

Family Name:

Other Names:

ID Number:

Signature

COMP 103: Test 1

9th August, 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.
- Brief Java documentation is supplied on the last pages.
- This test will be converted to 15% of your final grade.

Questions

Marks

1. Question 1

[13]

2. Question 2

[8]

3. Question 3

[24]

TOTAL:

Question 1.

[13 marks]

(a) [2 marks] Suppose you use a *List* called `mypets` to store information about your pets, represented as objects of type `Pet`. Write code to declare and initialise an empty instance of such a list.

.

(b) [2 marks] Many people today have more than one phone number at which they might be reached. Suppose you wish to use a *Map* called `phoneNums` to store the set of phone numbers associated with each of your friends, who are represented by a class `Person`. The *Map* needs to be from keys that are objects of class `Person`, to sets of integers. Write code to declare and initialise an empty instance of such a map.

.

(c) [2 marks] Java has interfaces *Iterable* and *Iterator*. In words, describe the difference between these two interfaces.

.

(d) [3 marks] Java uses the method `Collections.sort()` to sort *Lists* into what is called a “natural ordering”. What is the critical method that the class of objects in the list must have for this to work, and how is this ensured in Java?

.

(e) [2 marks] Which Java interface ensures that a class is able to compare two objects of some other class by returning an integer?

.

(f) [2 marks] *Queues* and *Stacks* can be thought of as *Lists* that have additional constraints placed on them. What are these extra constraints?

Queue constraint:

Stack constraint:

Question 2.

[8 marks]

(a) [2 marks] Here is a list:

```
List <String> mylist = new ArrayList <String> ();
```

Suppose the list has been populated with various Strings by scanning a file, for example.

Write code that uses `UI.println()` to print out the strings, by going through the list using a standard “for” loop such as `for (int i=0; ...)...`

(b) [2 marks] Write code that uses `UI.println()` to print out the strings, by going through the list using a “for each” loop instead.

(c) [4 marks] Write code that uses `UI.println()` to print out the strings, by getting and using an *Iterator* instead.

Question 3.

[24 marks]

This question concerns a program written to keep track of cars in a car sales yard.

Most cars are painted with just one colour, but some consist of several colours. Suppose there is a class `Car`, which has two fields: an integer registration number and a *List* of the all colours painted on that car.

```
public class Car {
    private int reg;
    private Set <String> colours;

    // constructor , which is passed a set of colours
    public Car(int registration , Set <String> cols) {
        this.reg = registration ;
        this.colours = cols;
    }

    // a method
    public Set<String> getColours() { return colours; }
}
```

Note that the constructor has two arguments: an integer and *Set* of *String* objects (the colours).

Each car is identified by its unique registration number, and stores its colours in a *Set*.

Information on an individual car is stored on a single line, in a simple text file. The format is the car model (eg "Mini"), registration number, and a list of colours, for each car. A car will always have at least one colour.

Here is a short example:

```
Cortina 1634    blue    yellow    green
Mini     7721    red
Mini     223    black    white
Jeep     989    white
```

Suppose you are writing a class which reads a text file formatted in this way. You may assume that the file is correctly formatted.

(a) [10 marks] Complete the following method `mapModelToCars`, which is passed a filename. The method needs to generate and return a *Map*. Each entry in the Map will have a model as its key, and a *Set* of *Car* objects as its value.

The first couple of lines are provided for you.

It is usually a good idea to start with pseudocode, as comments.

```
public . . . . . mapModelToCars(String filename) {

    Map <String, Set<Car> > modelsMap = new HashMap <String, Set<Car>> ();

    try {
        Scanner sc = new Scanner(new File(filename));

        }

    catch (IOException e) {
        UI.println ("Error: File not found!");
        return null;
    }

}
```

(b) [7 marks] Write code for `printAllColours`, which uses the `Map` to generate and print out the set of all the colours that appear on at least one car.

Note this is the same as all the colours mentioned in the original file, but they are now stored inside `Car` objects, which are in the *Map*.

Each such colour should be printed out just once, but the order does not matter.

```
public void printAllColours (Map <String, Set<Car>> modelsMap) {
```

```
}
```


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 documentation 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 Collection<V> values()
    public Set<Map.Entry<K,V>> entrySet()

interface Map.Entry<K,V> // a nested class of Map
    K getKey()
    V getValue()

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

```
interface Iterable :  
    public Iterator <T> iterator ()
```

```
interface Iterator :  
    public boolean hasNext()  
    public E next()  
    public void remove(E)
```

```
interface Comparable:  
    public int compareTo(E)
```

```
interface Comparator:  
    public int compare(E ob1, E ob2)
```

```
class UI:  
    public println (anything val)
```

```
class Scanner:  
    public boolean hasNext()  
    public boolean hasNextInt()  
    public String next()  
    public int nextInt ()  
    public String nextLine()
```