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

ID Number: ..... **Signature** .....

Model Solutions
COMP 103: Mid-term Test

21st of August, 2014

Instructions

- Time allowed: **50 minutes**
- There are 50 marks in total.
- Answer **all** the questions.
- Write your answers in the boxes in this test paper and hand in all sheets.
- Brief Java documentation is supplied on the last page.
- This test will be converted to 20% of your final grade
  (but your mark will be boosted up to your exam mark if that is higher.)
- You may use paper translation dictionaries.
- You may write notes and working on this paper, but make sure it is clear where your answers are.

Questions	**Marks**	
1. Collections	[10]	
2. Programming with Collections	[30]	
3. Costs, sorts, and recursion	[10]	
TOTAL:	50	

Question 1. Collections [10 marks]

How would you represent each of the following situations, using Java collections? There may be more than one reasonable answer so justify your choice.

Note: You can assume there are appropriate classes defined for Passenger, Car, Customer and so on. Some may involve more than one Collection type.
An example of an answer might be "With a Stack of Maps that are from Integers to Strings".

(a) [2 marks] Passengers riding on a ferris wheel: on a ferris wheel the first Passenger to embark is always the first to disembark.

Collection Type: A queue of Passengers.

(b) [2 marks] A collection recording the IDs and family names of the students in a course. Upon request, the program needs to be able to provide the name of the student having a certain ID.

Collection Type: Use a Map from Int (ID) to String (name).

(c) [2 marks] The Cars waiting in lines for petrol, at a large petrol station with numbered petrol pumps. When information is requested about a given pump, the program needs to be able to provide information about the Car that is currently filling up at that pump, and how many Cars are in the line.

Collection Type: Use a Map from Integers (pump id) to Queue of Car objects. OR... Array of Queues.

(d) [2 marks] The New Zealand Team members (Persons) attending the Commonwealth games, across all sports. Each sport is to be identified by its unique String (*eg:* "Hockey"). Some sports have just 1 person in the team, but others have more.

Collection Type: Map from String (sport) to Sets of Persons

(e) [2 marks] All the individual items (of type StockItem) in a "$2 shop" - these shops contain large numbers of essentially identical items, such as pingpong balls, plastic coins, joke stickers, etc.

Collection Type: Bag of StockItems. (another interpretation might be Map from StockItems to numbers in stock)

Question 2. Programming with Collections [30 marks]

(a) [2 marks] What will the following code print?

```
public static void main(String[] a) {
    Stack<String> ss = new Stack<String>();
    ss.push("W");
    ss.push("X");
    ss.push("Y");
    String pp = ss.peek();
    pp = ss.pop();
    ss.push("Z");
    while (!ss.isEmpty())
        UI.print(ss.pop());
}
```

ZXW

(b) [4 marks] Here is a set:

```
Set <Car> locals = new ArraySet <Car> ();
```

and here is code to go through the set using a for each loop, calling a method driveAround on each element in turn:

```
for (Car c : locals)
    c.driveAround();
```

Write code that uses the Set's *Iterator* to do the same thing.

```
Iterator <Car> iter = locals.iterator();

while ( iter.hasNext())
    iter.next().driveAround();
```

.

(c) [2 marks] By circling the number at the left, indicate clearly which of the following are **valid**:

1. `Set<Shape> mycollection = new Set<Shape> ();`
2. `Set<Shape> mycollection = new HashSet<Shape> ();`
3. `HashSet<Shape> mycollection = new Set<Shape> ();`
4. `List <Shape> mycollection = new HashSet<Shape> ();`

Only (2) is valid

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.

(d) [8 marks] Suppose you are working on a program that deals with two *List*s of String objects that are single words. Each of these *List*s could have repetitions of some words.

Write a method called "wordsInBoth" that takes the two lists as arguments. The method should detect those words that occur in **both** the lists, and

- remove all those words from *both the lists*, and
- return the *Set* of removed words.

Example: if the lists being passed in were

```
list1:  dog, cat, carrot, hamster, weasel, canary, budgie, hamster
list2:  carrot, grass, hamster
```

then the method should update them to be

```
list1:  dog, cat, weasel, canary, budgie
list2:  grass
```

and then return the set

```
        carrot, hamster
```

```
public Set<String> wordsInBoth (List<String> listA, List<String> listB) {

    // create an empy Set for the words collection
    Set<String> inboth = new HashSet<String> ();

    // go through the first list
    for (String s : listA)
        if ( listB.contains(s) ) inboth.add(s);

    // remove them from both lists.
    // careful to remove all instances, not just the 1st!
    for (String s : inboth) {
        while (listA.contains(s)) listA.remove(s);
        while (listB.contains(s)) listB.remove(s);
    }

    // return the words that were removed
    return inboth;




}
```

(e) [4 marks] Write a *Comparator* that compares two *Stacks* of String objects on the basis of how many elements they contain. That is, a *Stack* with fewer elements will be judged as being smaller than one with more elements.

```
public class myStackComparer implements Comparator <Stack<String>> {

    public int compare (Stack <String> stk1, Stack <String> stk2) {
      return stk1.size () - stk2.size ();
    }
}
```

(f) [3 marks] Describe the difference between the two interfaces *Iterable* and *Iterator*.

Iterable means you can call a method iterator() to generate one (an iterator, that is). Iterator means implement the hasNext() and next() methods().

(g) [7 marks] Write a method printRandElement that takes a Set of Strings and prints out a randomly chosen element from the set. For maximum credit, do this without converting it to a *List*. You can assume the *Set* is not empty.
Note: the set may not necessarily be implemented as a *HashSet*, so you cannot assume that simply iterating over the *Set* will automatically give a random ordering.
Hint: Java provides a class Random which has a method nextInt(int *n*). This will return a pseudo-random, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive).

```
public void printRandElement(Set <String> myset) {
    Random rnd = new Random();

     int i = rnd.nextInt(myset.size());
     String result;
     Iterator iter = myset.iterator ();
     for (int j=0; j<i; j++)
          result = iter.next();
     UI.println(result);
     // nb. original question has String as return type:
     // should have been void, as here.
     // This was confusing: award marks for return OR println

}
```

Question 3. Costs, sorts, and recursion [10 marks]

(a) [2 marks] For the SortedArraySet implementation of the Set interface, the *Binary Search* algorithm offers a speed-up from $\mathcal{O}(n)$ to $\mathcal{O}(\log n)$ on *which* of the following methods? (circle those that apply)

- size no
- contains YES
- add no - shifting up costs order n
- remove no

(b) [2 marks] Suppose SelectionSort takes 1 second to sort 1000 (one thousand) items on your new laptop. Approximately how long will the same machine take to sort 8 times as many (8000) items?

The second problem is 8 times larger than the first. SelectionSort scales as $\mathcal{O}(n^2)$, so by that reasoning it should take about 64 seconds. (NB: in fact half this, as it takes $\approx n^2/2$ steps, but we gave full credit to 64 since the "order" reasoning is what we're after).

(c) [2 marks] Suppose QuickSort takes 1 second to sort 1000 items on your old laptop. Approximately how long would the same machine take to sort 8 times as many (8000) items?

QuickSort scales as $\mathcal{O}(n \log n)$, so it should take about $8 \log_2 8 = 8 \times 3 = 24$ seconds.

(d) [2 marks] A sorting algorithm is said to be "stable" if it will never reverse the initial ordering of two items that are equal. Which of the following algorithms are *NOT* stable?

SelectionSort, InsertionSort, MergeSort, QuickSort

SelectionSort and QuickSort *are not* stable. The other two *are* stable.

(e) [2 marks] The factorial of n is the result of multiplying $n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$. The recursive code below calculates a factorial:

```
public int factorial ( int num) {
        if (num > 1) { return num * factorial(num-1); }
        else { return 1; }
}
```

What is the "big-$\mathcal{O}$" cost of calling factorial(n), in terms of n?

$\mathcal{O}(n)$, since the code just goes down through n recursion calls (and back).

****

appendix

Some brief and truncated documention that may be helpful:

```
interface Collection<E>
    public boolean isEmpty()
    public int size()
    public boolean add(E item)
    public boolean contains(Object item)
    public boolean remove(Object element)
    public Iterator<E> iterator()

interface List<E> extends Collection<E>
    // Implementations: ArrayList, LinkedList
    public E get(int index)
    public E set(int index, E element)
    public void add(int index, E element)
    public E remove(int index)
    // plus methods inherited from Collection

interface Set extends Collection<E>
    // Implementations: ArraySet, HashSet, TreeSet
    // methods inherited from Collection

interface Queue<E> extends Collection<E>
    // Implementations: ArrayQueue, LinkedList
    public E peek ()                    // returns null if queue is empty
    public E poll ()                    // returns null if queue is empty
    public boolean offer (E element)    // returns false if fails to add

class Stack<E> implements Collection<E>
    public E peek ()                    // returns null if stack is empty
    public E pop ()                     // returns null if stack is empty
    public E push (E element)           // returns element being pushed

interface Map<K, V>
    // Implementations: HashMap, TreeMap, ArrayMap
    public V get(K key)                 // returns null if no such key
    public V put(K key, V value)        // returns old value, or null
    public V remove(K key)              // returns old value, or null
    public boolean containsKey(K key)
    public Set<K> keySet()

public class Collections
    public void sort(List<E>)
    public void sort(List<E>, Comparator<E>)
    public void shuffle(List<E>, Comparator<E>)
```