IVE Information Technology

Information & Communications Technology

Programme Board

Instructions:

(a) This paper has a total of ELEVEN pages including the covering page.

(b) This paper contains TWO Sections.

(c) Section A is WORTH 40 marks and Section B is WORTH 60 marks.

(d) Section A contains FOUR questions. Answer ALL questions in Section A.

(e) Section B contains THREE questions. Answer ALL questions in Section B.

Note: The result of this assessment will not be counted if you do not meet the minimum attendance requirement (if any) governed by the general academic regulations of your programme/course unless approval of the campus principal has been granted.

HIGHER DIPLOMA IN

SOFTWARE ENGINEERING

(IT114105)

MODULE TITLE:

DATA STRUCTURES & ALGORITHMS: CONCEPTS AND IMPLEMENTATION

MODULE CODE: **ITP4510**

SEMESTER TWO
MAIN EXAMINATION

17 MAY, 2021
1:30 PM TO 3:30 PM (2 hours)

This paper contains <u>TWO</u> sections.

Section A (40 marks)

This section contains <u>4</u> questions.

Answer <u>ALL</u> questions.

A1 Consider the following Java program, answer the following questions.

```
import java.util.*;

class DivideByZException extends RuntimeException {}

public class EuroExchange {

    public static void main(String[] args) {
        try {
            Scanner kb = new Scanner(System.in);
            System.out.print("HKD ? ");
            double hkd = kb.nextDouble();
            System.out.print("Rate ? ");
            double rate = kb.nextDouble();
            if (rate == 0)
                throw new DivideByZException();
            double eur = hkd / rate;
            System.out.println(hkd + " HKD = " + eur + " EUR");
        } catch (InputMismatchException e) {
            System.out.println("Wrong input");
        } catch (DivideByZException e) {
            System.out.println("Rate cannot be zero");
        } catch (Exception e) {
            System.out.println("Something wrong");
        } finally {
            System.out.println("Enjoy");
        }
    }

}
```

(a) State the output of the programme if the inputs are HKD = **<u>200</u>**, rate = **<u>10</u>**. [2 marks]

(b) State the output of the programme if the inputs are HKD = **<u>40</u>**, rate = **<u>three</u>**. [2 marks]

(c) State the output of the programme if the inputs are HKD = **<u>65</u>**, rate = **<u>0</u>**. [2 marks]

(d) Describe the best way to modify the program to prevent user from inputting negative value (e.g. -9.8) as rate. [4 marks]

A2 Consider the following Java program, answer the following questions.

```
interface Flyable {
    void fly(double distance);
}

abstract class Vehicles {
    int capacity;
    abstract void startEngine();
    abstract void setDestination(int d);
}

abstract class FlyingVehicles extends Vehicles implements Flyable {
    String model;

    void startEngine() {
        System.out.println("Engine started");
    }
}

class Helicopter extends FlyingVehicles implements Flyable {
    String serialNo;
}

class Duck implements Flyable {
    String name;
    public void fly(double distance) {
    }
}

public class OnTheSky {
    public static void main(String[] args) {
        // main program is here
    }
}
```

(a) What method(s) MUST be implemented in class ***Helicopter*** before the program can be compiled successfully? [4 marks]

(b) Can the following program statements be compiled successfully? Explain why or why not. [3 marks]

```
Flyable[] a = new Flyable[6];   //line 1
a[4] = new Duck();              //line 2
a[4].fly(800);                  //line 3
```

(c) Can the following program statements be compiled successfully? Explain why or why not. [3 marks]

```
Vehicles b = null;               //line 4
b = new FlyingVehicles();        //line 5
```

A3 Consider the following classes.

```
class LinkedList {

    private ListNode head, tail;
    public LinkedList() { ...}
    public boolean isEmpty() { ... }
    public void addToTail(Object obj) { ... }
    public Object removeFromTail() throws EmptyListException { ... }
    public void addToHead(Object obj) { ... }
    public Object removeFromHead() throws EmptyListException { ... }
}

class LinkedStack {

    private LinkedList s = new LinkedList();

    public boolean empty() {
        /* missing segment (i) */
    }

    public void push(Object item) {
        s.addToHead(item);
    }

    public Object pop() {
        /* missing segment (ii) */
    }

   public Object top() {
      /* missing segment (iii) */
    }
}
```

The above class ***LinkedStack*** is a container of items that maintains the Last-In-First-Out (LIFO) property.

(a) Complete the missing code segments (i), (ii) and (iii) of the above ***LinkedStack***. [7 marks]

(b) The following series of operations are performed on an empty stack one after one. List the contents of the stack after **EACH** operation has completed. [3 marks]

Operations

(i) Push 300
(ii) Pop
(iii) Push 600
(iv) Push 900
(v) Top
(vi) Pop

A4 Consider the following Queue implemented using Array inside.

```
class QueueFullException extends RuntimeException {}
class QueueEmptyException extends RuntimeException {}

public class ArrayQueue {
   private int capacity;
   private Object [ ] array;
   private int front=0;
   private int rear=0;

   public ArrayQueue(int capacity) {
       this.capacity = capacity;
       array = new Object[capacity];
   }

   public int size() {
       return (rear - front + capacity) % capacity;
   }

   public boolean isEmpty() {
       /* To be completed in (a) */
   }

   public void enqueue (Object item) throws QueueFullException {
       if (size() == capacity-1)
            throw new QueueFullException();
       array [rear] = item;
       rear = (rear+1) % capacity;
   }

   public Object dequeue() throws QueueEmptyException {
       if (isEmpty())
            throw new QueueEmptyException();
       Object item = array [front];
       array [front] = null;
       front = (front+1) % capacity;
       return item;
   }

   public Object front() throws QueueEmptyException {
   }

   public String toString() {
       String s = "front:" + front + ", rear:" + rear;

       s += ", [ ";
       int next=front;

       for (int i=0; i<size(); i++) {
            s += array[next] + " ";
            next = (next + 1) % capacity;
       }
       return s + "]";
   }
}
```

***** Question A4 continues in next page *****

***** Question A4 continues from previous page *****

```
public class QueueTest {
   public static void main(String [] args) {
       ArrayQueue q = new ArrayQueue(5);

       System.out.println(q);

       q.enqueue("Cat");
       System.out.println(q);

       q.enqueue("Fish"); q.enqueue("Dog");
       System.out.println(q);

       q.dequeue(); q.dequeue();
       System.out.println(q);

       q.enqueue("Monkey"); q.enqueue("Tiger"); q.enqueue("Goat");
       q.dequeue(); q.dequeue();  q.dequeue();
       System.out.println(q);
   }
}
```

(a) Complete the method **`isEmpty()`** which returns **true** if the queue is empty, otherwise returns **false**. [2 marks]

(b) Write the output of the above program. The first two lines of output are provided as below. [6 marks]

```
front:0, rear:0, [ ]
front:0, rear:1, [ Cat ]
```

(c) What is the maximum number of objects can be stored in the following queue? [2 marks]

```
ArrayQueue q = new ArrayQueue(100);
```

Section B (60 marks)
This section contains 3 questions. Each question carries 20 marks.
Answer ALL questions.

B1 Consider the following classes ListNode, LinkedList, and EmptyListException.

```
class ListNode {
     Object data;
     ListNode next;
     ListNode(Object o) { data = o; next = null; }
     ListNode(Object o, ListNode node) { data = o; next = node; }
} // class ListNode

class EmptyListException extends RuntimeException {
     public EmptyListException () {
     super("List is empty"); }
} // class EmptyListException

class LinkedList {
     private ListNode head;
     private ListNode tail;
     public LinkedList() { head = tail = null; }
     public boolean isEmpty() { ... }
     public void addToHead(Object item) { ... }
     public void addToTail(Object item) { ... }
     public Object removeFromHead()throws EmptyListException { ... }
     public Object removeFromTail()throws EmptyListException { ... }
} // class LinkedList
```

(a) Complete the method **isEmpty ( )** of the LinkedList class. The method returns true if the Linked List is empty. Otherwise, it returns false. [2 marks]

(b) Complete the method **addToHead ( )** of the LinkedList class. The method adds a new node containing the input parameter ***item*** of type ***Object*** as data to the head of the Linked List. [5 marks]

(c) What is the value of the variable tail in the LinkedList if the LinkedList is empty? [1 mark]

(d) Describe how to keep track the number of ListNode contained in a LinkedList. [4 marks]

(e) Rewrite the class of ListNode to fulfill the requirement if the information of client ID (int) and client name (String) will be stored in the ListNode instead of storing Object data. [4 marks]

***** Question B1 continues in next page *****

***** Question B1 continues from previous page *****

(f) The LinkedList is now implemented with the ClientComparator to maintain the nodes in ascending order of Client Name.

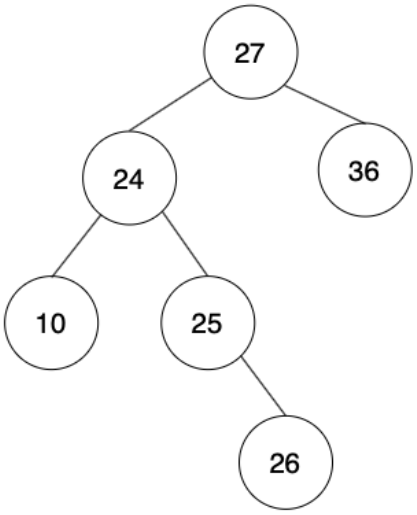
```
public interface Comparator {
     public abstract boolean isLessThan (Object item1, Object item2);
     public abstract boolean isGreaterThan (Object item1, Object item2);

}

public class ClientComparator  implements  Comparator {
     public boolean isLessThan(String s1, String s2) {
        return (s1.compareTo(s2) < 0) ;
     }
     public boolean isGreaterThan(String s1, String s2) {
        return (s1.compareTo(s2) > 0);
     }
}
```

Draw a diagram to represent the content and the ordering for the LinkedList after following operations have completed. [4 marks]

Operations for the LinkedList:

1. Add a client with Client ID **246513** and Client Name **Wong Tin Kin**.
2. Add a client with Client ID **243830** and Client Name **Ho Lam**.
3. Add a client with Client ID **241123** and Client Name **Tang Ki Lam**.
4. RemoveFromTail
5. Add a client with Client ID **256320** and Client Name **Lee Kin Hoi**.
6. Add a client with Client ID **240039** and Client Name **Wong Tai Tin**.

B2 Given the following Binary Search Tree (BST) with the inserting sequence: [27,24,10,25,36,26]



(a) List the traversal result of the above BST. [4 marks]

(i) In-order traversal
(ii) Post-order traversal

(b) Build an AVL tree using the same data set and inserting sequence. Draw the result of the insertion for the AVL tree **step by step**. [9 marks]

(c) Complete the following table about search performance for different data structures with the above same set of data. [3 marks]

	Number of comparisons needed for Best Case	Number of comparisons needed for Worst Case
AVL Tree		
Binary Search Tree		
Sequential Search		

(d) Given the set of numbers in the orders of [42,37,40,53,72,81]. [4 marks]

(i) Build a BST using the given data set in (d).
(ii) List the resulted array for the BST in (d)(i) that is implemented with array structure.

B3 (a) You are asked to sort the following sequence of numbers in ascending order:
31, 28, 55, 16, 72, 9, 53, 48

(i) Perform an **Insertion Sort** on the above sequence of numbers. Show the result of **each pass** of the sorting. The first pass is shown below for your reference (number(s) in [ ] is/are sorted) :

31, 28, 55, 16, 72, 9, 53, 48
[28, 31], 55, 16, 72, 9, 53, 48 [3 marks]

(ii) What is the time complexity (in Big-O notation) for a **Selection Sort** on data already **in ascending order**? [1 mark]

(iii) Perform a **Merge Sort** on the above sequence of numbers. Show your steps in partitioning and merging the list clearly. [4 marks]

(iv) Copy and complete the following tables on your answer book to show how the **Quick Sort** algorithm (using the first number as the pivot) partitions the above sequence of number into two sub-lists. The first two steps are shown below for your reference:

index	**0 (pivot)**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
data	31	28	55	16	72	9	53	48
Step 1	31	28						
Step 2	31	28	55					
Step 3								
Step 4								
Step 5								
Step 6								
Step 7								
Step 8 (swap pivot)								

storeIndex
1
2
2

[4 marks]

***** Question B3 continues in next page *****

***** Question B3 continues from previous page *****

(v) **Merge Sort** and **Quick Sort** have similar performance on random data.

(1) What is the time complexity (in Big-O notation) for a Merge Sort or a Quick Sort on random data? [1 mark]

(2) Even though **Merge Sort** and **Quick Sort** have similar performance on random data, Quick Sort is considered to be better than Merge Sort. Explain why. [1 mark]

(vi) **Quick Sort** has a much better performance than **Insertion Sort** on random data. However, in some special cases Insertion Sort will perform faster than Quick Sort. Describe, with explanation, under what condition Insertion Sort will have a better performance. [2 marks]

(b) What are the time complexities (in Big-O notation) of the following algorithms?

(i)
```
int sum = 0;
for (int i=1000; i<=n; i+=5)
   sum += i;
```
[1 mark]

(ii)
```
int sum = 0;
for (int i=1; i<=n; i++)
    for (int j=n; j>=i; j--)
        sum = sum + i*j;
```
[1 mark]

(iii)
```
int sum = 0;
for (int i=n-100; i<=n; i++)
   for (int j=1; j<=n; j*=3)
        sum += j;
```
[2 marks]

******* END OF PAPER *******