

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

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

## COMP 103 : Test

2020, Jan 8 **\*\* WITH SOLUTIONS \*\***

### Instructions

- Time allowed: **50 minutes**
- Attempt **all** questions. There are 50 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 15% of your final grade  
(But your mark will be increased to your exam mark if that is higher.)
- 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]	<input type="text"/>
2. Using Queues in Simulations	[13]	<input type="text"/>
3. Using Collections	[10]	<input type="text"/>
4. Cost of Algorithms	[7]	<input type="text"/>
5. Recursion	[8]	<input type="text"/>
	TOTAL:	<input type="text"/>

**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) **[4 marks]** For a Queue, state whether each property is true or false.

<input checked="" type="checkbox"/> true / <input type="checkbox"/> false	: The order of items in the collection is important
<input type="checkbox"/> true / <input checked="" type="checkbox"/> false	: Items must have a natural ordering to be stored in the collection.
<input type="checkbox"/> true / <input checked="" type="checkbox"/> false	: The first item added is the last item to be removed
<input checked="" type="checkbox"/> true / <input type="checkbox"/> false	: The collection can contain duplicate items

(b) **[4 marks]** For a Map, state whether each property is true or false.

<input type="checkbox"/> true / <input checked="" type="checkbox"/> false	: The order of values in the collection is important
<input checked="" type="checkbox"/> true / <input type="checkbox"/> false	: To remove a value, you must specify its key
<input type="checkbox"/> true / <input checked="" type="checkbox"/> false	: The collection cannot contain any duplicate values
<input checked="" type="checkbox"/> true / <input type="checkbox"/> false	: Items in the collection are stored as key-value pairs

(c) **[4 marks]** For a Set, state whether each property is true or false.

<input type="checkbox"/> true / <input checked="" type="checkbox"/> false	: An item can be only be removed by specifying its index
<input type="checkbox"/> true / <input checked="" type="checkbox"/> false	: The order of items in the collection is important
<input checked="" type="checkbox"/> true / <input type="checkbox"/> false	: The collection cannot contain any duplicate items
<input type="checkbox"/> true / <input checked="" type="checkbox"/> false	: Items can be added at a specified position

**Marking:**

4 marks for getting all four properties right;  
2.5 marks for getting three properties right;  
0 marks for less than three 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. Using Queues in Simulations****[13 marks]**

For this question, you are to extend a simple simulation program.

The program simulates the creation and delivery of pizza for a fast-food outlet. The outlet has two queues: kitchen and delivery. The kitchen queue holds all of the orders that need to be cooked. Once an order is cooked, it is moved to the delivery queue and delivered to the customer.

At each tick:

- a new Order may be created and added to the kitchen queue.
- the Orders at the heads of both queues are advanced one tick
- if the Order at the head of the kitchen queue is complete, it is removed from the queue and added to the delivery queue.
- if the Order at the head of the delivery queue is complete, it is removed from the queue.
- the queues are re-displayed.

The owner of the outlet wants to simulate what would happen if they hired more delivery drivers. You must extend the program to allow for multiple delivery drivers, each with their own queue.

- There can be any number of delivery drivers, so the queues should be stored in a List.
- When an order is finished in the kitchen, it should be added to the shortest delivery queue.
  - For convenience, a separate method (`getShortestQueue()`) will be used to find the shortest queue.
- On every time tick, the first job on every queue should be advanced.

(Question 2 continued on next page)

**(Question 2 continued)**

Here is the one-driver program:

---

```

public class Simulation{
    private boolean running = false;

    private Queue<Order> kitchen;
    private Queue<Order> delivery;

    public void setup(){
        running = false;
        kitchen = new ArrayDeque<Order>();
        delivery = new ArrayDeque<Order>();
    }

    public void run(){
        int time = 0;
        running = true;
        while (running){
            time++;
            if (Math.random()<0.2){
                Order o = new Order();
                kitchen . offer (o);
            }
            if (! kitchen . isEmpty()){
                Order topOrder = kitchen.peek();
                topOrder.advanceCookingByOneTick();
                if (topOrder.isCooked()){
                    delivery . offer (kitchen . poll ());
                }
            }
            if (! delivery . isEmpty()){
                Order topOrder = delivery . peek();
                topOrder.advanceDeliveryByOneTick();
                if (topOrder. isDelivered ()){
                    delivery . poll ();
                }
            }
            display (kitchen , 1); // redisplay queue on row 1 of the window
            display ( delivery , 2); // redisplay queue on row 2 of the window
        }
    }
}

```

---

(Question 2 continued on next page)

**(Question 2 continued)**

- (a) [5 marks] Complete the `getShortestQueue()` method that will find and return the delivery queue with the least number of orders

```

public class Simulation{
    public static final int NUM_DELIVERY_DRIVERS = 5;
    private boolean running = true;

    private Queue<Order> kitchen ;
    private List<Queue<Order>> deliveryQueues;

    public void setup() {
        running = false;
        kitchen = new ArrayDeque<Order>();
        deliveryQueues = new ArrayList<Queue<Order>>(); //new
        for (int i =0; i < NUM_DELIVERY_DRIVERS; i++){ //new
            deliveryQueues.add(new ArrayDeque<Order>()); //new
        }
    }

    private Queue<Order> shortestDeliveryQueue() {
        Queue<Order> shortest = deliveryQueues.get(0);
        for(int i = 1; i < deliveryQueues.size (); i++) {
            if(deliveryQueues.get(i).size () < shortest.size ())
                shortest = deliveryQueues.get(i);
        }
        return shortest ;
    }
}

```

(Question 2 continued on next page)

**(Question 2 continued)**

(b) [8 marks] Complete the run method for the simulation with multiple drivers

```

public void run() {
    int time = 0;
    running = true;
    while (running){
        time++;
        if (Math.random()<0.2){
            Order o = new Order();
            kitchen . offer (o);
        }
        if (! kitchen .isEmpty()){
            Order topOrder = kitchen.peek();
            topOrder.advanceCookingByOneTick();
            if (topOrder.isCooked()){
                shortestDeliveryQueue (). offer ( kitchen . poll ());
            }
        }
        display (kitchen , 1); // redisplay queue on row 1 of the window
        int row = 2;
        for (Queue<Order> queue : deliveryQueues){ //new
            if (!queue.isEmpty()){
                Order topOrder = queue.peek();
                topOrder.advanceJobByOneTick();
                if (topOrder.isCompleted()){
                    queue.poll ();
                }
            }
            display (queue, row++); //new
        }
    }
}

```

```

}
}

```



**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` of verbs (action words) and nouns (name words) that appear in the text.

You can assume that there are methods `boolean isVerb(String s)` and `boolean isNoun(String s)` that will return `true` if the provided `String` is a verb or a noun.

The method should be efficient: if the `List` has  $n$  words, then the method should run in  $O(n)$  time.

```
Set<String> verbs;
Set<String> nouns;
Map<String, Integer> wordCounts;

public void analyseText( List<String> words){
    verbs = new HashSet<String>();
    nouns = new HashSet<String>();
    wordCounts = new HashMap<String, Integer>();
    for (String word : words){
        if (wordCounts.containsKey(word)){
            wordCounts.put(word, wordCounts.get(word)+1);
        }
        else {
            wordCounts.put(word, 1);
        }
        if(isNoun(word)) nouns.add(word);
        if(isVerb(word)) verbs.add(word);
    }
}
```

}

**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 4. Cost of Algorithms****[7 marks]**

What is the big-O cost of the following method that uses a PriorityQueue to sort a list?

- write the big-O cost of performing each line with a comment
- write the number of times each of the lines will be performed,
- write the total cost of the algorithm (big-O) at the bottom of the box

```

public void sortList ( List<String> words){
    Queue<String> queue = new PriorityQueue<String>();

    for( String word : words) {
        queue.offer (word);                // cost= O( log n )   times= n
    }
    words.clear ();                        // cost= O( 1 )     times= 1
    while (!queue.isEmpty()) {
        words.add( queue.poll () );        // cost= O( log n )   times= n
    }
}
Total Cost = O( n log n )

```

**SPARE PAGE FOR EXTRA ANSWERS**

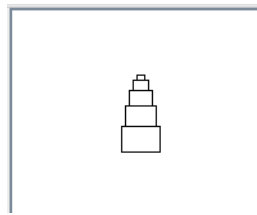
Cross out rough working that you do not want marked.  
Specify the question number for work that you do want marked.

**Question 5. Recursion****[8 marks]**

(a) **[4 marks]** Complete the following recursive `drawTower(x, y, width)` method to draw a "tower" made of stacked rectangles.

- You must use a "first-and-rest" recursion: `drawTower(...)` should draw the first rectangle and then call itself recursively to draw the rest of the rectangles.
- The first rectangle should be centered at  $(x,y)$  with the provided width. The rectangle's height should be  $2/3 * \text{width}$ .
- The next rectangle should be height-2 pixels above, and should be 6 pixels smaller in width.
- It should continue to draw rectangles as long as the width is larger than 3.

For example, `drawTower(100, 300, 30)` should output the following:



```

public void drawTower(double x, double y, double width){
    if(width >3) {
        double height = width * 2 / 3;
        UI.drawRect(x-width/2,y-height/2,width,height);
        drawTower(x,y-(height-2),width-6);
    }
}

```

Hint: you can use the method `UI.drawRect(left, top, width, height)` to draw a rectangle.

(Question 5 continued on next page)

**(Question 5 continued)**

(b) [4 marks] Suppose the variable letters contains the following List of 6 Strings:

"A"	"B"	"C"	"D"	"E"	"F"
0	1	2	3	4	5

What will printList(letters, 0, 5) print out?

---

```

/** print the letters in the list */
public void printList ( List<String> letters , int start , int end){
    if ( start <= end) {
        int mid = (start + end)/2;
        Ul. print ( letters .get(mid));
        printList ( letters , start , mid-1);
        printList ( letters , mid+1, end);
    }
}

```

---

Answer:

C A B E D F

\*\*\*\*\*

## 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)
```

**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()
    public 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 void sort (List<E> list); // cost = O(n log(n)), but O(n) if almost sorted
    public void sort (List<E> list, (E e1, E e2)->{...}); // cost = O(n log(n)), but O(n) if almost sorted
    public void swap(List<E> list, int i, int j); // cost = O(1)
    public void reverse (List<E> list); // cost = O(n)
    public 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
```

---