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

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

Model Solutions
COMP 103: Test 1

10 Aug, 2012

Instructions

- Time allowed: **25 minutes**
- There are 25 marks in total.
- Answer **all** the questions.
- Write your answers in the boxes in this test paper and hand in all sheets. You may ask for additional paper if you need it.
- If you think some question is unclear, ask for clarification.
- Brief Java documentation will be supplied with the test.
- This test will contribute 10% 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. Collections	[10]	
2. Programming with Collections	[10]	
3. Iterators	[5]	
	TOTAL:	

Question 1. Collections [10 marks]

For each of these programming tasks, state what kind of collection you would use, and justify why you chose that kind of collection.

(a) [2 marks] A collection recording the visitors to a building site. The program only needs check visitors in or out, and to be able to list all the visitors who are currently on site.

Collection Type: Set
Justification: We do not need to keep the visitors in any order. A Set allows adding and removing. We can't have the same visitor twice at the same time, so no duplicates are allowed.

(b) [2 marks] A collection recording the cards in a deck of cards. The program needs to shuffle the cards, and to deal the next card from the deck.

Collection Type: List
Justification: The cards must be kept in an order. A Stack would be good for dealing the next card, but shuffling is not feasible for a Stack since we need to change the order of the cards. Hence a List.

(c) [2 marks] A collection recording the patients waiting at a medical center that operates first-come, first served (no appointments allowed). The program needs to identify the next patient when a doctor becomes free.

Collection Type: Queue of patients.
Justification: We need to keep the patients in order of arrival, and we only remove the person at the head of the queue.

(Question 1 continued on next page)

(Question 1 continued)

(d) [2 marks] A collection recording the clients of an accounting partnership. Each client "belongs to" one of the partners. The program needs to let the receptionist identify which partner a client needs to see.

Collection Type: Map from clients to partners
Justification: each client is associated with the partner, so a map is appropriate. We need to look up the partner given the client, so the key is the client, and the value is the partner.

(e) [2 marks] Explain the difference(s) between List and ArrayList in the Java collections framework, and state the relation between them.

List is an interface, so it defines a type, but you cannot make an object of it. ArrayList is a class, and therefore you can create an object. ArrayList implements the List interface.

Question 2. Programming with Collections [10 marks]

This question concerns a program that manages a queue of Application objects. The queue is stored in the appQueue field; the program also stores a set of all the names of applicants who have recieved a grant in the grantees field. The declarations are shown below, along with part of the Application class.

```
public class ApplicationManager{

    private Queue<Application> appQueue = new ArrayQueue <Application>();
    private Set<String> grantees = new HashSet <String>; // people who already have grants

     ...

}
```

```
public class Application {

    private String name;      // name of the applicant
    private String category; // category of grant
    private boolean approved;

    /** Return the name of the applicant */
    public String getName(){
        return this.name;
    }
    /** Return the category of the grant application */
    public String getCategory(){
        return this.category;
    }
    /** return a string with all the details of the grant application */
    public String getDetails(){
         ...
    }
    /** Record that the application has been approved */
    public void setApproved(){
        approved = true;
    }

     ...
}
```

(Question 2 continued on next page)

(Question 2 continued)

(a) [5 marks] Complete the following processApplication method in the ApplicationManager class which should process the Application at the head of the queue, if there is one. Processing involves first checking that the applicant does not already have a grant. If they have a grant, it reports that the grant is denied, otherwise, it displays the details of the grant and asks the user whether the grant should be approved. If it should be approved, it records that the applicant now has received a grant, it approves the grant, and returns it.

```
public Application processApplication(){

    if (appQueue.isEmpty() // or appQueue.peek()==null  or   appQueue.size()==0 ){
        ..........................................................
        UI.println("No applications to process");
        return null;
    }

    Application app = appQueue.poll();
                      ..........................................................
    String name = app.getName();

    if (grantees.contains(name)                                  ){
        ..........................................................
        UI.println("Grant denied: " + name + " already has grant.");
        return null;
    }

    UI.println(app.getDetails());
    if (UI.askBoolean("Should grant be approved?")){
        grantees.add(name);

        ..........................................................
        app.setApproved();
        UI.println("Name recorded, and application approved.");
        return app;
    }
    return null;
}
```

(Question 2 continued on 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.

(Question 2 continued)

(b) [5 marks] Each category of grant has a different collection of rules for approving that kind of grant. The program stores the rules in a field called allRules which contains a Map that associates a list of rules with each grant category. Each individual rule is a separate String.

```
private Map < String, List<String> > allRules;  // grant category -> list of rules
```

Complete the following printRules method which prints out the list of rules associated with the category of an application, in order to help the user make their decision.

```
public void printRules(Application app){
    String category = app.getCategory();
    UI.println("Rules for " + category+ ":");
    List<String> rules = allRules.get(category);
    for (String rule : rules){
        UI.println(rule);
    }
//or
    for (String rule : allRules.get(app.getCategory())){
        UI.println(rule);
    }

//or  (not so nice - uses built in printing of lists)
    UI.println(allRules.get(app.getCategory()));

// but not good to iterate down allRules.getKeys() or allRules.getEntrySet()


}
```

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. [5 marks]

The Primes class below contains several methods for dealing with primes, including an iterator method that returns an iterator that will generate a sequence of all prime numbers. The PrimeIterator class is a private inner class inside the Primes class. You are to complete the methods in the PrimeIterator class.

```
public class Primes implements Iterable<Integer>{

    /** Return true if the given number is a prime */
    public boolean isPrime(int n ){....}

    /** Return a list of the prime factors of a number */
    public List<Integer> primeFactors(int n ){....}

    /** Return an iterator that generates all the prime numbers */
    public Iterator<Integer> iterator(){
        return new PrimeIterator();
    }

    private class PrimeIterator implements Iterator<Integer> {
        private int state = 0;  // The last value returned;
        public boolean hasNext(){
            return true;
        }
        public Integer next(){
            state++;
            while ( !isPrime(state) ) {state++;}
            return state;
        }
        //or (better, because handles the limit on integer values)
        private int state = 2;   // the next prime to return
        public boolean hasNext(){
            return state> 0;
        }
        public Integer next(){
            if (state<0) throw new NoSuchElementException();
            int ans = state; // remember current state to return
            state++;
            while (state>0 && !isPrime(state)) {state++;} // move state up to next prime
            return ans;
        }

        public void remove() {
            throw new UnsupportedOperationException();
        }
    }

}
```

****

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

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