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

3 May, 2024
1:30 PM TO 3:30 PM (2 hours)

This paper contains TWO sections.

Section A (40 marks)
This section contains 4 questions.
Answer ALL questions.

A1 The following Java program calculates parking fee based on user's inputs.

```
import java.util.Scanner;

class NegativeHrExeception extends RuntimeException { }  // line 3

public class ParkingFee {
    public static void main(String[] args) {
        int[] rate = {18, 32, 44};
        Scanner kb = new Scanner(System.in);
        try {
            System.out.print("Type of car [0:Autos, 1:Bus, 2:Truck]");
            String s = kb.nextLine();            // line 11
            System.out.print("Hours parked:");
            int hr = kb.nextInt();               // line 13
            if (hr < 0)
                throw new NegativeHrExeception();
            int t = Integer.parseInt(s);
            int fee = hr * rate[t];
            System.out.println("Total fee is " + fee);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("AIOB");
        } catch (NumberFormatException e) {
            System.out.println("NF");
        } catch (NullPointerException e) {
            System.out.println("NP");
        } catch (NegativeHrExeception e) {
            System.out.println("Invalid hour");
        } catch (Exception e) {
            System.out.println("Something wrong");
        }
    }
}
```

(a) State the output of the programme if the inputs for line 11 and line 13 are "4" and "2" respectively. [2 marks]

(b) State the output of the programme if the inputs for line 11 and line 13 are "Bus" and "10" respectively. [2 marks]

***** Question A1 continues in next page *****

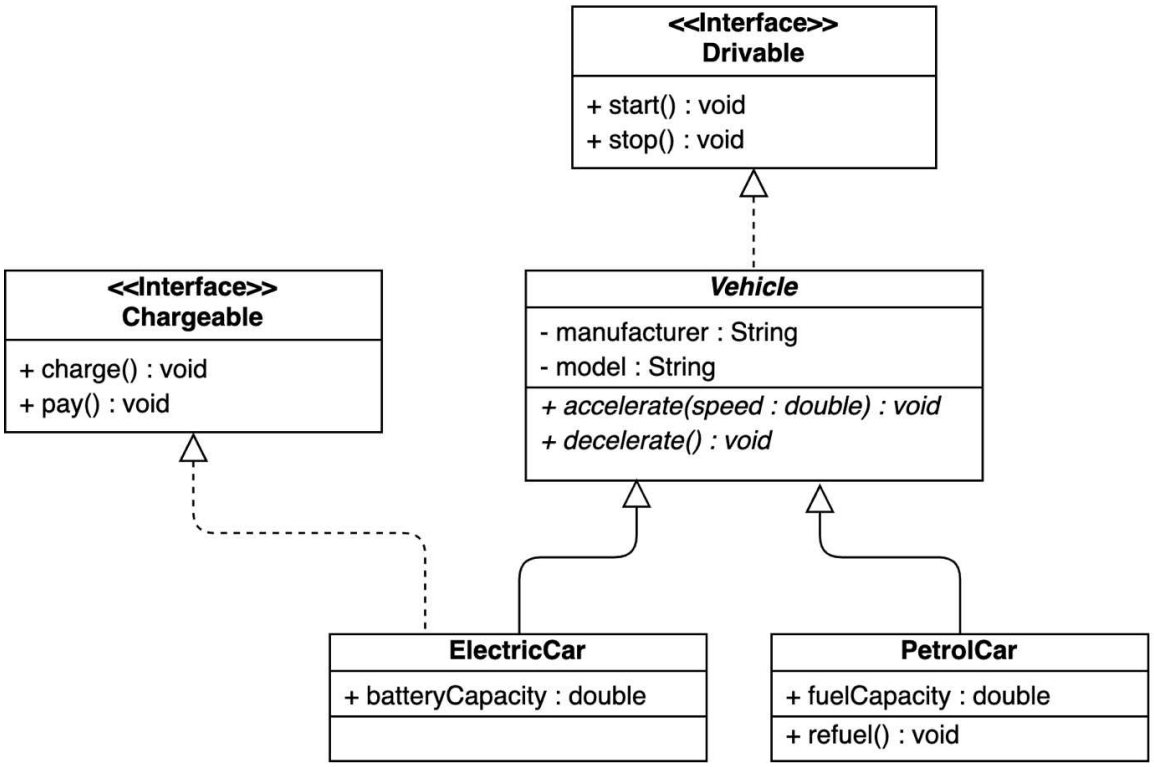
***** Question A1 continues from previous page *****

(c) State the output of the programme if the inputs for line 11 and line 13 are "1" and "four" respectively. [2 marks]

(d) State the output of the programme if the inputs for line 11 and line 13 are "2" and "-2" respectively. [2 marks]

(e) If the "RuntimeException" on line 3 is replaced with "Exception," will any code need to be modified? If the answer is yes, please describe the necessary changes. [2 marks]

A2 Consider the interfaces and classes in the following UML class diagram, answer the following questions. [10 marks]



Remarks: ***Vehicle***, ***accelerate***(), ***decelerate***() are in Italic form.

(a) Write the interface **Chargeable**.

(b) Write the **abstract** class **Vehicle**.

(c) Write the **class header** of **ElectricCar**.

(d) Except methods *charge()* and *pay()*, which **FOUR** methods **MUST** be implemented in the **ElectricCar** class?

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 ls = 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.

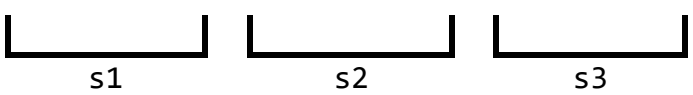
(a) Complete the missing code segments (i), (ii) and (iii) of the above LinkedStack. [6 marks]

(b) We have three stacks, **s1**, **s2** and **s3**, that can contain data of type String. Here are their initial contents: [4 marks]

As you can see, initially **s2** and **s3** are empty. Here is a sequence of operations on the three stacks:

```
s2.push(s1.pop());
s3.push(s1.pop());
s1.pop();
s1.push(s2.pop());
s2.push(s3.pop());
s2.push(s1.pop());
```

Copy and complete the following format in your answer book to show the content of the stacks after the operations are complete.



A4 Please answer the following questions about circular array-based queue:

(a) Consider a circular array-based queue with a capacity of 8. The front and rear indices of the queue are currently at positions 5 and 2, respectively. What is the total number of elements in the queue currently? [1 mark]

(b) Write a method **`isFull()`** to check if a given circular array-based queue is full, and return true if the queue is full. Assume the queue is implemented using an array of capacity 10 and has the following attributes :

```
Object[] queue, int front, int rear, int capacity = 10
```

[2 marks]

(c) Assume the circular array-based queue **`Q`** with a capacity of 6 which has been cleared and reset ***(i.e., front = rear = 0)***. Copy and complete with the following table on your answer book to show the final contents of the array after the following code is executed: [3 marks]

```
for (int k = 1; k <= 6; k++) {
    Q.enqueue(k);
    if (k % 2 == 0) {
        Q.dequeue();
    }
}
```

Index	**0**	**1**	**2**	**3**	**4**	**5**
Content						

(d) A queue can also be implemented by using Linked List. State ONE advantage and ONE disadvantage of using Linked List instead of using Array. [4 marks]

Section B (60 marks)

This section contains 3 questions. Each question carries 20 marks.

Answer ALL questions.

B1 Consider the following classes.

```
public class ListNode {
    int roomID; String catName; int petChipID; ListNode next;
    public ListNode(int roomID, String catName, int petChipID){...}
    public ListNode(int roomID, String catName, int petChipID, ListNode next){...}
}  // Class ListNode

public class LinkedList {
    private ListNode head; private ListNode tail;

    public LinkedList() { head = tail = null;}
    /***  Check  if  there  is  any  node,  except  the  Header  Node  ***/
    public boolean isEmpty() { return (head ==null); }

    public void addToTail(int roomID, String catName, int petChipID) {
       /*** TO BE COMPLETED in B1 part (a)(i) ***/
    }

    public int catLookup(int target){
       if (isEmpty())
             throw new EmptyListException();
        /*** TO BE COMPLETED in B1 part (a)(ii) ***/
    }
}  // Class LinkedList

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

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

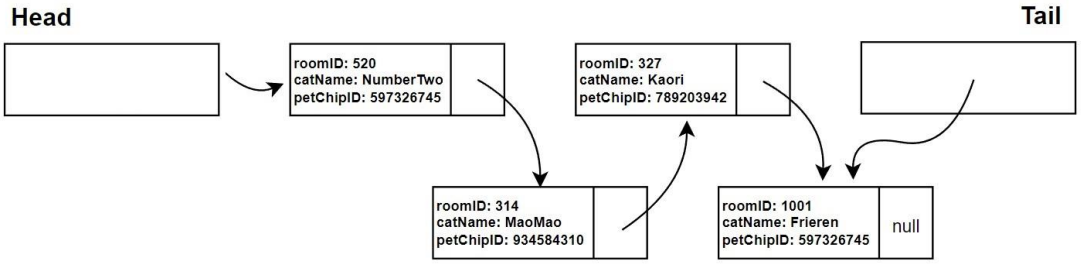
***** Question B1 continues from previous page *****

```
public interface Comparator {
    // returns true if value of cat A's petChipID is less than cat B's,
    // otherwise returns false.
    public abstract boolean isLessThan (int catA_ID, int catB_ID);

    // returns true if value of cat A's petChipID is greater than cat B's,
    // otherwise returns false.
    public abstract boolean isGreaterThan (int catA_ID, int catB_ID);
}

public class petChipIDComparator implements Comparator {
   /*** TO BE COMPLETED in B1 part (b)(i) ***/
}

public class CLinkedList {
    private ListNode head; private ListNode tail;
    private Comparator comparator;
    public LinkedList(Comparator comparator) {
        head = tail = new ListNode(true); this.comparator = comparator;
    }
    public boolean isEmpty() { return (tail.isHeader); }
    public void addToHead(int roomID, String catName, int petChipID){...}
    public void addToTail(int roomID, String catName, int petChipID){...}

    public  void  insertInOrder(int  roomID,  String  catName,  int  petChipID){
       /*** TO BE COMPLETED in B1 part b(ii) ***/
       if (isEmpty()){
           _____(1)_____;
       } else if (comparator.isGreaterThan(head.petChipID,petChipID)){
           addToHead(roomID, catName, petChipID);
       } else if (comparator.isLessThan(tail.petChipID, petChipID)){
           addToTail(roomID, catName, petChipID);
       } else { // insert in the middle
           /*** TO BE COMPLETED in B1 part b(ii) ***/
           ListNode current = _____(2)_____; // locate the header node
           while ( _____(3)_____ ) { // check boundary
               if( _____(4)_____ ) { // check ordering
                    ListNode newNode = new ListNode(roomID, catName, petChipID);
                    _____(5)_____
                    current.next = newNode;
               }
               current = current.next;
           }
    }
} // class CLinkedList
```

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

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

(a) The following diagram represents the Linked List used in the Cat Accommodation System (CAS) for domestic cats foster care service hotel. The nodes are inserted in **random** order. There is a Header Node added by default.



(i) Complete the method **addToTail** of the LinkedList class. The method adds the specified item as the node. [5 marks]

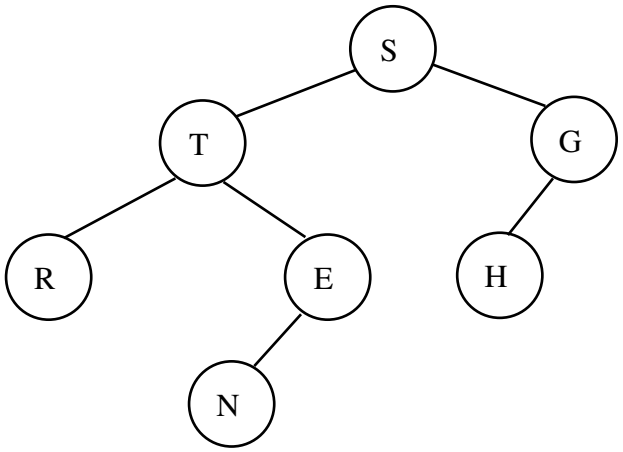
(ii) Complete the method **catLookup** of the LinkedList class. The method searches for the node with the specific ***petChipID*** and returns the corresponding node's ***roomID*** to the caller when the node is found. It will return 0 if node is not found. [6 marks]

(b) A comparator is implemented for the CAS.

(i) Complete the **petChipIDComparator** class. [3 marks]

(ii) Complete the method **insertInOrder()** of the LinkedList class. The method uses the comparator to insert the node in ascending order of ***petChipID***. [6 marks]

B2 Given a Binary Tree as below.



(a) List the traversal result of the above Binary Search Tree with each of the following traversal orders. [6 marks]

(i) Breadth-first traversal
(ii) Depth-first preorder traversal
(iii) Depth-first postorder traversal

(b) List the resulted array for the above binary tree that is implemented with array structure. [3 marks]

(c) Give a proper binary tree. [2 marks]

(d) What is the minimal height required if 8,000 nodes are stored in a binary tree? [2 marks]

(e) Build an AVL Tree using the following item set. Draw the result of the insertion for the AVL Tree **step by step**. [7 marks]

52, 30, 21, 77, 41, 48

B3 (a) You are asked to sort the following sequence of numbers in ascending order:
38, 67, 18, 36, 23, 45, 50, 5

(i) Perform a **Bubble 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) :

38, 67, 18, 36, 23, 45, 50, 5
38, 18, 36, 23, 45, 50, 5, [67] [3 marks]

(ii) What is the time complexity (in Big-O notation) for a **Selection Sort** on data **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 numbers into two sub-lists. The first two steps are shown below for your reference:

index	**0 (pivot)**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**storeIndex**
data	38	67	18	36	23	45	50	5	1
Step 1	38	67							1
Step 2	38	18	67						2
Step 3									
Step 4									
Step 5									
Step 6									
Step 7									
Step 8 (swap pivot)									

[4 marks]

(v) Explain why **Quick Sort** is considered to be better than **Merge Sort** in terms of space. [2 marks]

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

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

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

(i)
```
int sum = 0;
for (int i=n+100; i>=n; i--)
   sum += i;
```
[1 mark]

(ii)
```
int sum = 0;
for (int i=1; i<=n*n; i+=2)
        sum += i;
```
[1 mark]

(iii)
```
int f(int n) {
   if (n==0 || n==1) return 1;
   return 1+f(n/3);
}
```
[2 marks]

(iv)
```
long sum = 0;
for (int i=n/2; i<=n; i++)
   for (int j=1; j<=n; j*=2)
        sum += (i*j);
```
[2 marks]

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