Family Name: ..... **Other Names:** .....

Student ID: ..... **Signature** .....

COMP 103 : Test 1

2021, Jan 8 ** WITH SOLUTIONS **

Instructions

- Time allowed: **50 minutes**
- Attempt **all** questions. There are 30 marks in total.
- Write your answers in this test paper and hand in all sheets.
- If you think some question is unclear, ask for clarification.
- Brief Java documentation is provided with the test
- This test contributes 10% of your final grade
- You may use dictionaries.
- You may write notes and working on this paper, but make sure your answers are clear.

Questions	Marks	
1. Properties of Collections	[12]	
2. Choosing a Collection	[8]	
3. Using Collections	[10]	
	TOTAL:	

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Question 1. Properties of Collections **[12 marks]**

For these questions, circle "true" or "false" for each property. (Note some properties may be true for more than one type.)

(a) **[3 marks]** For a List, state whether each property is true or false.

true/**false**	:	The collection cannot contain any duplicate items
true/false	:	An item can be removed by specifying its index
true/**false**	:	Items can only be removed from one end.
true/false	:	Items can be added at a specified position

(b) **[3 marks]** For a Set, state whether each property is true or false.

true/false	:	The collection cannot contain any duplicate items
true/**false**	:	The order of items in the collection is important
true/**false**	:	Items are added and removed from opposite ends.
true/false	:	The time an item is added does not affect when it is removed

(c) **[3 marks]** For a Map, state whether each property is true or false.

true/**false**	:	The order of items in the collection is important
true/false	:	To remove an item, you must specify its key
true/false	:	Items must be added with a key
true/**false**	:	Items can be added at a specified position

(d) **[3 marks]** For a Stack, state whether each property is true or false.

true/**false**	:	The first item added is the first item to be removed
true/**false**	:	Items can be added at a specified position
true/**false**	:	The order of items in the collection is unimportant
true/false	:	The first item added is the last item to be removed

Marking:
3 marks for getting all four properties right;
2 marks for getting three properties right;
1 mark for getting two properties right;
0 marks for less than two right.

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Question 2. Collection Types **[8 marks]**

(a) **[2 marks]** Suppose that you are writing a program to keep track of the stock in your retail store. You need to use a collection that can link the product code of an item (a string), to the amount of that item that you have in stock.

Complete the declaration of the stockCollection field below by providing the appropriate collection type.

```
private Map<String, Integer> stockCollection;
```

(b) **[2 marks]** In a popular collectible card game, the actions that players perform during a turn are added to a collection in the order they happen. The effects of those actions are then resolved in reverse order, with the last action taking effect first.

Is this collection a **Stack** or a **Queue**?

Stack

(c) **[4 marks]** Suppose you are writing a piece of software to simulate the a checkout at a supermarket. You have a Basket that holds all of the items the customer is purchasing, and a ConveyorBelt that the items are placed on.

1. A Basket can hold any number of items, and is allowed to contain duplicates. Items can be removed from anywhere in the basket and added to the ConveyorBelt.
2. Items are added to the end of the ConveyorBelt, and move up to the front, where they are scanned and bagged.

Which collection types would you use to represent the Basket and ConveyorBelt? Briefly explain why.

The basket could be a Bag/Collection or a List. We need to be able to access items stored anywhere inside it.
The conveyor belt should be a queue, as we add to the end and remove from the front.

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Question 3. Using Collections **[10 marks]**

Complete the following analyseText(...) method which is given a List of words (Strings) and counts the number times each word appears in the text. It also creates a Set containing all the long words (at least 10 letters) that appear in the text.

```
private Map<String, Integer> wordCounts = new HashMap<String, Integer>();
private Set<String> longWords = new HashSet<String>();


public void analyseText( List<String> words){

    for (String word : words){
        if (wordCounts.containsKey(word)){
            wordCounts.put(word, wordCounts.get(word)+1);
        }
        else {
            wordCounts.put(word, 1);
        }
        if (word.length()>=10) longWords.add(word);
    }









}
```

* * * * * * * * * * * * * * *

Documentation

Brief, simplified specifications of some relevant Java collection types and classes.

Note: E stands for the type of the item in the collection.

```
interface Collection<E>
   public boolean isEmpty()                    // cost: O(1) for all standard collection classes
   public int size()                           // cost: O(1) for all standard collection classes
   public void clear()
   public boolean add(E item)
   public boolean contains(Object item)
   public boolean remove(Object element)

interface List<E> extends Collection<E>
   // Implementations: ArrayList
   public boolean isEmpty()
   public int size()
   public void clear()
   public E get(int index)                     // cost: O(1)
   public E set(int index, E element)          // cost: O(1)
   public boolean contains(Object item)        // cost: O(n)
   public void add(int index, E element)       // cost: O(n) (unless index is close to the end.)
   public E remove(int index)                  // cost: O(n) (unless index is close to the end.)
   public boolean remove(Object element)       // cost: O(n)
   public void sort((E e1, E e2)->{...});      // cost: O(n log(n)), but O(n) if almost sorted

interface Set extends Collection<E>
   // Implementations: HashSet, TreeSet
   public boolean isEmpty()
   public int size()
   public void clear()
   public boolean add(E item)                  // cost: O(1) for HashSet, O(log(n)) for TreeSet
   public boolean contains(Object item)        // cost: O(1) for HashSet, O(log(n)) for TreeSet
   public boolean remove(Object element)       // cost: O(1) for HashSet, O(log(n)) for TreeSet

interface Map<K, V>
   // Implementations: HashMap, TreeMap
   public V get(K key)                 // cost: O(1) for HashMap, O(log(n)) for TreeMap
   public V put(K key, V value)        // cost: O(1) for HashMap, O(log(n)) for TreeMap
   public V remove(K key)              // cost: O(1) for HashMap, O(log(n)) for TreeMap
   public boolean containsKey(K key)   // cost: O(1) for HashMap, O(log(n)) for TreeMap
   public Set<K> keySet()              // cost: O(1)
   public Collection<V> values()       // cost: O(1)
   // (get returns null if the key is not present)
   // (get put and remove return the old value, if any)
```

```
interface Queue<E> extends Collection<E>
   // Implementations: ArrayDeque, LinkedList, PriorityQueue
   public boolean isEmpty()
   public int size()
   public void clear()
   public E peek()                  // cost: O(1) for ArrayDeque, LinkedList, O(1) for PriorityQueue
   public E poll()                  // cost: O(1) for ArrayDeque, LinkedList, O(log(n)) for PriorityQueue
   public boolean offer(E element)  // cost: O(1) for ArrayDeque, LinkedList, O(log(n)) for PriorityQueue
   // (peek and poll return null if the queue is empty)

class Stack<E> implements Collection<E>
   public boolean isEmpty()
   public int size()
   pubic void clear()
   public E peek()                  // cost: O(1)
   public E pop()                   // cost: O(1)
   public E push(E element)         // cost: O(1)
   // (peek and pop return null if the queue is empty)


class Collections
   public static void sort(List<E> list);                          // cost: O(n log(n)), but O(n) if almost sort
   public static void sort(List<E> list, (E e1, E e2)->{...});     // cost: O(n log(n)), but O(n) if almost sort
   public static void swap(List<E> list, int i, int j);            // cost: O(1)
   public static void reverse(List<E> list);                       // cost: O(n)
   public static void shuffle(List<E> list);                       // cost: O(n)


interface Comparable<E> // All Comparable objects have a compareTo method:
   public int compareTo(E other);
          // returns
          //     -ve if this comes before other;
          //     +ve if this comes after other,
          //       0 if this and other are the same
```