CSC248

FUNDAMENTALS OF DATA STRUCTURES

SEMESTER OCTOBER 2020 – FEBRUARY 2021

GROUP: CS1103C

AHMAD NAZIF BIN AZELAN (2020852256)

MUHAMMAD HAZIQ DANISH BIN MOHD KAMARUZIHAN (2020870808)

MUHAMAD HANIF FARHAN BIN SAIPOL BAHRI (2020452026)

NURUL SYAZWINA SYAHIRAH BINTI MOHD AZIZ (2020893698)

ROGER CANDA

FSKM, UiTM Pahang

TABLE OF CONTENTS

1. GROUP MEMBERS INFORMATION

MEMBERS	INFORMATIONS
	Full Name: AHMAD NAZIF BIN AZELAN Student ID: 2020852256 Group: CS1103C Contact Number: 013-3938214
	Full Name: MUHAMMAD HAZIQ DANISH BIN MOHD KAMARUZIHAN Student ID: 2020870808 Group: CS1103C Contact Number: 013-2454762
	Full Name: MUHAMAD HANIF FARHAN BIN SAIPOL BAHRI Student ID: 2020452026 Group: CS1103C Contact Number: 011-56627681
	Full Name: NURUL SYAZWINA SYAHIRAH BINTI MOHD AZIZ Student ID: 2020893698 Group: CS1103C Contact Number: 019-3153746

2. SOURCE CODES

Queue.java

```
public class Queue extends LinkedList
{
   public Queue() {}

   public void enqueue(Cake element) {
      addLast(element);
   }

   public Cake dequeue() {
      return removeFirst();
   }

   public Cake getFront() {
      return getFirst();
   }
     }
```

Node.java

```
public class Node {
   Cake element;
   Node next;

   public Node(Cake element) {
      this.element = element;
   }
     }
```

LinkedList.java

```
public class LinkedList {

   private Node head, current, tail;

   public LinkedList() {
      head = current = tail = null;
   }

   public boolean isEmpty() {
      return head == null;
   }

   public void addFirst(Cake element) {
      Node newNode = new Node(element);
      newNode.next = this.head;
      this.head = newNode;
      if(this.tail == null) {
         this.tail = this.head;
      }
   }

   public void addLast(Cake element) {
      Node newNode = new Node(element);

      if(this.tail == null) {
         this.head = this.tail = newNode;
      }
      else {
         this.tail.next = newNode;
         this.tail = this.tail.next;
      }
   }
```

```
}

public Cake getFirst() {
    if (this.isEmpty()) {
        return null;
    }
    else {
        this.current = this.head;
        return this.current.element;
    }
}

public Cake getLast() {
    if (this.isEmpty()) {
        return null;
    }
    else {
        return this.tail.element;
    }
}

public Cake getNext() {
    if (this.current == this.tail) {
        return null;
    }
    else {
        this.current = this.current.next;
        return this.current.element;
    }
}

public void clear() {
    this.head = this.current = this.tail = null;

}

public boolean contains(Cake element) {
        boolean isContain = false;
        this.current = this.head;

        while (this.current != null) {
            if (element.equals(this.current.element)) {
                isContain = true;
                break;
            }
        }

        return isContain;
    }

    public Cake removeFirst() {
        if (this.isEmpty()) {
            return null;
        }
        else {
            this.current = this.head;
            this.head = this.head.next;
            if (this.head == null)
                this.tail = null;
            return current.element;
        }
    }

    public Cake removeLast() {
        if (this.isEmpty())
            return null;
        else if (this.head == this.tail) {
            this.current = this.head;
            this.head = this.tail = null;
            return current.element;
        }
        else {
```

```
        this.current = this.head;
        while (this.current.next != tail) {
            this.current = this.current.next;
        }
        Node temp = this.tail;
        this.tail = this.current;
        this.tail.next = null;
        return temp.element;
    }
}

public Cake removeAfter(Cake element) {
    if (this.isEmpty()) {
        return null;
    }
    else if (this.head == this.tail) {
        this.current = this.head;
        this.head = this.tail = null;
        return current.element;

    }
    else {
        Node previous = this.head;
        while (previous.next != null) {
            if (element.equals(previous.element))
            {
                break;
            }
            previous = previous.next;
        }
        current = previous.next;
        previous.next = current.next;
        return current.element;
    }
}
```

```
public String toString() {
    StringBuilder result = new StringBuilder("[");
    if (this.isEmpty()) {
        result.append("The list is empty]");
    }
    else {
        this.current = this.head;
        while (this.current != null) {
            result.append(this.current.element);
            this.current = this.current.next;
            if (this.current != null)
                result.append(", ");
            else
                result.append("]");
        }
    }
    return result.toString();
}
   }
```

Cake.java

```
/**
 *
 * Group members name and matric number:
 * 1) AHMAD NAZIF BIN AZELAN [2020852256]
 * 2) MUHAMMAD HAZIQ DANISH BIN MOHD KAMARUZIHAN [2020870808]
 * 3) MUHAMAD HANIF FARHAN BIN SAIPOL BAHRI [2020452026]
 * 4) NURUL SYAZWINA SYAHIRAH BINTI MOHD AZIZ [2020893698]
 * Class : CS1103C
 */
import java.lang.*;
public class Cake
{
  private String custID; //D001, P003
  private String cakeType; //D24 Chocolate Cake,Red Velvet,Burnt Cheese Cake,Black
     Forest
  private int qty;
  private double price;

  public Cake(String ID,String cakeType, int qty)
  {
     this.custID=ID;
     this.cakeType=cakeType;
     this.qty=qty;
  }

  public void setID(String ID){this.custID=ID;}
  public void setCakeType(String cakeType){this.cakeType=cakeType;}
  public void setQty(int qty){this.qty=qty;}

  //2.a)Write the retriever method for custID, cakeType and qty
  public String getID() {return this.custID;}
  public String getCakeType() {return this.cakeType;}
  public int getQty() {return this.qty;}
```

```
//2.b)Write the detPrice() method that will return the price of cake based on
    cakeType.Refer 2_GroupProject.docx
public double detPrice()
{
    if (this.getCakeType().equalsIgnoreCase("D24 Chocolate Cake"))
        price = 120.00;
    else if (this.getCakeType().equalsIgnoreCase("Red Velvet"))
        price = 80.00;
    else if (this.getCakeType().equalsIgnoreCase("Burnt Cheese Cake"))
        price = 100.00;
    else if (this.getCakeType().equalsIgnoreCase("Black Forest"))
        price = 80.00;
    return price;
 }
//2.c)Write the toString method that will return the output look like as below:
//Customer ID:XXXX    Cake Type:XXXXXXX          Price: RM XX.XX
    Quantity:XX
 public String toString()
   {
      return "Customer ID: "+ this.getID() +
      " Cake Type: "+this.getCakeType()+" Price: RM "
      + this.detPrice() +" Quantity: "+this.getQty();
    }
    }
```

MyAss2QApp.java

```
import java.util.*;
import java.io.*;
import java.lang.*;
public class MyAss2QApp {
   public static void main(String[] args) throws Exception
   {
      try
      {
         BufferedReader br = new BufferedReader (new FileReader ("cakeOrder.txt"));
         PrintWriter pickOut = new PrintWriter (new FileWriter ("pickup.txt"));
         PrintWriter delOut = new PrintWriter (new FileWriter ("delivery.txt"));
         Queue cakeQ=new Queue();
         Queue tempQ=new Queue();

         //a)b)c)read the data from cakeOrder.txt and insert into cakeQ
         String str= br.readLine(); //read data in cakeOrder.txt
         Cake c;                  //declares variable to store cakeOrder data

         while (str !=null) //while loop to go through every data in cakeOrder.txt, condition
     checks whether if str is null or not
         {
            StringTokenizer token= new StringTokenizer (str, "*"); //to separate data

            String id= token.nextToken(). trim();                 //get custID
            String type= token.nextToken(). trim();               //get cakeType
            int qtty= Integer.parseInt(token.nextToken(). trim()); //get quantity

            c= new Cake(id, type, qtty); //insert data in c in order
            cakeQ.enqueue(c);            //enqueues data c into cakeQ
            str= br.readLine();          //read another line of data
         }

         //d)display the data in the cakeQ
```

```
    System.out.println("Data in CakeQ: ");
    Cake d = cakeQ.dequeue(); //dequeues the first data in cakeQ
    while(d != null) //while loop to go through every data in cakeQ, condition checks
whether if d is null or not
    {
        System.out.println(d.toString()); //prints the details of Cake d
        tempQ.enqueue(d);               //stores Cake d into tempQ temporarily while
cakeQ is dequeueing
        d = cakeQ.dequeue();            //retrieve the next data in cakeQ to continue the
loop
    } //while loop ends

    //e)The first character of custID is based on the delivery type. if the first character
is 'P'
    //mean the customer choose to pickup the cake and if the first character is 'D', the
customer
    //choose to have delivery service. Example of custID are P002,D112 and etc. Write
the data for delivery
    //into delivery.txt output file and the data for customer that choose self pick up into
pickup.txt.

    pickOut.println("Data for self-pickup:\n"); //print in pickOut.txt
    delOut.println("Data for delivery:\n"); //print in delOut.txt

    int cp = 0, cd = 0;  //declares two integer variables, as counters for each service
    d = tempQ.dequeue(); //dequeues data in tempQ
    while(d != null) //while loop to go through every data in tempQ, condition checks
whether if d is null or not
    {
        if(d.getID().charAt(0) == 'D') //checks if the id starts with a 'D' for delivery
service
        {
            delOut.println((cd+=1) + ")" + d.toString()); //writes the details of the cake
order into delivery.txt
        }
        else
        {
            pickOut.println((cp+=1) + ")" + d.toString()); //writes the details of the cake
```

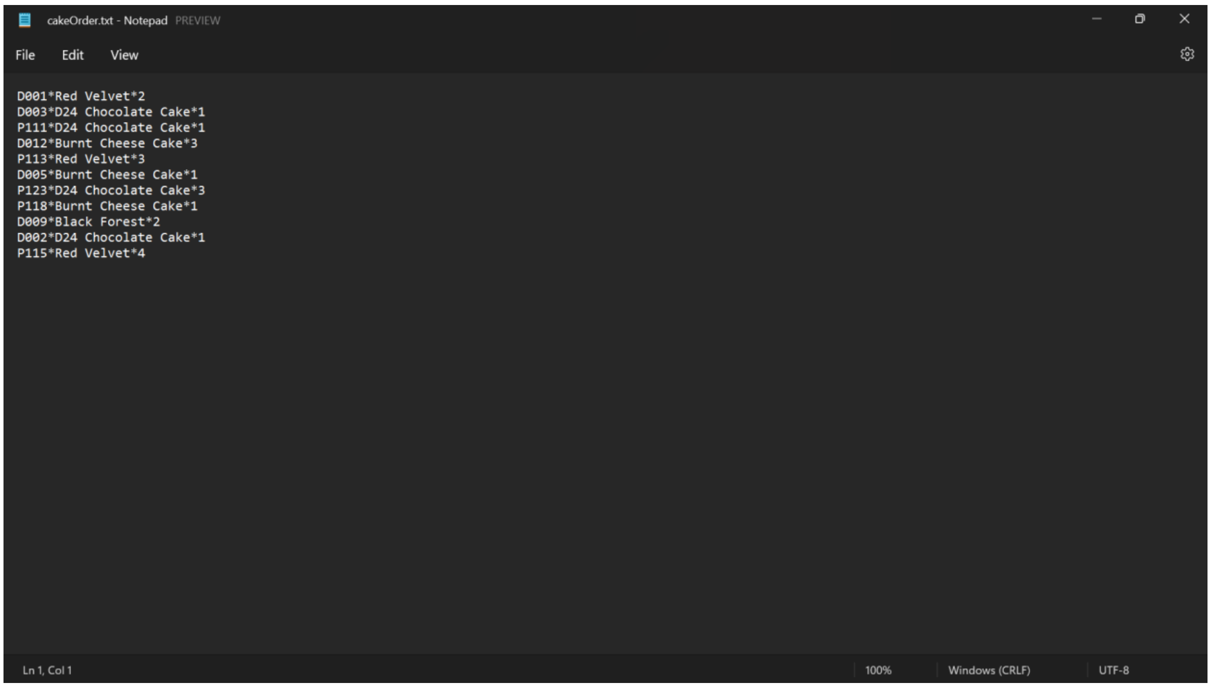
```
order into pickup.txt
        }
        cakeQ.enqueue(d);    //stores Cake d into cakeQ while tempQ is dequeueing
        d = tempQ.dequeue(); //retrieve the next data in tempQ to continue the loop
    } //while loop ends

    //f)Display the total quantity order for each cake type and display the cake name of
the highest total order

    int qtyOrder[] = new int[4]; //declares an array of integers with the capacity of 4,
for each cake, 0 = d24, 1 = rv, 2 = bcc, 3 = bf
    String cake[] = {"D24 Chocolate Cake","Red Velvet","Burnt Cheese Cake","Black
Forest"}; //declares and initialize an array of strings
    //that stores the name of each cake type, = d24, 1 = rv, 2 = bcc, 3 = bf

    d = cakeQ.dequeue(); //dequeues the first data in cakeQ
    while(d != null) //while loop to go through every data in cakeQ, condition checks
whether if d is null or not
    {
        if (d.getCakeType().equalsIgnoreCase(cake[0]))      //checks if the cake type
equals "D24 Chocolate Cake"
          qtyOrder[0]+=d.getQty();                     //adds the cake's quantity into D24
Chocolate Cake's counter variable
        else if (d.getCakeType().equalsIgnoreCase(cake[1])) //checks if the cake type
equals "Red Velvet"
          qtyOrder[1]+=d.getQty();                     //adds the cake's quantity into Red
Velvet's counter variable
        else if (d.getCakeType().equalsIgnoreCase(cake[2])) //checks if the cake type
equals "Burnt Cheese Cake"
          qtyOrder[2]+=d.getQty();                     //adds the cake's quantity into Burnt
Cheese Cake's counter variable
        else
          qtyOrder[3]+=d.getQty();                     //adds the cake's quantity into Black
Forest's counter variable

        tempQ.enqueue(d);    //stores Cake d into tempQ temporarily while cakeQ is
dequeueing
        d = cakeQ.dequeue(); //retrieve the next data in cakeQ to continue the loop
    } // while loop ends
    int highest = 0;  //initializes an integer variable for the highest quantity of cake
    String high = ""; //String variable for the cake type with highest quantity

    for (int i=0;i<qtyOrder.length;i++)  //for loop to go through each quantity of cake
    {
        if(qtyOrder[i] > highest) //checks if the quantity of the current index is higher
than the highest variable
        {
            highest = qtyOrder[i]; //sets the integer variable of the highest quantity as the
quantity with the current index
            high = cake[i];        //sets the string variable of the highest quantity as the
name with the current index
        }
    } //for loop ends

    System.out.println("\nTotal quantity of D24 Chocolate Cake: " + qtyOrder[0]);
//displays the total quantity for each cake
    System.out.println("Total quantity of Red Velvet: " + qtyOrder[1]);
//displays the total quantity for red velvet cake
    System.out.println("Total quantity of Burnt Cheese Cake: " + qtyOrder[2]);
//displays the total quantity for burnt cheese cake
    System.out.println("Total quantity of Black Forest: " + qtyOrder[3]);
//displays the total quantity for black forest cake
    System.out.println("\nHighest total order is " + highest + " for " + high);
//displays the highest quantity with the name of the cake

    //g)Display the receipt that will display the custID, cakeType, price(using
detPrice() method), qty,
    //payment for each order. In order to calculate the payment for each order you need
to multiply quantity
    //with the cake price and it is an extra charge of RM 5.00 for delivery service.
Lastly, display the total
    //payment for all the orders.

     int count = 0; double total = 0; //declares and initializes 2 variables for the count
of customers and total amount of payment
     d = tempQ.dequeue();            //dequeues the first data in tempQ
```

```
   while(d != null) //while loop to go through every data in tempQ, condition checks
whether if d is null or not
   {
      System.out.println("\n++++++++++++++++++++++++++++++++++");
      System.out.println("Customer no " + (count+=1));                    //displays the
number of the customer, increment with every loop
      System.out.println("++++++++++++++++++++++++++++++++++");
      System.out.println("Customer ID: " + d.getID());                    //displays the id
of the customer
      System.out.println("Cake Name: " + d.getCakeType());                  //displays
the cake type
      System.out.println("Price: " + String.format("RM %.2f", d.detPrice()));
//displays the price, set the precision to 2 decimal places
      System.out.println("Quantity: " + d.getQty());                     //displays the
quantity of the cake
      String del = "";                                              //String variable to display
whether the order requests for delivery service or self pick-up
      double payment = 0;                                            //double variable to
calculate the payment for each customer
      payment = d.getQty() * d.detPrice();                             //calculate the
payment by multiplying the quantity of the cake with the price

      if(d.getID().charAt(0) == 'D') //checks if the id starts with "D" for delivery
service
      {
         del = "Delivery"; //sets the String variable as "Delivery"
         payment += 5.00;  //adds RM 5.00 to the payment for delivery fee
      }
      else
      {
         del = "Self Pick-Up"; //sets the String variable as "Self Pick-Up"
      }

      total += payment; //adds the payment into the total variable

      System.out.println("Delivery/Self Pick-Up:" + del);                //displays the
service
      System.out.println("Payment: " + String.format("RM %.2f", payment));
//displays the payment, set the precision to 2 decimal places
```

```
        cakeQ.enqueue(d);                               //stores Cake d into cakeQ
while tempQ is dequeueing
        d = tempQ.dequeue();                             //retrieve the next data in
tempQ to continue the loop
    }//while loop ends
    System.out.println("\nTotal payment: " + String.format("RM %.2f",total));
//displays the total payment of all customers, set the precision to 2 decimal places


    br.close();      //closes the BufferedReader br
    pickOut.close(); //closes the PrintWriter pickOut
    delOut.close();  //closes the PrintWriter delOut
 }
 catch (EOFException eof) { System.err.println("Problem: "+eof.getMessage()); } //to
catch potential exceptions when the program is executed
 catch (FileNotFoundException fnfe) { System.err.println(fnfe.getMessage()); }
 catch (IOException io) { System.err.println(io.getMessage()); }
 catch (Exception e) { System.err.println(e.getMessage()); }
} /***End of main() Method***/
}/***End of Application Class***/
```

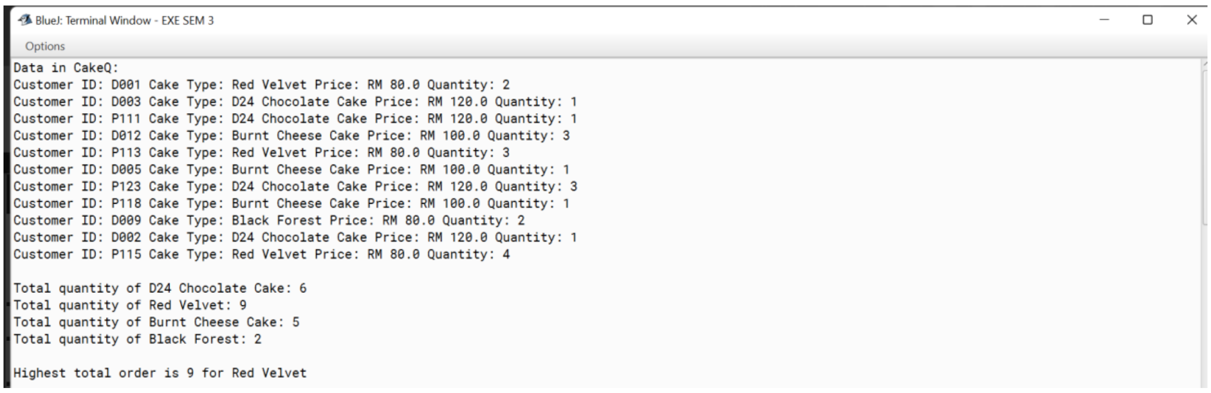
3. SCREEN SHOT INPUT FILES

cakeOrder.txt

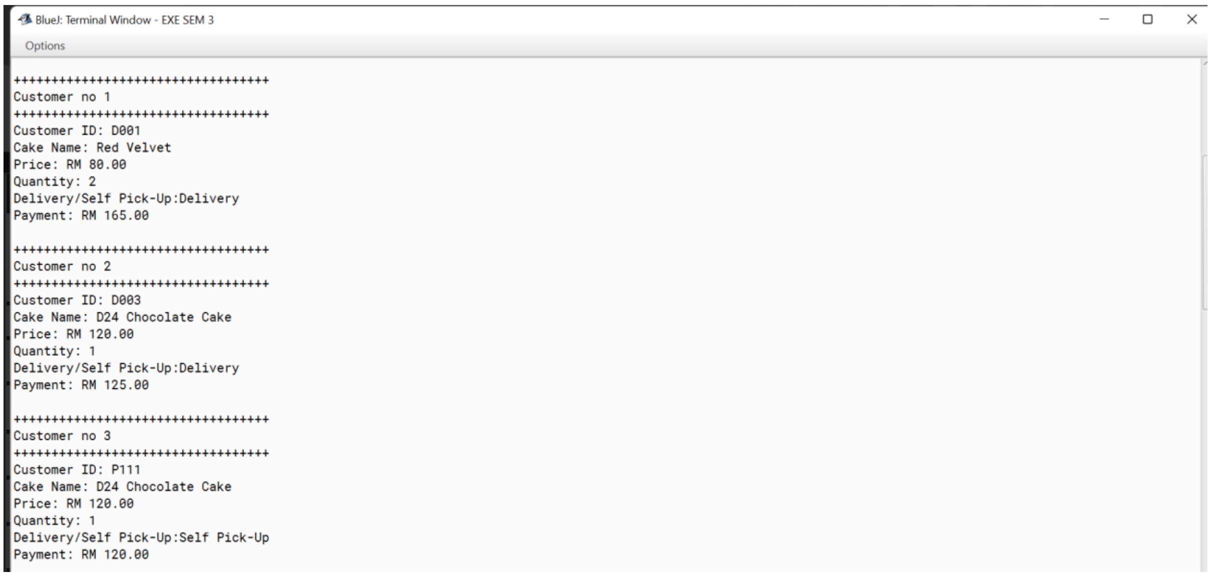


```
D001*Red Velvet*2
D003*D24 Chocolate Cake*1
P111*D24 Chocolate Cake*1
D012*Burnt Cheese Cake*3
P113*Red Velvet*3
D005*Burnt Cheese Cake*1
P123*D24 Chocolate Cake*3
P118*Burnt Cheese Cake*1
D009*Black Forest*2
D002*D24 Chocolate Cake*1
P115*Red Velvet*4
```

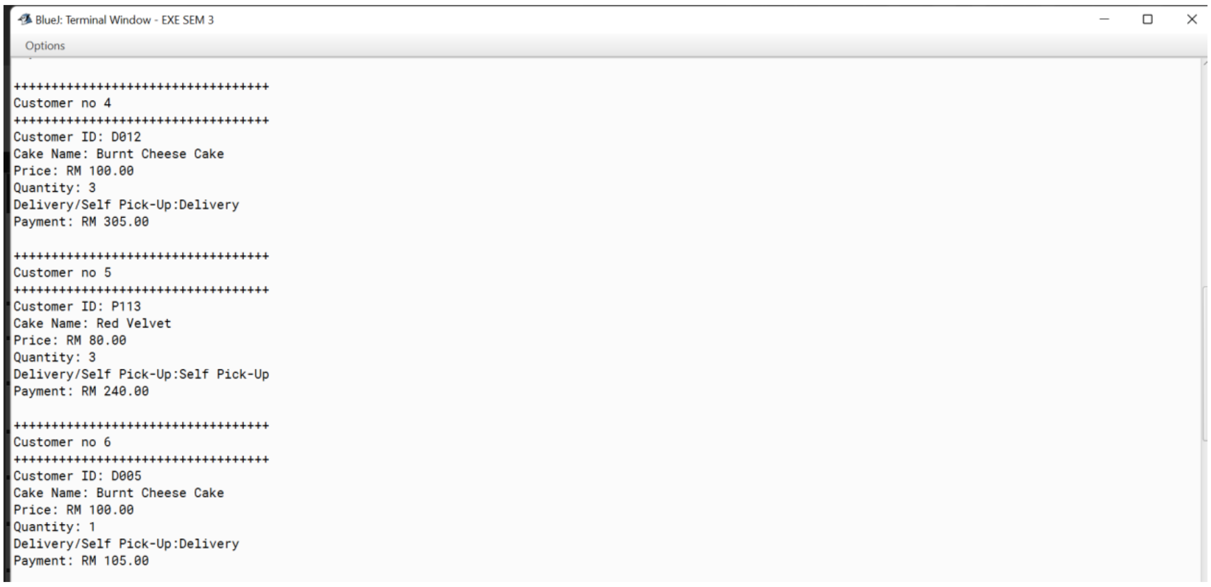
4. SCREEN SHOT SAMPLE OUTPUT ON SCREEN



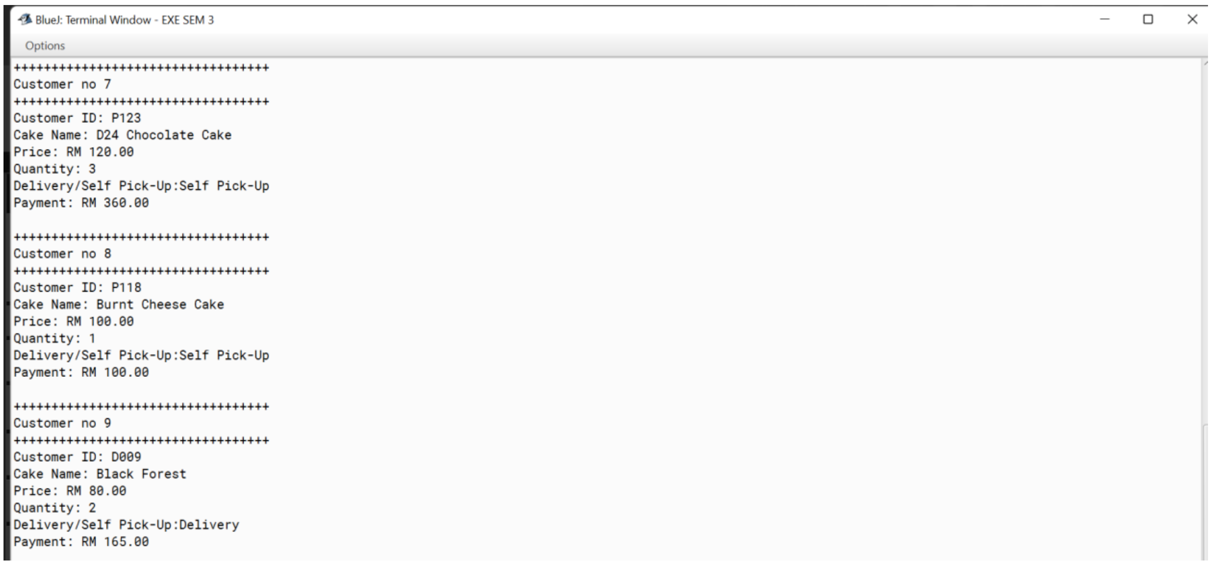
```
Data in CakeQ:
Customer ID: D001 Cake Type: Red Velvet Price: RM 80.0 Quantity: 2
Customer ID: D003 Cake Type: D24 Chocolate Cake Price: RM 120.0 Quantity: 1
Customer ID: P111 Cake Type: D24 Chocolate Cake Price: RM 120.0 Quantity: 1
Customer ID: D012 Cake Type: Burnt Cheese Cake Price: RM 100.0 Quantity: 3
Customer ID: P113 Cake Type: Red Velvet Price: RM 80.0 Quantity: 3
Customer ID: D005 Cake Type: Burnt Cheese Cake Price: RM 100.0 Quantity: 1
Customer ID: P123 Cake Type: D24 Chocolate Cake Price: RM 120.0 Quantity: 3
Customer ID: P118 Cake Type: Burnt Cheese Cake Price: RM 100.0 Quantity: 1
Customer ID: D009 Cake Type: Black Forest Price: RM 80.0 Quantity: 2
Customer ID: D002 Cake Type: D24 Chocolate Cake Price: RM 120.0 Quantity: 1
Customer ID: P115 Cake Type: Red Velvet Price: RM 80.0 Quantity: 4

Total quantity of D24 Chocolate Cake: 6
Total quantity of Red Velvet: 9
Total quantity of Burnt Cheese Cake: 5
Total quantity of Black Forest: 2

Highest total order is 9 for Red Velvet
```



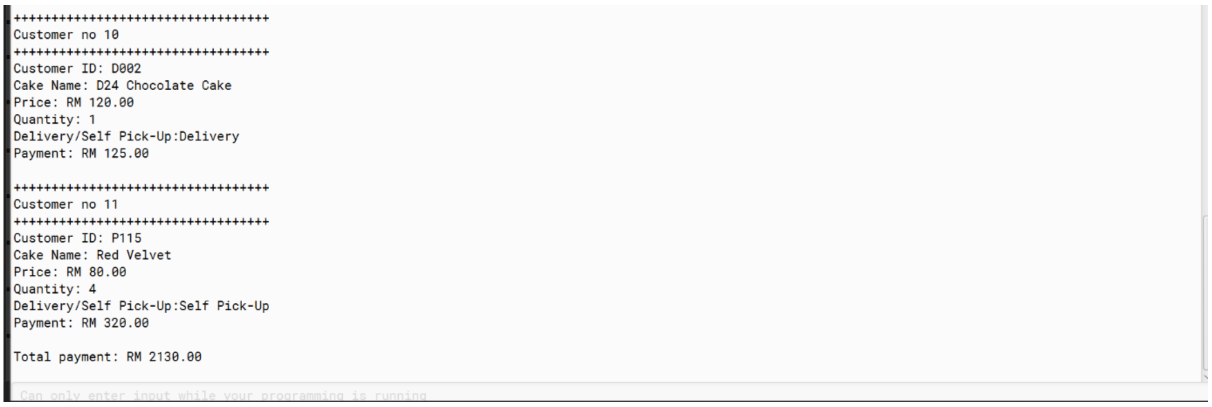
```
++++++++++++++++++++++++++++++++++
Customer no 1
++++++++++++++++++++++++++++++++++
Customer ID: D001
Cake Name: Red Velvet
Price: RM 80.00
Quantity: 2
Delivery/Self Pick-Up:Delivery
Payment: RM 165.00

++++++++++++++++++++++++++++++++++
Customer no 2
++++++++++++++++++++++++++++++++++
Customer ID: D003
Cake Name: D24 Chocolate Cake
Price: RM 120.00
Quantity: 1
Delivery/Self Pick-Up:Delivery
Payment: RM 125.00

++++++++++++++++++++++++++++++++++
Customer no 3
++++++++++++++++++++++++++++++++++
Customer ID: P111
Cake Name: D24 Chocolate Cake
Price: RM 120.00
Quantity: 1
Delivery/Self Pick-Up:Self Pick-Up
Payment: RM 120.00
```



```
++++++++++++++++++++++++++++++++++
Customer no 4
++++++++++++++++++++++++++++++++++
Customer ID: D012
Cake Name: Burnt Cheese Cake
Price: RM 100.00
Quantity: 3
Delivery/Self Pick-Up:Delivery
Payment: RM 305.00

++++++++++++++++++++++++++++++++++
Customer no 5
++++++++++++++++++++++++++++++++++
Customer ID: P113
Cake Name: Red Velvet
Price: RM 80.00
Quantity: 3
Delivery/Self Pick-Up:Self Pick-Up
Payment: RM 240.00

++++++++++++++++++++++++++++++++++
Customer no 6
++++++++++++++++++++++++++++++++++
Customer ID: D005
Cake Name: Burnt Cheese Cake
Price: RM 100.00
Quantity: 1
Delivery/Self Pick-Up:Delivery
Payment: RM 105.00
```



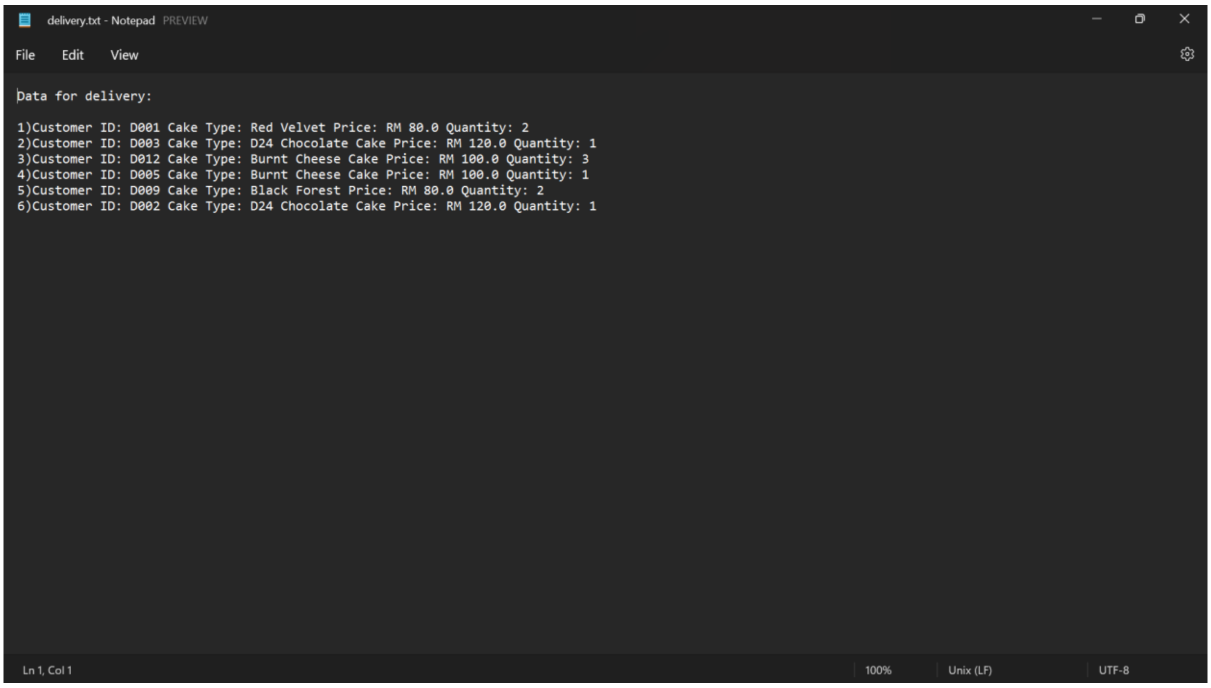
```
+++++++++++++++++++++++++++++++++++
Customer no 7
+++++++++++++++++++++++++++++++++++
Customer ID: P123
Cake Name: D24 Chocolate Cake
Price: RM 120.00
Quantity: 3
Delivery/Self Pick-Up:Self Pick-Up
Payment: RM 360.00

+++++++++++++++++++++++++++++++++++
Customer no 8
+++++++++++++++++++++++++++++++++++
Customer ID: P118
Cake Name: Burnt Cheese Cake
Price: RM 100.00
Quantity: 1
Delivery/Self Pick-Up:Self Pick-Up
Payment: RM 100.00

+++++++++++++++++++++++++++++++++++
Customer no 9
+++++++++++++++++++++++++++++++++++
Customer ID: D009
Cake Name: Black Forest
Price: RM 80.00
Quantity: 2
Delivery/Self Pick-Up:Delivery
Payment: RM 165.00
```



```
+++++++++++++++++++++++++++++++++++
Customer no 10
+++++++++++++++++++++++++++++++++++
Customer ID: D002
Cake Name: D24 Chocolate Cake
Price: RM 120.00
Quantity: 1
Delivery/Self Pick-Up:Delivery
Payment: RM 125.00

+++++++++++++++++++++++++++++++++++
Customer no 11
+++++++++++++++++++++++++++++++++++
Customer ID: P115
Cake Name: Red Velvet
Price: RM 80.00
Quantity: 4
Delivery/Self Pick-Up:Self Pick-Up
Payment: RM 320.00

Total payment: RM 2130.00
```

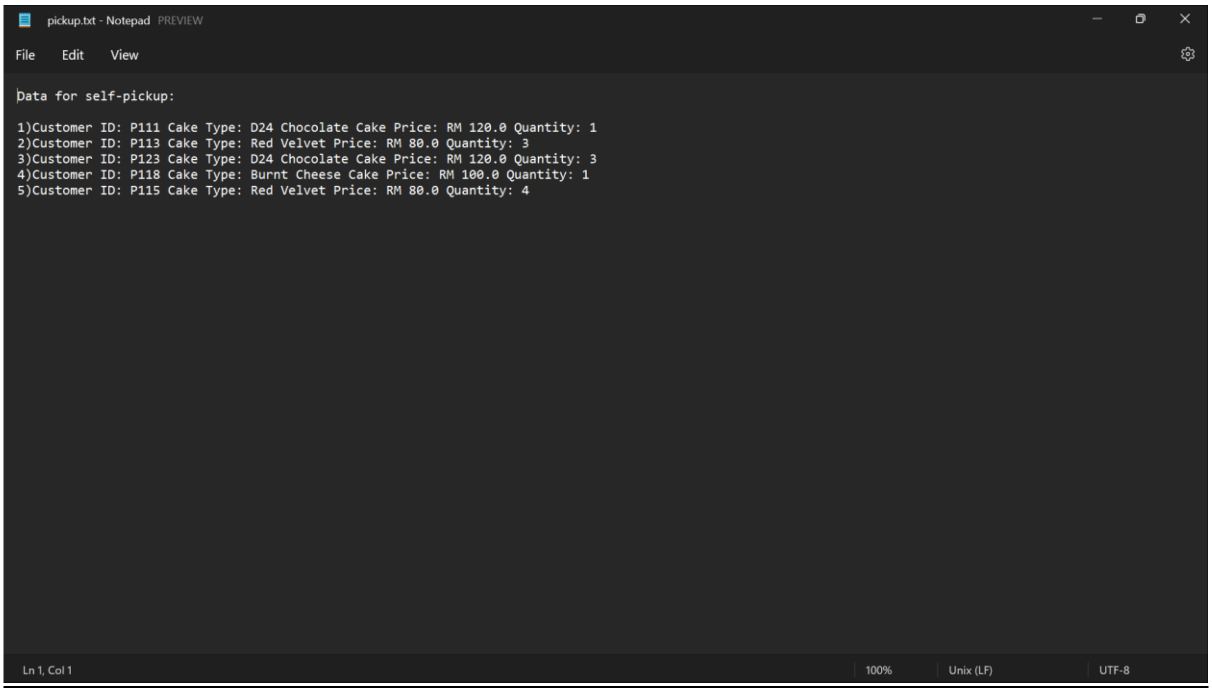
5. SCREEN SHOT OUTPUT FILES

delivery.txt



```
Data for delivery:

1)Customer ID: D001 Cake Type: Red Velvet Price: RM 80.0 Quantity: 2
2)Customer ID: D003 Cake Type: D24 Chocolate Cake Price: RM 120.0 Quantity: 1
3)Customer ID: D012 Cake Type: Burnt Cheese Cake Price: RM 100.0 Quantity: 3
4)Customer ID: D005 Cake Type: Burnt Cheese Cake Price: RM 100.0 Quantity: 1
5)Customer ID: D009 Cake Type: Black Forest Price: RM 80.0 Quantity: 2
6)Customer ID: D002 Cake Type: D24 Chocolate Cake Price: RM 120.0 Quantity: 1
```

pickup.txt



```
Data for self-pickup:

1)Customer ID: P111 Cake Type: D24 Chocolate Cake Price: RM 120.0 Quantity: 1
2)Customer ID: P113 Cake Type: Red Velvet Price: RM 80.0 Quantity: 3
3)Customer ID: P123 Cake Type: D24 Chocolate Cake Price: RM 120.0 Quantity: 3
4)Customer ID: P118 Cake Type: Burnt Cheese Cake Price: RM 100.0 Quantity: 1
5)Customer ID: P115 Cake Type: Red Velvet Price: RM 80.0 Quantity: 4
```

ASSESSMENT #2: GROUP PROJECT
CSC248: FUNDAMENTALS OF DATA STRUCTURES
QUEUE DATA STRUCTURE

Deadline: 7th January 2022, 11:59PM

REPORTS OF GROUP PROJECT (20%)

SCORING RUBRIC

No.	Name	Student ID	Mark
1	AHMAD NAZIF BIN AZELAN	2020852256	/ 30
2	MUHAMMAD HAZIQ DANISH BIN MOHD KAMARUZIHAN	2020870808	
3	MUHAMAD HANIF FARHAN BIN SAIPOL BAHRI	2020452026	
4	NURUL SYAZWINA SYAHIRAH BINTI MOHD AZIZ	2020893698	
Group	: CS1103C		
Project Title	: QUEUE DATA STRUCTURE		

Attribute	Attribute	1 - Very weak	2 - Weak	3 - Fair	4 - Good	5 - Very good
Problem Solving	Understanding DS Understands the Problem and Requirements	Student's work shows incomplete understanding of problem and/or requirements	Student's work shows slight understanding of problem and requirements	Student's work shows understanding of problem and most requirements	Student's work shows complete understanding of problem and all requirements	Student recognizes potential conflicts between requirements and seeks clarification from client/user
	Algorithm Uses Appropriate Algorithms	Student 'hacks out' program with no thought to algorithm design	Student chooses/ designs algorithm(s) that are incorrect	Student chooses/ designs algorithm(s) that is/are correct but somewhat inefficient	Student chooses/ designs efficient algorithm(s)	Student research trade-offs between different algorithms & implements the results of this research

ASSESSMENT #2: GROUP PROJECT
CSC248: FUNDAMENTALS OF DATA STRUCTURES
QUEUE DATA STRUCTURE

Deadline: 7th January 2022, 11:59PM

Attribute	Attribute	1 - Very weak	2 - Weak	3 - Fair	4 - Good	5 - Very good
	Select DS Uses Appropriate Data Structures	No use of ADTs (aggregate data types/structures)	Use of ADTs; but are none are appropriate for task	Use of ADTs; but some are not most appropriate for task	Use of ADTs; all are appropriate for task	Uses advanced ADTs that improves program performance
Learning Skills	Design Designs Appropriate User Interface	Implements very poor I/O functionality	Only implements basic I/O functionality	Some concepts of ‘user-friendly’ I/O used	Uses well-designed ‘user-friendly’ I/O interface appropriate for task and client	‘User-friendly’ I/O interface with GUI components
	Testing Tests Program for Correctness	No evidence of any testing by student	Evidence of only one case tested	Evidence of a few cases tested	Evidence of “typical cases tested, but only assuming valid inputs	‘Robust design’ with extensive testing.
	Documentation Documents Program	Absolutely no documentation other than name.	Little or no documentation; few or no internal comments	Some documentation, but sparse internal comments	Complete documentation with numerous internal comments	Thorough documentation;