BIRKBECK

(University of London)

MSc EXAMINATION FOR INTERNAL STUDENTS

MSc Computer Science

MSc Data Science

Department of Computer Science and Information Systems

Principles of Programming II

PRACTICE PRACTICAL QUESTIONS

BUCI033S7

RUBRIC:

1. Candidates should attempt ALL 12 questions on this paper.
2. You are advised to look through the entire examination paper before getting started, in order to plan your strategy.
3. Simplicity and clarity of expression in your answers is important.
4. All programming questions should be answered using the JAVA programming language.
5. Electronic calculators are **NOT** allowed.

Question:	1	2	3	4	5	6	7	8	9	10	11	12	Total
Marks:	22	35	16	25	5	6	3	6	6	25	25	35	209

Question 1 ..... Total: 22 marks

A university has a number of departments, each of which is responsible for one or more programmes of study. For example, the following code fragment shows that the *Computing* department has two programmes, *Computing* and *Information Systems*, which have 600 and 300 enrolled students, respectively. (Each department is represented as a map, and the University as a map of maps.)

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

public class Univ {
    public static void main(String[] args) {
        new Univ().run();
    }

    class Course {
        private String name;
        private int id;

        Course(String name, int id) {
            this.name = name;
            this.id = id;
        }
    }

    public void run() {
        List cs = new ArrayList();
        cs.add("Computer Science");

        cs.add(new Course("Computing", 600));
        cs.add(new Course("Information Systems", 300));

        List eng = new ArrayList();
        eng.add("Engineering");

        eng.add(new Course("Mechanical", 100));
        eng.add(new Course("Civil", 150));

        List mng = new ArrayList();
        mng.add("Business");
        mng.add(new Course("Management", 800));

        List courses = new ArrayList();
        courses.add(cs);
        courses.add(eng);
        courses.add(mng);

        // ...
    }
}
```

Write code to do the following:

(a) Rewrite the above using a Java `Map` data structure. 6 marks

(b) Print out how many university departments there are. 2 marks

(c) Print how many programmes are delivered by the Computing department. 4 marks

(d) Print how many students are enrolled in the Civil Engineering programme. [4 marks]

(e) Add a *Data Management* programme, with 100 students, to the *Computing* department. [6 marks]

Question 2 ..... Total: 35 marks

In mathematics, several operations are defined on sets.

- The *union* of two sets A and B is a set that contains all the elements that are in A together with all the elements that are in B.
- The *intersection* of A and B is the set that contains elements that are in both A and B.
- The *difference* of A and B is the set that contains all the elements of A except for those elements that are also in B.

Suppose that A and B are variables of type `Set` in Java. The mathematical operations on A and B can be computed using methods from the `Set` interface. In particular:

`A.addAll(B)` computes the union of A and B;

`A.retainAll(B)` computes the intersection of A and B; and

`A.removeAll(B)` computes the difference of A and B.

(These operations change the contents of the set A, while the mathematical operations create a new set without changing A, but that difference is not relevant to this problem.)

For this question, you should write a program that can be used as a "set calculator" for simple operations on sets of non-negative integers. (Negative integers are not allowed.) A set of such integers will be represented as a list of integers, separated by commas and, optionally, spaces and enclosed in square brackets. For example:

```
[1,2,3]
```

or

```
[17, 42, 9, 53, 108]
```

The characters `+`, `*`, and `-` will be used for the union, intersection, and difference operations. The user of the program will type in lines of input containing two sets, separated by an operator. The program should perform the operation and print the resulting set. Here are some examples:

```
[1, 2, 3] + [3,  5,  7]
```

results in

```
[1, 2, 3, 5, 7]
```

and

```
[10,9,8,7] * [2,4,6,8]
```

results in

```
[8]
```

and finally,

```
[ 5, 10, 15, 20 ] - [ 0, 10, 20 ]
```

results in

```
[5, 15]
```

To represent sets of non-negative integers, use sets of type `TreeSet<Integer>`. Read the user's input, create two `TreeSet`s, and use the appropriate `TreeSet` method to perform the requested operation on the two sets.

Your program should be able to read and process any number of lines of input. If a line contains a syntax error, your program should not crash. It should report the error and move on to the next line of input.

Note: To print out a `Set`, A, of `Integer`s, you can just say `System.out.println(A)`.

Question 3 ..... Total: 16 marks

The following questions all apply the the Java Collections library and specifically `LinkedList`.

(a) Write a Java method to iterate a linked list in reverse order. 2 marks

(b) Write a Java method to get the first and last occurrence of the specified elements in a linked list. 2 marks

(c) Write a Java method to remove first and last element from a linked list. 2 marks

(d) Write a Java method of swap two elements in an linked list. 2 marks

(e) Write a Java method to shuffle the elements in a linked list. 2 marks

(f) Write a Java method to join two linked lists. 2 marks

(g) Write a Java method to compare two linked lists. 2 marks

(h) Write a Java method to check if a particular element exists in a linked list. 2 marks

Question 4 ..... Total: 25 marks

A *predicate* is a boolean-valued function with one parameter. Some languages use predicates in generic programming.

In Java, we could implement "predicate objects" by defining a generic interface:

```
public interface Predicate<T> {
    public boolean test( T obj );
}
```

The idea is that an object that implements this interface knows how to "test" objects of type `T` in some way. Define a class that contains the following generic static methods for working with predicate objects. The name of the class should be `Predicates`, in analogy with the standard class `Collections` that provides various static methods for working with collections.

```
public static <T> void remove(Collection<T> coll, Predicate<T> pred)
   // Remove every object, obj, from coll for which
   // pred.test(obj) is true.

public static <T> void retain(Collection<T> coll, Predicate<T> pred)
   // Remove every object, obj, from coll for which
   // pred.test(obj) is false.  (That is, retain the
```

```
   // objects for which the predicate is true.)

public static <T> List<T> collect(Collection<T> coll, Predicate<T> pred)
   // Return a List that contains all the objects, obj,
   // from the collection, coll, such that pred.test(obj)
   // is true.

public static <T> int find(ArrayList<T> list, Predicate<T> pred)
   // Return the index of the first item in list
   // for which the predicate is true, if any.
   // If there is no such item, return -1.
```

Question 5 ..... Total: 5 marks

What is the following class converted to after type erasure?

```
public class Pair<K, V> {

    public Pair(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public K getKey() {
        return key;
    }

    public V getValue() {
        return value;
    }

    public void setKey(K key) {
        this.key = key;
    }

    public void setValue(V value) {
        this.value = value;
    }

    private K key;
    private V value;
}
```

Question 6 ..... Total: 6 marks

Consider the following stack implementation:

```
public class Stack {
    class Node { Object value; Node next; };
    private Node top;

    public void push(Object newValue) {
        Node n = new Node();
```

```
            n.value = newValue;
            n.next = top;
            top = n;
        }

        public Object pop() {
            if (top == null) return null;
            Node n = top;
            top = n.next;
            return n.value;
        }
    }
```

Describe two different ways in which the data structure can fail to contain the correct elements.

Question 7 ..... Total: 3 marks

What is wrong with this code snippet?

```
public class Stack {
    private Object myLock = "LOCK";

    public void push(Object newValue) {
        synchronized (myLock) {
            ...
        }
    }
    ...
}
```

Question 8 ..... Total: 6 marks

Generate 1,000 threads, each of which increments a counter 100,000 times. Compare the performance of using `AtomicLong` versus `LongAdder`.

Question 9 ..... Total: 6 marks

Use a `LongAccumulator` to compute the maximum (or minimum) of the accumulated elements.

Question 10 ..... Total: 25 marks

Use a blocking queue for processing files in a directory. One thread walks the file tree and inserts files into a queue. Several threads remove the files and search each one for a given keyword, printing out any matches. When the producer is done, it should put a dummy file into the queue.

Question 11 ..... Total: 25 marks

Repeat the preceding question, but instead have each consumer compile a map of words and their frequencies that are inserted into a second queue. A final thread merges the dictionaries and prints the ten most common words. Why don't you need to use a `ConcurrentHashMap`?

Question 12 ..... Total: 35 marks

Repeat the preceding question, making a `Callable<Map<String, Integer>>` for each

file and using an appropriate *executor service*. Merge the results when all are available. Why don't you need to use a `ConcurrentHashMap`?

