

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

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

## COMP 103 : Test 1

2022, Jan 17 \*\* 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.

<input checked="" type="checkbox"/>	true	/	<input type="checkbox"/>	false	:	The time an item is added does not affect when it is removed
<input checked="" type="checkbox"/>	true	/	<input type="checkbox"/>	false	:	An item can be removed by specifying its index
<input type="checkbox"/>	true	/	<input checked="" type="checkbox"/>	false	:	An item can be removed by specifying its key
<input checked="" type="checkbox"/>	true	/	<input type="checkbox"/>	false	:	Items can be added at a specified position

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

<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 order of items in the collection is important
<input checked="" type="checkbox"/>	true	/	<input type="checkbox"/>	false	:	Items are added and removed from opposite ends.
<input type="checkbox"/>	true	/	<input checked="" type="checkbox"/>	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.

<input type="checkbox"/>	true	/	<input checked="" type="checkbox"/>	false	:	An item can be removed by specifying its index
<input checked="" type="checkbox"/>	true	/	<input type="checkbox"/>	false	:	An item can be removed by specifying its key
<input checked="" type="checkbox"/>	true	/	<input type="checkbox"/>	false	:	Items must be added with a key
<input type="checkbox"/>	true	/	<input checked="" type="checkbox"/>	false	:	The collection cannot contain any duplicate values

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

<input type="checkbox"/>	true	/	<input checked="" type="checkbox"/>	false	:	The first item added is the first item to be removed
<input type="checkbox"/>	true	/	<input checked="" type="checkbox"/>	false	:	Items can be added at a specified position
<input checked="" type="checkbox"/>	true	/	<input type="checkbox"/>	false	:	The order of items in the collection is unimportant
<input checked="" type="checkbox"/>	true	/	<input type="checkbox"/>	false	:	The collection cannot contain any duplicate items

**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 a collection of gender neutral names from around the world for the purposes of writing test questions. Each name is a String, and the following facts are true:

- You don't want duplicate names in your collection
- You don't care about the order they're stored in

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

```
private ___Set___ < ___String___ > nameCollection;
```

(b) [2 marks] A recipe book app needs to keep a collection of the ingredients required for each recipe. Each ingredient has a name and a quantity.

Is this collection best represented using a **List** or a **Map**?

Map

(c) [4 marks] Suppose you are writing a piece of software to handle book reservations at your local library. You have two collections:

- a Shelf collection that contains all of the books that have been reserved. The library has multiple copies of books, and the shelf is not kept in any particular order.
- a Reservations collection for each book that contains a record of who currently has the book reserved. Reservations need to be processed first-in, first-out.

Which collection types would you use to represent the Shelf and Reservations? Briefly explain why.

The shelf could be a Bag/Collection or a List. We need to be able to access items stored anywhere inside it, and permit duplicates.  
Reservations 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 `analyseCards(...)` method which is given a List of cards (type `Card` shown below) and counts the number times each card *kind* appears in the text. It also creates a Set containing all the cards that are Aces (have value 14).

```
class Card {
    private String kind; // Can be: "SPADES", "HEARTS", "CLUBS", "DIAMONDS"
    private int value; // 2, ... , 10, 11 (J), 12 (Q), 13 (K), 14 (A)
    public String getKind() { return this.kind; }
    public int getValue() { return this.value; }
}

private Map<String, Integer> cardCounts = new HashMap<String, Integer>();
private Set<Card> aces = new HashSet<Card>();

public void analyseCards( List<Card> cards){

    for (Card card : cards){
        if (cardCounts.containsKey(card.getKind())){
            cardCounts.put(card.getKind(), cardCounts.get(card.getKind())+1);
        }
        else {
            cardCounts.put(card.getKind(), 1);
        }
        if (card.getValue()==14) aces.add(card);
    }
}
```

Student ID: .....

\*\*\*\*\*



## 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()
    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 static void sort (List<E> list); // cost: O(n log(n)), but O(n) if almost sorted
    public static void sort (List<E> list, (E e1, E e2) -> {...}); // cost: O(n log(n)), but O(n) if almost sorted
    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
```

---