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

ID Number: . . . . . .

COMP103 Test #1

26 March, 2009

| ********** WITH SOLUTIONS **********

Instructions

- Time: **45 minutes**.
- Answer **all** the questions.
- There are 45 marks in total.
- Write your answers in the boxes in this test paper and hand in all sheets.
- Every box with a heavy outline requires an answer.
- If you do not understand a question, ask for clarification.
- There is some java documentation at the end of the exam paper.

		Marks		
1.	Collection Types	[11]	1	
2.	Using Collections	[10]	2	
3.	Iterators, Comparators etc.	[15]	3	
4.	Cost of Algorithms	[9]	4	
			Total:	

Student ID: .....

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. Collection Types [11 marks]

(a) [2 marks] Which collection type gives you access to a sequence of items in the same order as you put them in?

A queue.

(b) [2 marks] Which collection type would be most suitable to store recent URLs (websites), for use by the browser program when the user presses the "back" button? Explain why.

A stack: this is "undo".

(c) [2 marks] Which collection type would be most suitable for storing information about the people in a building and which room each person is in?

Map. Use a map with the people (who form a set) as the keys and room they are in as the value. Alternatively use a map with rooms as keys and each value being a set of the people in that room.

(d) [2 marks] Which types of collections can access any element directly?

Bag, Set, List, Map

(e) [3 marks] All the java collections are Iterable, even Set, which is not an ordered collection. Why is this?

Implementing Iterable means we provide an iterator, and we do frequently need to iterate over the elements in a set (e.g. in a foreach loop), even though they have no natural ordering.

Question 2. Using Collections [10 marks]

A group of people have then been asked to name their favourite colours in a questionaire. Instead of giving the Person class a new field, the results of the questionaire have been stored in a Map from people to their favourite colours (stored as Strings). Our program, however, needs to find the groups of all people who share the same favourite colour.

Complete the following method that takes the original Map as an argument and returns a different one. This new Map has the colours as keys. Each value is the Set of people who had that colour as their favourite. You will need to

- make a new Map,
- fill it by going through the original one, and
- return the result.

```
public Map <String,Set<Person>> convertInfo (Map<Person,String> favColours) {

    // make a new Map
    Map<String,Set<Person>> newmap = new HashMap<String, Set<Person>>();

    // go through the old one, filling new map up
    for (Person pers: favColours.keySet() ) {
        String colour = favColours.get(pers);
        // if first use of colour, make a new map entry with a new set.
        if (!newmap.keySet().contains(colour))
            newmap.put(colour, new HashSet<Person>());
        newmap.get(colour).add(pers);
    }

    return(newmap);









    return .....
}
```

Question 3. Iterators, Comparators and Exceptions [15 marks]

(a) [3 marks] Consider a List of Task objects, which have a field urgency. Provided the Task class includes a suitable method compareTo, the list can be sorted in terms of increasing urgency. Complete the code for this method.

```
    public int compareTo(Task other) {
      return (this.urgency − other.urgency );

or more verbosely like this:
       if (this.urgency > other.urgency ) return(1);
      else if (this.urgency < other.urgency ) return(−1);
      return(0);
    }
```

(b) [3 marks] Explain the difference between using

obj1 == obj2

and

obj1.isEqual(obj2)

to compare two objects.

"==" compares references (unless operating on primitive types). So if 2 objects are == they're the same, single, object.
".isEqual()" compares the objects themselves. So if A.isEqual(B) they're the same according to whatever code is in the "isEqual()" method.

Consider the following code for a random number generator:

```
public class RandNumIterator implements Iterator <Integer> {
    private int num = 1;
    public boolean hasNext() {
        return true;
    }
    public Integer next() {
        num = (num * 92863) % 104729 + 1;
        return num;
    }
    public void remove(){ throw new UnsupportedOperationException(); }
}
```

(c) [2 marks] Why is the remove method included here, when all it does is throw an exception?

> It is required for the interface Iterator

(d) [2 marks] UnsupportedOperationException is one of a larger class of exceptions that don't need to be caught: what is the name of that class?

> RuntimeException

(e) [5 marks] Complete the following method, that first creates a RandNumIterator (an instance of the class given above) and then uses it to print out N random numbers.

```
private void printNNumbers(int N) {
        Iterator <Integer> lottery = new RandNumIterator();
        for (int i=1; i<N; i++)
            System.out.println( lottery .next ());




}
```

Student ID: .....

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 [9 marks]

In lectures we looked at an ensureCapacity method for ArrayList, which is called from within add and increases the size of the array if the collection increases in size beyond the current array. We might also want to have a reduceCapacity method, which is called from within remove and releases memory when a collection *decreases* in size sufficiently.

The code below gives an implementation for these two methods, based on doubling, and halving, the size of the array.

(Note that the constructor and other methods are omitted or truncated here.)

```
public class ArrayList <E> extends AbstractList <E> {
  private E[] data;
  private int count=0;
  private static final int INITIALCAPACITY = 1;

  // methods
  public ArrayList () {...}
  public void add (int index, E item) {...}
  public E remove (int index) {...}

  private void ensureCapacity() {
      if (count < data.length) return; // not full yet
      E[] tmpArray = (E[]) (new Object[data.length * 2]);
      for ( int i=0; i<count; i++)
          tmpArray[i] = data[i];
      data = tmpArray;
  }

  private void reduceCapacity() {
      if (data.length <= INITIALCAPACITY) return;
      int threshold = data.length / 2;
      if (count >= threshold) return;  // not empty enough yet
      E[] tmpArray = (E[]) (new Object[threshold]);
      for ( int i=0; i<count; i++)
          tmpArray[i] = data[i];
      data = tmpArray;
  }
}
```

(a) [4 marks] What is the "amortised" (per item) cost of removing all n elements one at a time from an array that starts off completely full?

n items to start with, and we reduceCapacity if it is half full.

The first n/2 items : 1 each, subtotal=n/2
n/2 item itself: copy n/2 items, subtotal=n/2, total=n
The next n/4 items : 1 each, subtotal=n/4
n/4 item itself: copy n/4 items, subtotal=n/4, total= n + n/2
The next n/8 items : 1 each, subtotal=n/8
n/8 item itself: copy n/8 items, subtotal=n/8, total= n + n/2 +n/4
...
The total will be n + n/2 + n/4 + n/8 + n/16 + ...
This comes to 2n.
Amortised cost is this PER ITEM, so divide by n.
Overall then this is 2 per item, which in 'big O' terms is O(1).

(b) [5 marks] (*Hard*) Can you identify a worst-case scenario for this algorithm using the current threshold, and suggest a better alternative?

A worst-case scenario would be where the array is just on half full, and keeps getting adds and removes close to this threshold. Every time a remove() makes it goes below half-full, the array is halved so it ends up almost full. A subsequent add() makes it double again, and so on... In the worst case it might have to copy the array (via ensureCapacity, or reduceCapacity) almost every time.

A way to handle this would be to wait until the array is a *quarter* full before reducing its size. That way, both ensureCapacity() and reduceCapacity() leave an array that is half full.

****

Student ID: .....

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 (may be removed)

Brief (and simplified) specifications of some relevant interfaces and classes.

```
public interface Iterator<E>
    public boolean hasNext();
    public E next();
    public void remove();

public interface Iterable<E>                    // Can use in the "for each" loop
    public Iterator<E> iterator();

public interface Comparable<E>                  // Can compare this to another E
    public int compareTo(E o);

public interface Comparator<E>                  // Can use this to compare two E's
    public int compare(E o1, E o2);
```

```
public interface Collection<E>
    public boolean isEmpty();
    public int size ();
    public boolean add();
    public Iterator <E> iterator();

public interface List<E> extends Collection<E>
    // Implementations: ArrayList
    public E get( int index);
    public void set( int index, E element);
    public void add(E element);
    public void add(int index, E element);
    public void remove(int index);
    public void remove(Object element);

public interface Set extends Collection<E>
    // Implementations: ArraySet, SortedArraySet, HashSet
    public boolean contains(Object element);
    public boolean add(E element);
    public boolean remove(Object element);

public 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);

public 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);

public interface Map<K, V>
    // Implementations: HashMap, TreeMap, ArrayMap
    public V get(K key);                 // returns null if no such key
    public void put(K key, V value);
    public void remove(K key);
    public Set<Map.Entry<K, V>> entrySet();
```