has set up a Map from String to Integer, i.e. Map<String, Integer>. Now it wants to make two different keys, "x" and "y" map to the same Integer 6.
Yes it is possible to do this because there is allowed duplicates of Integers but not duplicates of keys. In this case, both keys are not duplicates, therefore it is valid that keys "x", and keys "y" can point to the same integer 6. For example, let's take this scenario into consideration..

```
// we create a map
Map<String, Integer> theMap = new HashMap<String,Integer>();
// then we add x and y to it with the values 6
theMap.put("x",6);
theMap.put("y",6);

// when we print this map out it will contain

{
        "X" : 6,
        "Y" : 6
}
```

b. The program has set up a Map from String to Integer. Now it wants to make one key, "x", map to two different Integers 6 and 7.

No, this is not possible because in this case it is saying that "x" is pointing to two values, in order words there are two keys that are the same, pointing to two different values (6 and 7), but the rules of HashMaps do not allow you to have duplicates of keys.

For example, let's take this scenario into consideration..

```
// we create a map
Map<String, Integer> theMap = new HashMap<String,Integer>();
// then we add x both times to point to both values 6 and 7
theMap.put("x",6);
theMap.put("x",7);
```

// when we print this map out it will contain X key pointing to value 7 since, there is no duplication allowed, the second time we put X in it will override the first value

```
{
        "X" : 7
}
```

c. A program has added 10 elements to a Set<String> and no longer has individual references to them. Now it wants to determine which element was the first one added to the set.
This is not possible because when you are adding elements to a concrete class that is part Set interface, it does not provide any orderings or the elements when added. Therefore you cannot keep track of it and determine which element was the first one added to the set.

d. The program has set up a List of Strings, List<String>, and added several elements to it. Now it wants to determine the string that comes first in dictionary order.
We can't really determine if it is in alphabetical order with the list unless we perform an operation on it. (Can't be determined by one function call). In order to determine we have to iterate through the list and use the compareTo method to check which one is greater than (alphabetically) and keep that in a variable record. After that we will return that variable.

4. A Map/Set App. You are helping the post office automate the sorting of mail into postal routes, corresponding to the actual bags of mail given to letter carriers. The address scanning machines pick up address strings off the letter's address, for example the zip code, that each specify a certain set of postal routes. The crucial construct you need is a Map from these address strings to Sets of postal route numbers (themselves integers). Here is an example simplified Map:

"Cambridge" --> { 201, 202, 203, 404, 405 }
"Boston" --> {500, 501, 502, 503, 504}
"02138" --> { 201, 203} (a zipcode entirely in Cambridge)
"Ware St." --> {203} (a tiny street in Cambridge)

a. Give the code for constructing an empty instance of this Map, using appropriate generic types:

```
Map<String, Set<Integer>> instanceOfThisMap = new Map<String, Set<Integer>>();
```

b. Give the code (3 lines of code, for example) for adding the given information for "Ware St."

```
// create an instance of a HashSet
Set<Integer> postalSets = new HashSet<Integer>();
```

```
// add the integer 203 to the postalSets
postalSets.add(203);

// add the "Ware St" as a key and add the postalSets as a value to the map
instanceOfThisMap.put("Ware St.", postalSets);
```

c. For address strings with O(1) postal routes, what is the performance of looking up the address string in the map? Discuss any dependence on what Map implementation is chosen, that is, HashMap vs. TreeMap. What is N here?

The performance looking up the map will O(1) for hashmap and O(logN) for the treemap since the tree map lookup is similarly to a binary search. N will be the size of the map. Depending on the size of the map, hashmap will be faster than tree map if the size is bigger

d. If the machine picks up two strings off the same envelope, the Map provides 2 sets of route numbers. How should we combine these two sets to get an answer set of route numbers consistent with both strings? In particular, what Set method should we use?

5. Interfaces. Consider the Stack API from pg. 121:

public class Stack<Item> implements Iterable<Item>		
	Stack()	create an empty queue
void	push(Item item)	add an item
Item	pop()	remove the most recently added item
boolean	isEmpty()	is the stack empty?
int	size()	number of items in the stack

a. Write a Java interface for the methods of this API, with name StackIf (If for interface, saving "Stack" for the implementing class). Start it with "public interface Stack<Item> extends Iterable<Item>" to handle the Iterable part. This will make StackIf a subtype of Iterable<Item>.

```
public interface Stack<Item> extends Iterable<Item>{
        Void push(Item item);
        Item pop();
        Boolean isEmpty();
        Int size();
}
```

b. What can you add to the top line of the source for class Stack of page 149 to say it complies with this interface?
We have to add the interable() method in order for it to comply with the interface.

c. Recall our Bag implementation using LinkedList in homework 2. That approach is possible here too, to implement Stack.java using LinkedList or ArrayList. The usual way is to push onto the end of the list and pop from there too. What List method(s) would you use for push (just give method names)? What List method(s) are needed for pop? Include any methods from any iterator obtained from the List.