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 WRITTEN QUESTIONS

BUCI033S7

DURATION OF PAPER: One hour

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.
6. START EACH QUESTION ON A NEW PAGE.

Question:	1	2	3	4	5	6	7	8	9	10	11	12	Total
Marks:	9	10	8	7	15	10	15	17	15	16	18	12	152

Question 1 ..... Total: 9 marks

Provide brief definitions of the following terms:

(a) Object-oriented programming — 1 mark

(b) Object — 1 mark

(c) Class — 1 mark

(d) Information hiding — 1 mark

(e) Encapsulation

(f) Inheritance — 1 mark

(g) Polymorphism — 1 mark

(h) Derived class — 1 mark

(i) Interface — 1 mark

(j) Unit test — 1 mark

You should provide appropriate examples in Java to illustrate your answer.

Question 2 ..... Total: 10 marks

Briefly explain each of the following terms:

- dynamic binding,
- cast expression,
- overriding,
- `this`
- `super`

You should provide appropriate examples in Java to illustrate your answer.

Question 3 ..... Total: 8 marks

Consider the following code fragment.

```
ArrayList list = new ArrayList();
list.add("P");
list.add("Q");
list.add("R");
list.set(2,"s");
list.add(2,"T");
list.add("u");
System.out.println(list);
```

What is printed as a result of executing the fragment?

[An extract from the API for the `java.util.ArrayList` class is given at the end of the paper.]

Question 4 ..... Total: 7 marks

Suppose you have a class `Person` with instance variables `String firstName` and `String lastName`. Provide methods to override the default methods in the following way:

(a) `toString` returns a person's full name.

(b) `equals` returns true if another `Person` object has the same first and last names as the given `Person`.

Question 5 ..... Total: 15 marks

Using the fragments below, complete the following code so that it compiles and prints

```
4 3 2 1
```

Note, you may use a fragment any number of times, including zero.

```
import java.util.*;

class Item {
        int size;
        Item(int s){
                size = s;
        }
}

public class MyGenerics {
        List __①__ p = new ArrayList __②__ ();

        class Comp implements Comparator<Item> {
                public int __③__(Item one, Item two){
                        return __④__ - __⑤__;
                }
        }

        public static void main(String[] args) {
                new MyGenerics().go();
        }

        void go(){
                p.add(new Item(4));
                p.add(new Item(1));
                p.add(new Item(3));
                p.add(new Item(2));

                Comp c = new Comp();
                Collections.sort( __⑥__ , __⑦__ );
                for( __⑧__ x : p)
                        System.out.print( __⑨__ + " ");
        }
}
```

The fragments are:

`<Item>`	`<MyGenerics>`	`<Integer>`	
`Item`	`MyGenerics`	`Integer`	
`compare`	`sort`	`p`	`c`
`one.size`	`two.size`	`x.size`	`p.size`

You do not need to rewrite the program. You may just indicate which fragment should be substituted for each of ① to ⑨.

Question 6 ..... Total: 10 marks

What is printed as a result of executing the following program?

```
public class ArrayTest {
  public static void main(String[] args) {
    int[] test = new int[2];
    test[0] = test[1] = 5;
    System.out.println(test[0] + "," + test[1]);
    fiddle(test, test[1]);
    System.out.println(test[0] + "," + test[1]);
  }

  static void fiddle(int[] test, int element) {
    test[0] = 10;
    test[1] = 11;
    element += 7;
    System.out.println(test[0] + "," + test[1] + "," + element);
    test = new int[2];
    test[0] = 20;
    test[1] = 21;
    System.out.println(test[0] + "," + test[1]);
  }
}
```

Question 7 ..... Total: 15 marks

Examine the following statements and indicate which are *true* and which are *false*.

(a) All methods in an abstract superclass must be declared `abstract`.

(b) A class declared `final` cannot be subclassed.

(c) A redefinition of a superclass method in a subclass need not have the same signature as the superclass method.

(d) A constructor is a special method with the same name as the class that is used to initialise the members of a class object.

(e) A method declared `static` can access both static and non-static class members.

(f) An array subscript may be any numerical expression.

(g) If a method in a superclass is overloaded in a subclass, the subclass method may have the same parameter list as the superclass method provided it has a different return type.

(h) All non-static methods in an interface must be abstract.

(i) Interfaces can only contain method signatures.

(j) Sometimes `this.meth()` and `super.meth()` refer to the same method.

Question 8 ..... Total: 17 marks

(a) Describe the four access regimes from `public` to `private` that may be applied to Java member variables and methods. Why are they useful? | 6 marks | 6 marks

(b) When you extend a class, the constructor for your new class will reference a constructor of the parent class, and this latter constructor may have any of the four possible access regimes. Comment on the consequences of each of the four possibilities. [6 marks] [6 marks]

(c) If the only constructor for a class is marked as `private`, is it ever possible to have an instance of that class or any subclass of it? Explain why or why not. [5 marks] [5 marks]

Question 9 ..... Total: 15 marks

You are given the following incomplete class declaration:

```
public class TimeRecord {
  private int hours;

  private int minutes; // 0 <= minutes < 60

  /** @param extraTime adds extraTime to this TimeRecord */
  public void add(TimeRecord extraTime) {
    /* YOUR CODE SHOULD GO HERE */
  }
  // ... other methods not shown
}
```

(a) Provide accessors and mutators for the class. [5 marks]

(b) What should replace [6 marks]

`/* YOUR CODE SHOULD GO HERE */`

so that the method `add(TimeRecord extraTime)` will correctly add `extraTime` to this `TimeRecord`.

(c) The following declaration that appears in a client program: [4 marks]

```
TimeRecord[] timeCards = new TimeRecord[100];
```

You may assume that `timeCards` has been initialised with `TimeRecord` objects. Consider the following code segment that is intended to set `total` to the sum of all the times stored in `timeCards`:

```
TimeRecord total = new TimeRecord(0,0);

for (TimeRecord t : timeCards)
    /* missing expression */
```

What should replace `/* missing expression */` so that the code segment will work as intended?

Question 10 ..... Total: 16 marks

Consider the following three class declarations:

```
1  public class Constructor {
2    private String s;
3    protected int x;
4
5    public Constructor() {
6      System.out.println("[1]" + this);
7    }
8
```

```
9     public Constructor(String s) {
10      this();
11      this.s = s;
12      System.out.println("[2]" + this);
13    }
14
15    public String toString() {
16      return " s = " + this.s + " x = " + this.x;
17    }
18  }
19
20  class SubConstructor extends Constructor {
21    protected String s;
22
23    SubConstructor(int x) {
24      super("label");
25      x = this.x;
26      System.out.println("[4] x = " + x);
27    }
28
29    public SubConstructor(String s) {
30      this(12);
31      this.s = s;
32      x = 19;
33      System.out.println("[5] x = " + x);
34    }
35  }
36
37  class SubSubConstructor extends SubConstructor {
38    private String s;
39
40    public SubSubConstructor() {
41      super("item");
42      s = super.s;
43      x++;
44      System.out.println("[6] [s = " + this.s + "] " + this);
45    }
46
47    public String toString() {
48      return "******* " + super.toString();
49    }
50  }
```

Provide a trace of the execution of the program when the following line of code is executed:

```
SubSubConstructor ssc = new SubSubConstructor();
```

Question 11 ..... Total: 18 marks

What would be the result of executing the following code?
You should show your working.

```
1  public class MaxClass {
2    public static void main(String args[]) {
3      try {
4        C c = new C();
5        System.out.println(c.max(13, 29));
6      } catch (RuntimeException rte) {
7        System.out.println(rte);
8      } finally {
9        System.out.println("In finally of main");
10     }
11   }
12 }
13
14 class A {
15   int max(int x, int y) {
16     try {
17       if (x > y) x++;
18       else throw new Exception("Oh Dear!");
19       System.out.println("A::max value of x is " + x);
20     } catch (Exception ex) {
21       System.out.println("In exception " + ex.getMessage());
22       System.out.println("x = " + x + " y = " + y);
23       return y;
24     } finally {
25       System.out.println("A::max finally block");
26       throw new IllegalArgumentException("A::max Finally x = " + x);
27     }
28   }
29 }
30
31 class C extends A {
32   public int max(int x, int y) {
33     return super.max(x + 10, y + 10);
34   }
35 }
```

Question 12 ..... Total: 12 marks

In an ordinary queue elements are removed in a first-in-first-out (FIFO) manner. In a *priority queue*, however, each element has a fixed numerical priority and, when an element is removed, it is the element in the queue with the highest priority that is removed. The following code creates a priority queue and inserts a few strings with various integer priorities. The `extractMax` method returns the element with the highest priority and removes it from the queue. In the `while` loop the elements are removed one by one in descending order of priority.

```
PriorityQueue<String> names = new PriorityQueue<String>();

names.insert("Bob", 5);
names.insert("Susan", 10);
names.insert("Dave", 7);
names.insert("Lisa", 3);
```

```
while (!names.isEmpty())
        System.out.print(names.extractMax());
```

The output will be:

```
Susan Dave Bob Lisa
```

The queue will work with any specified type of object and the priorities are always integers. An `ArrayList` is used for the underlying collection storage. The inner class `QueueEntry<T>` is used to wrap each element and its priority into a single object. The elements in the `ArrayList` are stored in ***ascending*** order of priority.

You should complete the outline code provided below for the `PriorityQueue` class by adding the required code fragments at the indicated points (1) to (5).

If `extractMax` is called when the priority queue is empty, a `NoSuchElement` exception should be raised.

```
import java.util.ArrayList;

public class PriorityQueue<T> {

  private ArrayList<QueueEntry<T>> elements;

  private class QueueEntry<T> { // private inner class

    private int priority;
    private T elm;

    public QueueEntry(T elm, int priority) {
      this.priority = priority;
      this.elm = elm;
    }
  }

  public PriorityQueue() {
    // YOUR CODE HERE (1)
  }

  public void insert(T elm, int priority) {
    int index;
    for (index = 0; index < elements.size(); index++)
      //           if (/* YOUR CODE HERE (2) */) // check for the correct position
      break;
    // YOUR CODE HERE (3) --- insert the element
  }

  public T extractMax() {
    if (elements.isEmpty()) {
      // YOUR CODE HERE (4)
    }

    // elements in increasing order, max element is in the last slot
```

```
    // YOUR CODE HERE (5)

    return null;
  }
}
```


Extract from the API for `java.util.ArrayList`

`ArrayList()`	`Constructs an empty list with an initial capacity of ten.`
`boolean add(Object elem)`	`Appends the specified element to the end of this list and returns true.`
`void add(int index, Object element)`	`Inserts the specified element at the position index in this list.`
`Object set(int index, Object element)`	`Replaces the element at position index with the specified element.`
`Object get(int index)`	`Returns the element at the specified position in this list; throws an exception if the index is out of range.`
`Object remove(int index)`	`Removes the element at the specified position in this list and returns the element removed.`
`int size()`	`Returns the number of elements in this list.`

