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

WITH OUTLINE SOLUTIONS

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

Solution: Object-oriented programming is a programming paradigm based on the specification, implementation, and use of objects.

(b) Object — 1 mark

Solution: An object is a data type with structure and state. Each object defines methods that can access and manipulate the state. These methods exist to be called by other objects.

(c) Class — 1 mark

Solution: A class is a structure representing an entity that contains fields and methods relating to this entity. An object is a specific instance of a class.

(d) Information hiding — 1 mark

Solution: Information hiding is the concept of hiding implementation details and making certain fields and methods of your object inaccessible.

(e) Encapsulation

Solution: Encapsulation is the combination of grouping related data and operations in a common object together with hiding the implementation of this object from its users.

(f) Inheritance — 1 mark

Solution: Inheritance is the mechanism used to implement hierarchical relationships.

(g) Polymorphism — 1 mark

Solution: Polymorphism is allowing a reference type to refer to objects of other, different reference types.

(h) Derived class — 1 mark

Solution: A derived class is a class that inherits properties and methods from a base class. It is also known as a *subclass*. It can extend the based class' functionality by adding its own fields and methods, and it can change the base class' functionality by overriding methods.

(i) Interface — 1 mark

Solution: An interface is a class where NO methods are implemented but only declared.

(j) Unit test — 1 mark

Solution: Unit testing is a procedure used to validate that individual modules or units of source code.

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.

Solution: Should mention some of the following:

- dynamic binding,
  ad-hoc polymorphism, runtime binding.
- cast expression,
  Coerce the type — might throw an exception.
- overriding,
  redefine a method in the inheritance chain. Should mention the method signature.
- `this`
  refers to the current object.
- `super`
  refers to the parent class.

+ appropriate examples.

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.]

Solution:

```
[P, Q, T, s, u]
```

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.

Solution:

① `<Item>`

② `<Item>`

③ `compare`

④ `two.size`

⑤ `one.size`

⑥ `p`

⑦ `c`

⑧ `Item`

⑨ `x.size`

1 mark for each slot + 0.5 mark each for getting p/c and two/one.size in the correct order.

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]);
  }
}
```

Solution:

```
5,5
10,11,12
20,21
10,11

0.5 and 2 and 0.5 and 2
```

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.

Solution: 1 mark for correct
1/2 off for incorrect
0 for no attempt

(a) *false*

(b) *true*

(c) *true*

(d) *true*

(e) *false*

(f) *false*

(g) *false*

(h) *true*

(i) *false*

(j) *true*

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]

Solution: Should list:

public Can be accessed from any class in a program.

protected The method or member variable in a class, can be accessed in subclasses (direct or indirect) of the class in which it is defined, but it cannot be used in non-subclasses. (There is one exception: A protected member can also be accessed by any class in the same package as the class that contains the protected member.)

package The item can be accessed from anywhere in the package that contains the class, but not from outside that package.

private The item can be accessed only from inside the same class.

If they only provide the list then just 1 mark.

(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]

Solution: `private` — cannot construct object of subclass because of no access
package — to construct the object of the subclass it would need to be in the same package as the parent class
`public` and `protected` has no effect.

(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]

Solution: Yes — but you need to use a static method of the class to return an instance of the class (*Factory method*).

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]

Solution: See following answer for code.

(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`.

Solution:

```
public class TimeRecordSolution {
  private static final int MINSINHOUR = 60;

  private int hours;

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

  public void add(TimeRecordSolution extraTime) {
    minutes += extraTime.getMinutes();
    if (minutes >= MINSINHOUR) {
      minutes -= MINSINHOUR;
      hours++;
    }
    hours += extraTime.getHours();
  }

  public int getHours() {
    return hours;
  }

  public void setHours(int hours) {
    this.hours = hours;
  }

  public int getMinutes() {
    return minutes;
  }

  public void setMinutes(int minutes) {
    if (minutes < 0 || minutes >= MINSINHOUR)
            throw new IllegalArgumentException();
    this.minutes = minutes;
  }
}
```

(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?

Solution:

```
total.add(t);
```

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();
```

Solution:

```
[1]*******  s = null x = 0
[2]*******  s = label x = 0
[4] x = 0
[5] x = 19
[6] [s = item] *******  s = label x = 20
```

1 per slot in line + 2 for ordering + 1 for stars on first two lines.

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  }
```

Solution:

```
In exception Oh Dear!
x = 23 y = 39
A::max finally block
java.lang.IllegalArgumentException: A::max Finally x = 23
In finally of main
```

2 for each line + 2.

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;
  }
}
```

Solution: For the five "blocks" of code:

1. `elements = new ArrayList<QueueEntry<T>>();`

   1 mark

2. `priority < elements.get(index).priority`

   2 marks

3. `elements.add(index, new QueueEntry<T>(elm, priority));`

   2 marks

4. `throw new NoSuchElementException();`

   1 mark

5. ```
   QueueEntry<T> qe = elements.get(elements.size()-1);
   elements.remove(elements.size()-1);
   return qe.elm;
   ```

   4 marks
   ```

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.`