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

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

COMP 103: Test 2

23 May, 2013

Instructions

- Time allowed: **45 minutes**
- There are 45 marks in total.
- Answer **all** the questions.
- Write your answers in the boxes in this test paper and hand in all sheets. Ask for additional paper if you need it.
- If you think some question is unclear, ask for clarification.
- Brief Java documentation is supplied on the last page.
- This test will be converted to 15% 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, and calculators without a full set of alphabet keys.
- You may write notes and working on this paper, but make sure it is clear where your answers are.

Questions	Marks	
1. Linked Lists	[13]	
2. Data structures using lists	[12]	
3. Trees	[20]	
	TOTAL:	

Question 1. Linked Lists [13 marks]

(a) [2 marks] If you have a large number of items, which data stucture will be faster for finding items: a sorted array, or a linked list? Briefly explain why.

Justification:

(b) [4 marks] The lectures described a LinkedNode class that represents a single node in a linked list. It contains the value, and a pointer to the next node in the list.
The lectures also described the benefits of using a higher level class that might be called LinkedList. This class includes a private LinkedNode field and possibly other information.

Indicate two benefits of using the LinkedList class:

(Question 1 continued on next page)

(Question 1 continued)

(c) [5 marks] Write the code for a small class called Person that includes name and age fields, the associated public getName() and getAge() methods, and a constructor that takes name and age arguments.

(d) [2 marks] Assume that you are given a standard LinkedList class. Here is a partial definition of the class:

```
public class LinkedList<E> {
    public LinkedList()      // constructor
    public void insert( int  index,  E element)
    public E remove(int index)
}
```

Using this linked list class and the Person class, write the declaration for a linked list of Persons.

Question 2. Data structures using lists [12 marks]

(a) [2 marks] In implementing a Stack using an existing linked list class, is it better to implement push(E object) by inserting the object at the beginning of the list, or adding it to the end of the list? Explain your answer.

Justification:

(b) [2 marks] Recall that a Queue has methods Offer(E object) that tries to add object to the queue, and E poll(), that removes and returns the object at the beginning of the queue. In implementing a Queue using a linked list, is it possible to get O(1) performance for *both* the offer() and poll() methods? If not, indicate why this is not possible. If so, indicate how this is possible.

Justification:

(c) [6 marks] Suppose you know in advance what the maximum size of the queue will be. Instead of using a Linked List, is it possible to implement the queue using a simple array and get O(1) performance for *both* poll() and offer()? If yes, briefly describe how this could be done. If no, argue why it is not possible. The answer should be several sentences, describing roughly how poll() and offer() map to array operations, and what happens when you hit the end of the array.

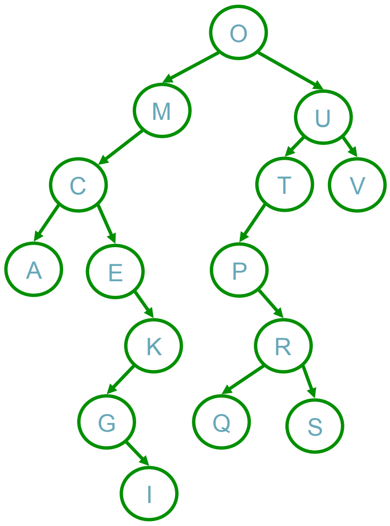
(d) [2 marks] Consider these statements:

```
LinkedQueue<String> Q = new LinkedQueue<String>();
Q.offer("Jenny");
Q.offer("Hamish");
String person1 = Q.poll();
Q.offer("Tanya");
```

After this portion of the program runs, what is the value of the variable person1, and what is the name is at the head of the Queue?

Question 3. Trees

[20 marks]



Consider the binary tree shown here.

(a) [1 mark] How many leaves are in this tree?

(b) [2 marks] List the nodes in the branch that starts at the root and ends at the leftmost leaf.

(c) [1 mark] Is this tree balanced? Explain.

Justification

(d) [2 marks] This question uses a standard Binary Tree class that includes the following constructor:

```
public class BinaryTreeNode<E> {
  public BinaryTreeNode(E item,
                        BinaryTreeNode<E> leftChild,
                        BinaryTreeNode<E> rightChild)
  ...
}
```

Draw the tree that is created by the following statements:

```
BinaryTreeNode<String> A = new BinaryTreeNode<String>("A", null, null);
BinaryTreeNode<String> X = new BinaryTreeNode<String>("X", null, null);
BinaryTreeNode<String> M = new BinaryTreeNode<String>("M", A, null);
BinaryTreeNode<String> B = new BinaryTreeNode<String>("B", null, M);
BinaryTreeNode<String> Y = new BinaryTreeNode<String>("Y", X, B);
```

draw your tree here:

(e) [2 marks] Consider the following function to print the node values in a tree.

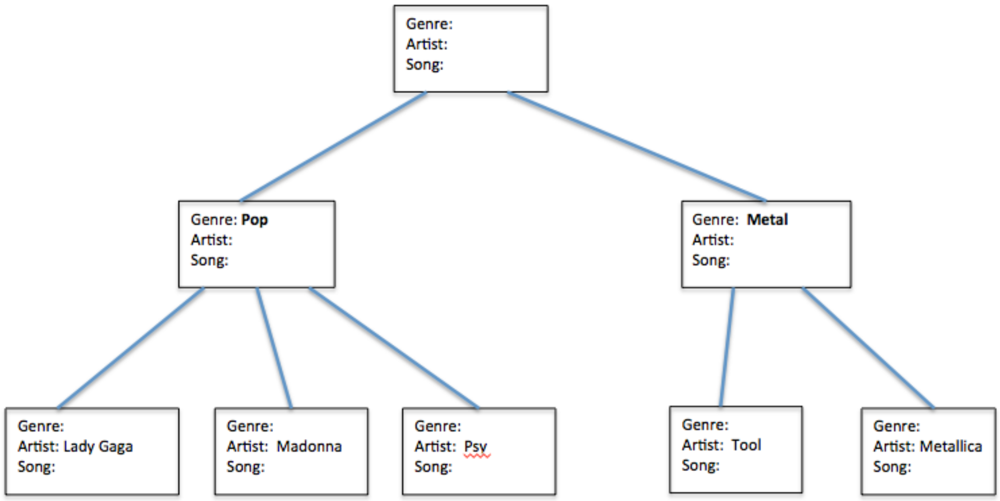
```
void treeprint (BinaryTreeNode node) {
  if (node.leftChild != null)  treeprint (node.leftChild );
  System.out.println(node.value);
  if (node.rightChild != null)  treeprint (node.rightChild );
}
```

What will this function print if called on the root of the tree constructed in question (d).

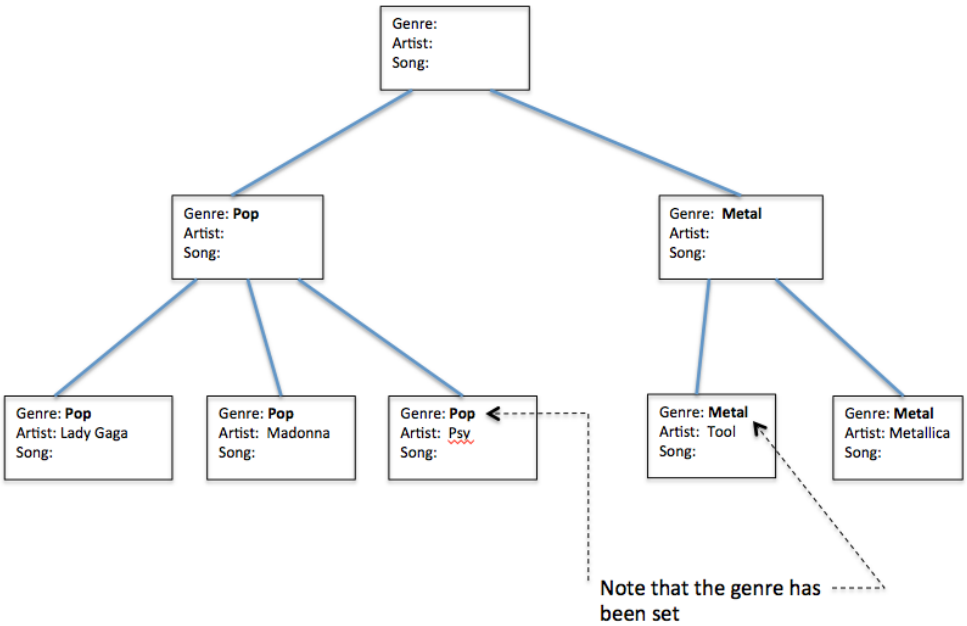
(f) [2 marks] Suppose that you are given a tree, and want to verify that the nodes of the tree are in order with respect to some traversal, such as depth-first preorder traversal. What will be the complexity of an efficient algorithm for verifying that the nodes are in order when visiting with a depth-first preorder traversal?

(g) [10 marks] This question is about a program called SongsOrganiser, that manages a collection of music using a tree structure. The root of the tree is the whole music collection. Below that are nodes that represent music genres. (A *genre* describes the style of music, for example, rap music, rock, electronic, etc.) Below the genres are nodes that represent artists, albums, and songs.

In this question, you will write a function that finds nodes that have the genre defined, and propagate the genre down the tree to the artists, which need their genre defined. In other words, your function will modify a tree like this one:



to be like this:



(Question 3 continued on next page)

(Question 3 continued)

The information at each node is represented by a class called MusicInfo. Here is a portion of that class:

```
public class MusicInfo {
  public MusicInfo() {}
  public String getGenre() { return genre; }
  public void setGenre(String genre) { this.genre = genre; }

  private String genre;
  private String  artist ;
  private String song;
}
```

(Note that using a single node type for all this information is not the best sofware design, but it keeps things simple.)

The tree itself is represented with nodes of the generic TreeNode class:

```
class TreeNode<E> {
  public TreeNode(E value) { this.value = value; }
  public E getValue() { return value; }
  public ArrayList<TreeNode<E>> getChildren() { return children; }

  private E value;
  private ArrayList<TreeNode<E>> children;
}
```

Write the requested function on the next page.

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: 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>)
```