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

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

Model Solutions
COMP 103: Mid-term Test

19th of August, 2016

Instructions

- Time allowed: **45 minutes**
- There are ~~45~~ marks in total. 40
- 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 contributes 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 it clear where your answers are.

(2 should've been 10, not 5)

Questions	Marks	
1. Interfaces	[5]	
2. Stacks	[5]	
3. Linked Structures	[10]	
4. Costs	[5]	
5. Trees	[15]	
TOTAL:	~~45~~ 40	

Question 1. Interfaces [5 marks]

Write a class called `StudentRecordsImplementation` that implements the following `StudentRecords` interface. It should have a private field that stores a list of students that is initialised with an array list of students.

```
import java.util.List;
import java.util.ArrayList;

class Student { }

interface StudentRecords {
    public void addStudent(Student s);
    public int numberOfStudents();
    public void printStudents();
}
```

UI ≡ System.out

```
public class StudentRecordsImplementation implements StudentRecords {
        private List<Student> list = new ArrayList<Student>();
        public void addStudent(Student s) {
                this.list.add(s);
        }
        public int numberOfStudents() {
                return this.list.size();
        }
        public void printStudents() {
                for (Student s : this.list) {
                        System.out.println(s);
                }
        }
}
```

2

1

1

~~2~~ 1

(UI.println(list);)

✓

Question 2. Stacks [5 marks]

Consider the following `MyCollection` class that implements `add` and `remove` methods using a *stack*:

```
class MyCollection<E> {
        Stack<E> s = new Stack<E>();
        public void add(E e) { s.push(e); }
        public E remove() { return s.pop(); }
        // The following method prints out the collection on one line
        // using comma to separate the values:
        public void printAll() { System.out.println(s); }
}
```

(a) What would the following method print out?

```
public void doIt() {
     MyCollection<String> stuff = new MyCollection<String>();
     stuff.add("A");    stuff.add("B");    stuff.remove();
     stuff.add("C");    stuff.add("D");    stuff.remove();
     stuff.printAll();
     stuff.add("E");    stuff.add("F");    stuff.remove();
     stuff.printAll();
}
```

(half mark per each correct one
=> 2.5)

[A, C]
[A, C, E]

(b) Now assume that we replace `MyCollection` with the following implementation that uses a *queue*:

```
import java.util.ArrayDeque;
class MyCollection<E> {
     ArrayDeque<E> q = new ArrayDeque<E>();
     public void add(E e) { q.offer(e); }
     public E remove() { return q.poll(); }
     // The following method prints out the collection on one line
     // using comma to separate the values:
     public void printAll() { System.out.println(q); }
}
```

What would the same code as in part (a) print out now?

[C, D] (same as above => 2.5)
[D, E, F]

Question 3. Linked Structures [10 marks]

empty 1

single 1

task 3

NO NEED TO USE CONSTRUCTORS

Doubly Linked List is a data structure based on a Linked List where each node has a pointer to both *next* and *previous* node. Here is an example of such DLLNode class:

```
public class DLLNode<E> {
  public E value;
  public DLLNode<E> next;
  public DLLNode<E> prev;
}
```

Implement *add* and *remove* methods for a DoublyLinkedList class:

5

```
public class DoublyLinkedList<E> {
  private DLLNode root;

  // returns true if the value is added successfully,
  // returns false if the value is already present
  public boolean add(E value) {
```

-2 marks

(check for null optional for value)

```
     ~~if (root.value).equals(value))~~
              ~~return false;~~

     ~~[illegible]~~
     if (contains(value)) return false;
     DLLNode n = new DLLNode<E>();
     n.value = value;
     n.next = root;
     n.prev = root.prev;
     root.prev.next = n;
     root.prev = n;

  }
```

3

2

[illegible]

```
public boolean contains(E value) {
   if (root == null) return false;
   DLLNode n = root.next;
   ~~while (n if~~
   if (root.value.equals(value)) return true;
   while (n != root) {   // assume nonulls in DLL;
      if (n.value.equals(value)) return true;
      n = n.next;
   }
}
```

(Question 3 continued on next page)

(Question 3 continued)

5

```
// returns true if the value is removed successfully,
// returns false if the value is not found
public boolean remove(E value) {
    if (!contains(value)) return false;
    DLL.node n = root;
    while (!n.value.equals(value)) {
        n = n.next;
    }
    ~~n.prev = n.next~~
    if ( n == root && root.next == root.prev
        root= null; return true;
    } // otherwise more than 1 node
    n.prev = n.next;
    n.next = n.prev;

}
```

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. Costs [5 marks]

linear

(a) Consider a search implemented on an unsorted array. Assume that it takes 10 seconds to sort 1,000 items. How many seconds will it take to sort 1,000,000 items?

$\boxed{2}$

10,000 seconds

(b) Consider a binary search implemented on a sorted array. Assume that it takes 10 seconds to sort 1,000 items. How many seconds will it take to sort 1,000,000 items?

$\boxed{3}$

log 1000 = 10 sec

log 1000000 = ? sec

20 seconds

log 1000000 = log1000 + log100

= 10 + 10

log a.b = log a + log b

1- realising its about log

2- doing wrong but close

3- correct even with loss in

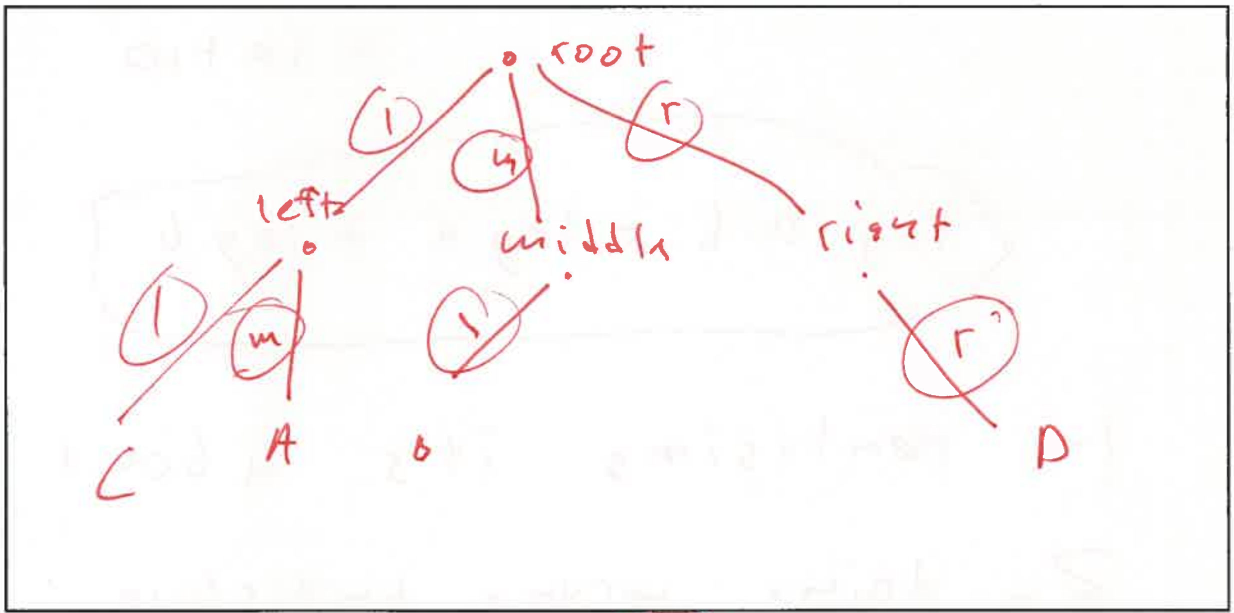
Question 5. Trees [15 marks]

Consider the following triple tree data structure:

```
class TripleTreeNode {
        public TripleTreeNode left, middle, right;
        public String value;
        public TripleTreeNode(String value) {
                this.value = value;
        }
}
```

(a) [5 marks] Draw a resulting tree that the following code will create:

```
TripleTreeNode root = new TripleTreeNode("root");
root.left = new TripleTreeNode("left");
root.middle = new TripleTreeNode("middle");
root.right = new TripleTreeNode("right");
root.left.middle = new TripleTreeNode("A");
root.middle.left = new TripleTreeNode("B");
root.left.left = new TripleTreeNode("C");
root.right.right = new TripleTreeNode("D");
```



5

~~(1 for direction)~~

2 marks for edge labels or dir's

3 marks for correct tree

(Question 5 continued on next page)

(Question 5 continued)

(b) [5 marks] Implement *recursively* the `size()` method inside `TripleTreeNode` class that returns the total number of nodes in a tree with the current node as its root:

```
p.int size () {
1       ~~pa~~ ~~if~~   int l = (left ==null) 0: left.size();
1                  int r = (right==null) 0: right.size();
1                  int m = (middle ==null) 0: middle.size();
2                  return l+r+m+1;
    }
```

(c) [5 marks] Implement *iteratively* the `size` method that takes a parameter of type `TripleTreeNode` that returns the total number of nodes in a tree with the given node as its root:

```
p. int size(TTN node) {
  Stack<TTN> s = new Stack<TTN>();
    ~~if (n.left != null) s.push(n~~
2 if(n != null)
    s.push (node.
    int answer = 0;
  while (!s.empty()) {
        TTN n = s.pop();
3       if (n.left != null)   s.push (n.left);
        if (n.middle!=null)  s. push(n.middle);
        if (n.right !=null)  s. push(n.right);
        answer ++;
  }
  return ans;
}
```

****

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.

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: ArrayDeque, 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>)
```