

FACULTY OF APPLIED SCIENCES AND COMPUTING

TUNKU ABDUL RAHMAN UNIVERSITY COLLEGE

**BACS2063** Data Structures and Algorithms

**Assignment**

**Semester 1**

**2017/2018**

|  |  |  |  |
| --- | --- | --- | --- |
| Programme | : |  |  |
| Tutorial Group | : |  |  |
| Date Submitted | : |  |  |
| Tutors | : |  |  |

**Team Members:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Student Name** | **Student ID** | **Marks** | **Signature** |
| 1. |  |  |  |  |
| 2. |  |  |  |  |
| 3. |  |  |  |  |
| 4. |  |  |  |  |
|  | | | | |

**Declaration**

**We confirm that we have read and shall comply with all the terms and conditions of TAR University College’s plagiarism policy.**

**We declare that this assignment is free from all forms of plagiarism and for all intents and purposes is my own properly derived work.**

|  |  |  |
| --- | --- | --- |
| Signature | Name | Date |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Table of contents

|  |  |
| --- | --- |
| Item | Page |
| Module A | 4-14 |
| Part 1: ADT Specification | 4-5 |
| Part 2: ADT Implementation | 6-10 |
| Part 3: Source code listing & description | 11-14 |
|  |  |
| Module B | 15-24 |
| Part 1: ADT Specification | 15 |
| Part 2: ADT Implementation | 16-20 |
| Part 3: Source code listing & description | 21-24 |
|  |  |
| Module C | 27-64 |
| Part 1: ADT Specification | 25-27 |
| Part 2: ADT Implementation | 28-30 |
| Part 3: Source code listing & description | 31-64 |
|  |  |
| Module D | 65 |
| Part 1: ADT Specification | 65-66 |
| Part 2: ADT Implementation | 67-69 |
| Part 3: Source code listing & description | 70-84 |

**Module A**

**ADT Specification**

ADT List

**add(T newEntry);**

|  |  |  |
| --- | --- | --- |
| Description | : | Adds **newEntry** to the end of the list. |
| Postcondition | : | **newEntry** has been added to the end of the list. |

**remove(int givenPosition);**

|  |  |  |
| --- | --- | --- |
| Description | : | Remove the entry at givenPosition. |
| Precondition | : | **givenPosition** is >=0 and **givenPosition** is less than the size of list. |
| Postcondition | : | The entry at **givenPosition** has been removed. |

**clear();**

|  |  |  |
| --- | --- | --- |
| Description | : | Removes all data from the list. |
| Postcondition | : | The list is now empty. |

**T get(int givenPosition)**

|  |  |  |
| --- | --- | --- |
| Description | : | Retrieve the entry at givenPosition. |
| Precondition | : | The list is not null. |
| Postcondition | : | The list remain unchanged. |
| Returns | : | The entry at **givenPosition** has been retrieved. |

**boolean replace(T newEntry, int givenPosition);**

|  |  |  |
| --- | --- | --- |
| Description | : | Replaces the entry at position **givenPosition** with **newEntry**. |
| Precondition | : | **givenPosition** must be larger or equal to 0 and the list is not null. |
| Postcondition | : | The entry at **givenPosition** has been replaced with **newEntry**. |
| Returns | : | Return **true** if the entry is replace successfully, else return **false**. |

**boolean isEmpty();**

|  |  |  |
| --- | --- | --- |
| Description | : | Determines whether the list is empty. |
| Postcondition | : | The list remains unchanged. |
| Returns | : | **True** if the list is empty, else **False**. |

**boolean contains(T newEntry);**

|  |  |  |
| --- | --- | --- |
| Description | : | Check whether the list contain **newEntry.** |
| Precondition | : | List is not empty. |
| Postcondition | : | The list remain unchanged. |
| Returns | : | Return **true** if the entry is exist, else return **false**. |

**int size();**

|  |  |  |
| --- | --- | --- |
| Description | : | Check the size of the list |
| Postcondition | : | The list remain unchanged. |
| Returns | : | Size; |

**void display()**

|  |  |  |
| --- | --- | --- |
| Description | : | Display all the entry from list. |
| Precondition | : | List is not empty. |
| Postcondition | : | The list remain unchanged. |
| Returns | : | The entries. |

**ADT Implementation**

Implementation Classes Source Code

2 /\*

3 \* To change this license header, choose License Headers in Project Properties.

4 \* To change this template file, choose Tools | Templates

5 \* and open the template in the editor.

6 \*/

7 package ModuleA;

8

9 /\*\*

10 \*

11 \* @author Greyson

12 \*/

13 public class MyList<T> implements ListInterface<T>{

14 Node head = null;

15 Node tail = null;

16 int count = 0;

17

18 public void add(T newEntry){

19 if(head == null){

20 head = new Node(newEntry);

21 tail = head;

22 }

23 else{

24 tail.setNext( new Node(newEntry) );

25 tail = tail.getNext();

26 }

27

28 count++;

29 }

30

31 public void remove(int givenPosition){

32 if(givenPosition >= 0 && givenPosition < count){

33 Node temp = this.head;

34

35 if(givenPosition == 0){

36 this.head = head.getNext();

37 count--;

38 return;

39 }

40

41 // [0,1]->[1,2]-[2,null]

42 int i = 0;

43 Node previous = temp;

44 while(temp.getNext() != null){

45

46 if(i == givenPosition){

47 break;

48 }

49 previous = temp;

50 temp = temp.getNext();

51 i++;

52 }

53

54 previous.setNext(temp.getNext());

55 temp = null;

56 count--;

57 }

58 }

59

60 public void clear(){

61 while(this.size() != 0){

62 this.remove(0);

63 }

64 }

65

66 public T get(int givenPosition){

67

68 int i = 0;

69

70 Node<T> temp = head;

71

72 while(temp != null){

73

74 if(i == givenPosition){

75 return temp.getData();

76 }

77

78 temp = temp.getNext();

79 i++;

80 }

81

82 return null;

83 }

84

85 public boolean replace(T newEntry, int givenPosition) {

86 Node current = head;

87 Node prev = null;

88

89 while (current != null && givenPosition >= 0) {

90 givenPosition--;

91 prev = current;

92 current = current.next;

93 }

94

95 if (givenPosition > 0) return false;

96

97 if (prev != null)

98 prev.data = newEntry;

99

100 return true;

101 }

102

103 public boolean isEmpty() {

104 boolean result;

105

106 if (count == 0) {

107 result = true;

108 } else {

109 result = false;

110 }

111

112 return result;

113 }

114

115

116

117

118

119

120 public void display(){

121

122 Node temp = head;

123

124 while(temp != null){

125

126 System.out.println("\n " + temp);

127 temp = temp.getNext();

128

129 }

130 }

131

132 public boolean contains(T newEntry){

133

134 Node temp = head;

135

136 while(temp != null){

137

138 if(newEntry.equals(temp.getData())){

139 return true;

140 }

141

142 temp = temp.getNext();

143 }

144

145 return false;

146 }

147

148 public int size(){

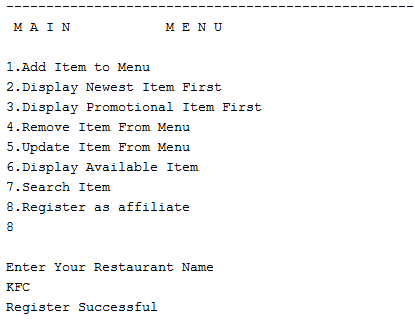
149 return this.count;

150 }

**Application Program**

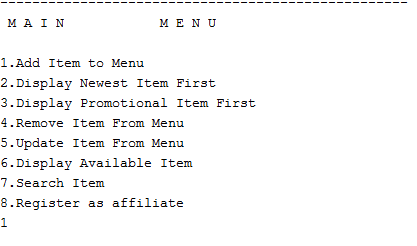
1. Register

In the main menu, enter 8 to perform this action. Enter Restaurant name.

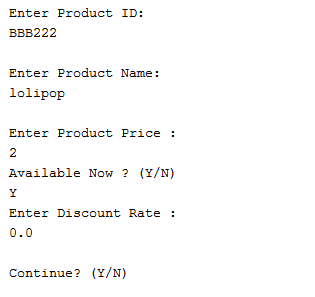


1. Add Item to menu.

In the main menu, enter 1 to perform this action.

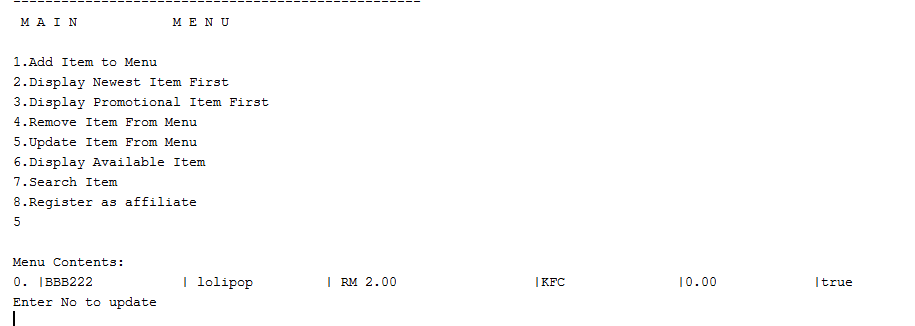


System will prompt the form to add item.



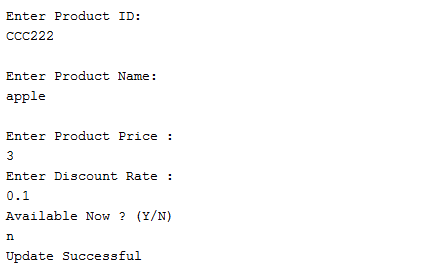
1. Update Item

In the main menu , Enter 5 to update item.



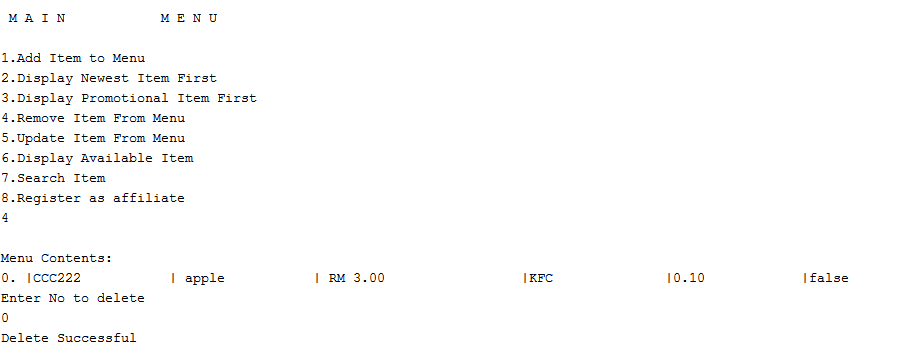
Enter No to update item.

System will prompt the form to update item.



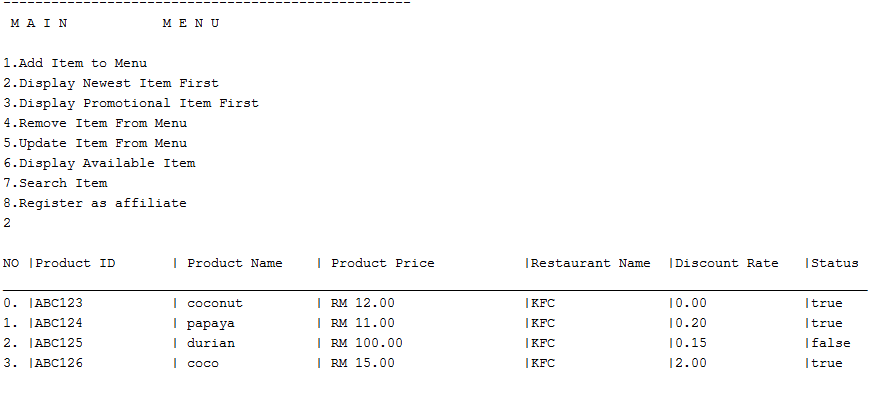
1. Remove Item

In the main menu, Enter 4 to remove item. Enter No to delete item.



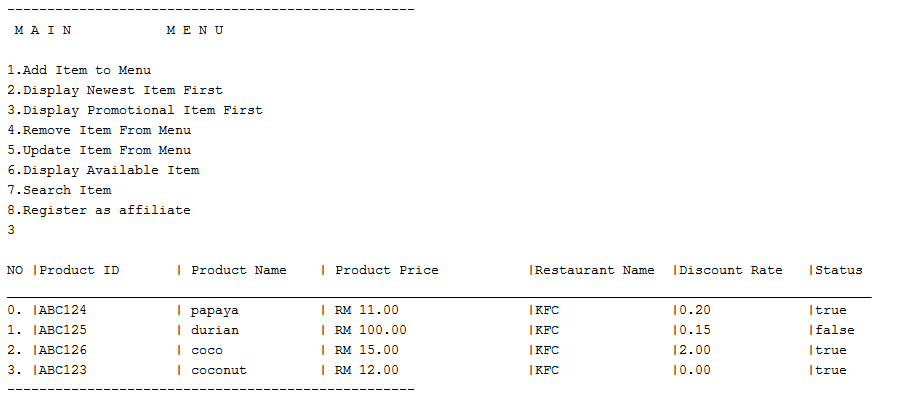
1. Display Newest Item First

In the main menu, Enter 2 to Display newest item.



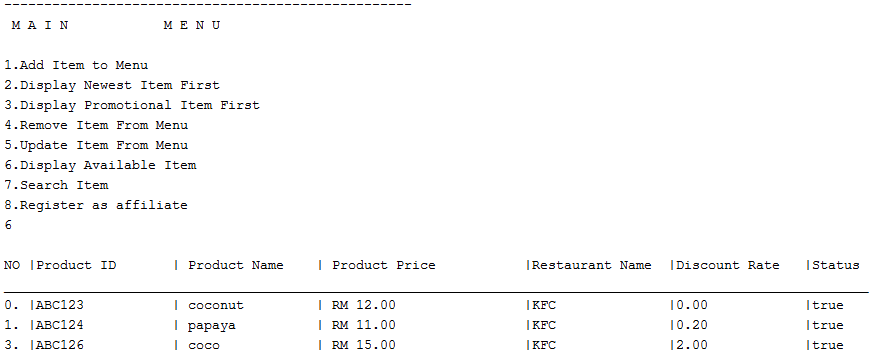
1. Display Promotional Item first

In the main menu, Enter 3 to Display promotional item first.



1. Display Available Item

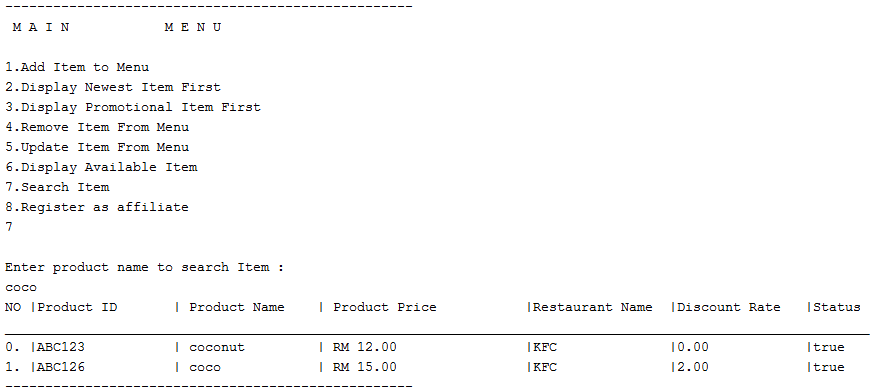
In the main menu, Enter 6 to Display Available Item.



1. Search Item.

In the main menu, Enter 7 to Search Item.

Enter product name, system will retrieve the similar product.



**Module B**

**ADT S pecification**

ADT List

**add(T newEntry)**

|  |  |  |
| --- | --- | --- |
| Description | : | Adds **newEntry** to the end of the list. |
| Postcondition | : | **newEntry** has been added to the end of the list. |

**clear()**

|  |  |  |
| --- | --- | --- |
| Description | : | Removes all entries from the list. |
| Postcondition | : | The list is now empty. |

**boolean update(Integer givenPosition, T newEntry)**

|  |  |  |
| --- | --- | --- |
| Description | : | Replaces the entry at position **givenPosition** with **newEntry**. |
| Precondition | : | **newPosition** must be between1 to total entries. |
| Postcondition | : | The entry at **givenPosition** has been replaced with **newEntry**. |
| Returns | : | **true** if the entry at **givenPosition** was successfully replaced in the list; **false** otherwise. |

**T getEntry(Integer givenPosition)**

|  |  |  |
| --- | --- | --- |
| Description | : | Retrieves the entry at position **givenPosition** in the list. |
| Precondition | : | **newPosition** must be between1 to total entries. |
| Postcondition | : | The list remains unchanged. |
| Returns | : | The entry at position **givenPosition**. |

**boolean isEmpty()**

|  |  |  |
| --- | --- | --- |
| Description | : | Determines whether the list is empty. |
| Postcondition | : | The list remains unchanged. |
| Returns | : | **true** if the list is empty, or false if not. |

**ADT Implementation**

Implementation Classes Source Code

1 /\*

2 \* To change this license header, choose License Headers in Project Properties.

3 \* To change this template file, choose Tools | Templates

4 \* and open the template in the editor.

5 \*/

6 package Bmodule;

7

8 /\*\*

9 \*

10 \* @author C K

11 \*/

12 public class List<T> implements listInterface<T> {

13

14 private Node firstNode; // reference to first node

15 private int staffCounter; // number of entries in list

16

17 public List() {

18 clear();

19 }

20

21 @Override

22 public final void clear() {

23 firstNode = null;

24 staffCounter = 0;

25 }

26

27 @Override

28 public void add(T newEntry) {

29 Node newNode= new Node(newEntry);

30

31 if(isEmpty())

32 {

33

34 firstNode=newNode;

35

36 }

37 else{

38

39 Node currentNode=firstNode;

40 while (currentNode.next != null) {

41 currentNode = currentNode.next;

42 }

43 currentNode.next = newNode;

44 }

45 staffCounter++;

46

47 }

48

49

50

51

52 @Override

53 public boolean update(int position, T newEntry) {

54 boolean isSuccessful = true;

55

56 if ((position >= 1) && (position <= staffCounter)) {

57 Node currentNode = firstNode;

58 for (int i = 0; i < position - 1; ++i) {

59

60 currentNode = currentNode.next;

61 }

62 currentNode.data = newEntry;

63 } else {

64 isSuccessful = false;

65 }

66

67 return isSuccessful;

68 }

69

70 @Override

71 public boolean isEmpty() {

72 boolean result;

73

74 result = staffCounter == 0;

75

76 return result;

77 }

78

79

80

81

82 public int size(){

83

84 return staffCounter;

85 }

86

87 @Override

88 public T getEntry(int position) {

89 T result = null;

90

91 if ((position >= 1) && (position <= staffCounter)) {

92 Node currentNode = firstNode;

93 for (int i = 0; i < position - 1; ++i) {

94 currentNode = currentNode.next; // advance currentNode to next node

95 }

96 result = currentNode.data; // currentNode is pointing to the node at givenPosition

97 }

98

99 return result;

100 }

101

102 public String toString() {

103 String outputStr = "";

104 Node currentNode = firstNode;

105 while (currentNode != null) {

106 outputStr += currentNode.data + "\n";

107 currentNode = currentNode.next;

108 }

109 return outputStr;

110 }

111

112

113

114

115

116 private class Node {

117

118 private T data;

119 private Node next;

120

121 private Node(T data) {

122 this.data = data;

123 this.next = null;

124 }

125

126 private Node(T data, Node next) {

127 this.data = data;

128 this.next = next;

129 }

130 }

131}

132

Interface Source Code

1 /\*

2 \* To change this license header, choose License Headers in Project Properties.

3 \* To change this template file, choose Tools | Templates

4 \* and open the template in the editor.

5 \*/

6 package Bmodule;

7

8 /\*\*

9 \*

10 \* @author C K

11 \*/

12public interface listInterface<T> {

13 public void add(T newEntry);

14 public boolean update(int position, T newEntry);

15 public boolean isEmpty();

16 public T getEntry(int position);

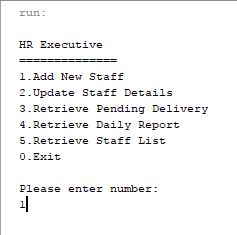
17 public void clear();

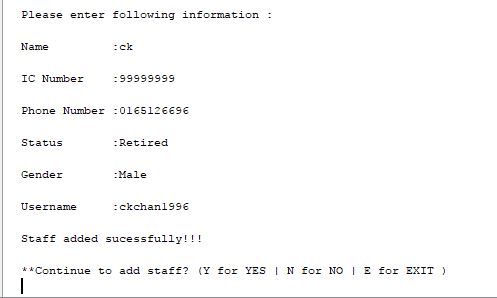
18 public int size();

19}

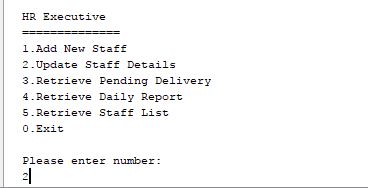
20

**Application Program**

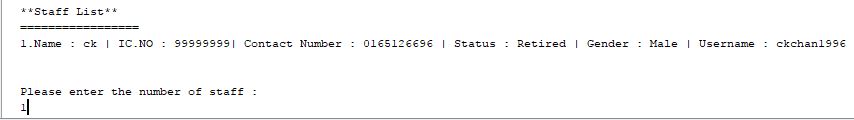
1. Add new staff.  
   In the main menu, enter 1 to perform this action.  
   

System will prompt the form to add staff.  


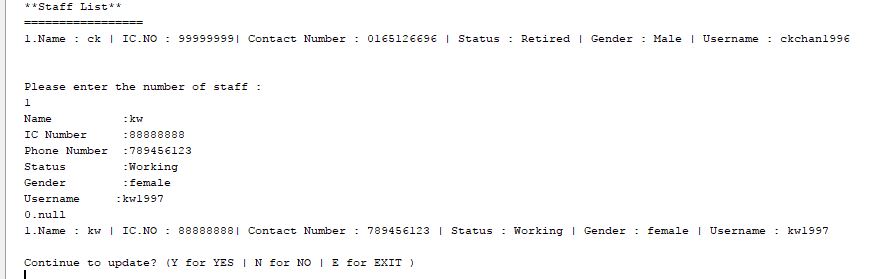
1. Update staff details.  
   In the main menu, enter 2 to perform this action.



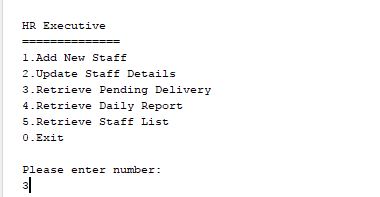
System will show the list of staff, enter the number of the staff to update



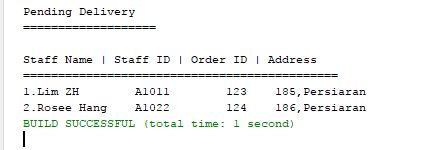
After entered the latest details of the staff, system will show the updated details of the staff



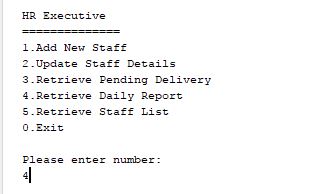
1. Retrieve pending report.  
   In the main menu, enter 3 to perform this action.



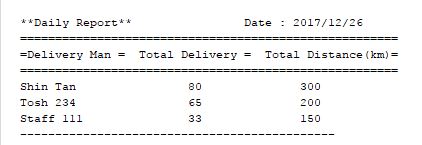
System will show the pending delivery



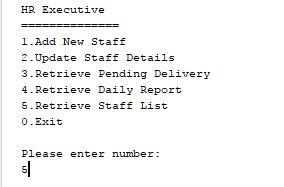
1. Retrieve daily report.  
   In the main menu, enter 4 to perform this action.



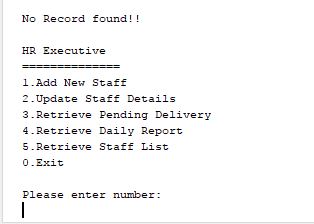
System will show the daily report.



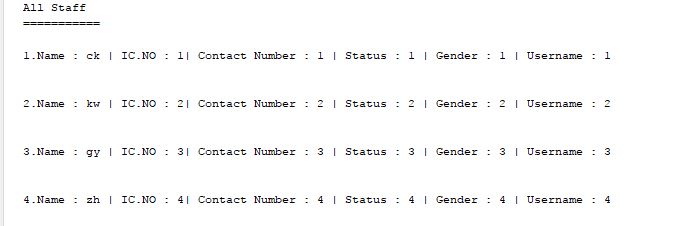
1. Show all staff.  
   In the main menu, enter 5 to perform this action.



If there is no record being entered, error message will show and direct user to menu.



After record have been entered, system will show the list of staff.



Module C

Part 1: ADT Specification

ADT Stack

Stack is an Abstract Data Type that uses 2 principal operations, push and pop to insert and remove its elements. The order in which the elements would go out follows the principal Last-In-Fist-Out.

OrderStack()

Description : Empty class constructor that can be invoked to create a stack

Post condition : Creates a stack with default size of 1

push(OrderData newEntry)

Description : Inserts a new OrderData type object into the stack

Post condition : A new OrderData type object is inserted on top of the stack

pop()

Description : Removes the top object found in the stack

Post condition : The object on top of the list is removed

peek()

Description : Obtains the value on top of the stack without altering the stack

Post condition : Returns the value of the object on top of the stack

removeSelected(int index)

Description : removes the selected item with the index number that is passed into

the method

Post condition : The item marked with the index in the stack is removed and the remaining

data is repositioned

isEmpty()

Description : Check whether the stack is empty

Post condition : Returns a Boolean value to indicate whether the stack is empty

clear()

Description : Removes all value from the stack and changes it to an empty stack

Post condition : The stack is changed to empty status

isArrayFull()

Description : Checks if the stack array has reached its upper limit

Post condition : Returns Boolean value to indicate if the array is full

doubleArray()

Description : Increases the size of the stack ADT

Post condition : The size of the stack ADT is multiplied by 2

Part 2: ADT Implementation

ADT Interface

1 package Module\_C;

2

3 public interface OrderStack\_Interface<OrderData> {

4 public void push(OrderData newEntry);

5 public OrderData pop();

6 public OrderData peek();

7 public boolean isEmpty();

8 public void clear();

9 }

ADT Class

001 package Module\_C;

002

003 public class OrderStack<OrderData> implements OrderStack\_Interface<OrderData> {

004 Node topNode;

005 int length = -1; //keeps track of size

006

007 //initializes link list

008 public OrderStack(){

009 topNode = null;

010 }

011

012 //Adds a new node on top of the existing top node

013 public void push(OrderData newEntry){

014 length++;

015 if (isEmpty()){

016 Node newNode = new Node(newEntry);

017 topNode = newNode;

018 topNode.next = null;

019 }

020 else {

021 Node newNode = new Node (newEntry, topNode);

022 newNode.next = topNode;

023 topNode = newNode;

024 }

025 }

026

027 //Removes top most node and set the node below it as top

028 public OrderData pop(){

029 if (isEmpty()){

030 return null;

031 }

032 else {

033 length--;

034 Node temp = topNode;

035 topNode = topNode.next;

036 return temp.data;

037 }

038 }

039

040 //Returns topmost node without modifying the list

041 public OrderData peek(){

042 if (isEmpty()){

043 return null;

044 }

045 else {

046 return topNode.data;

047 }

048 }

049

050 //Checks if the linked list is empty

051 public boolean isEmpty(){

052 if (topNode==null){

053 return true;

054 }

055 else {

056 return false;

057 }

058 }

059

060 //Clears the linked list

061 public void clear(){

062 topNode = null;

063 }

064

065 //Node object class

066 private class Node{

067 private OrderData data;

068 private Node next;

069

070 public Node(OrderData Data){

071 this.data = Data;

072 this.next = null;

073 }

074

075 public Node(OrderData Data, Node top){

076 this.data = Data;

077 this.next = top;

078 }

079 }

080 }

081

Part 3: Application Program

Description of functionalities

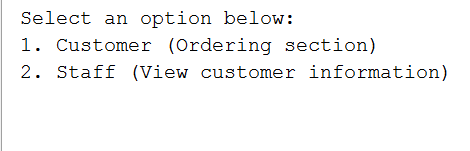


Figure 1: Module D Main Menu

In the figure above, the user may choose their operations, to make orders or proceed as a staff.

Entering 1 will move user to the next stage as shown in figure 2.

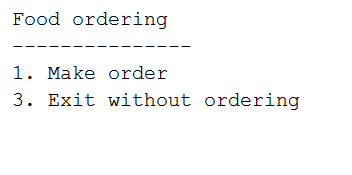


Figure 2: Customer ordering section

In this menu, the user may choose to make new order by entering 1 or leave by entering 3. Entering any other number would cause the user to loop in this menu

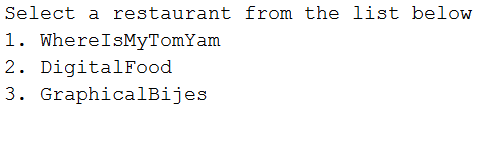


Figure 3: Restaurant selection

After selecting make order in the previous section, the user would be prompted by this menu to select one of the restaurants associated with FastestDeliveryman. The menu may display more restaurants if more restaurants are added in via Module A

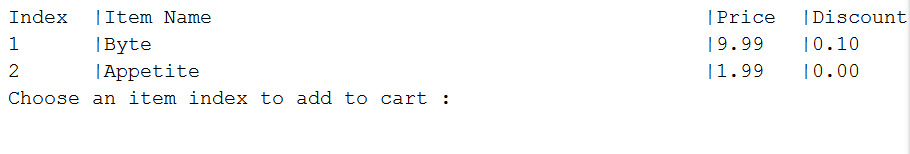


Figure 4: Items for sale from each restaurant

In figure 4, the menu shows the item available for sale after entering “2” when prompted in figure 3. This figure shows the sample data of what is available for sale by DigitalFood.

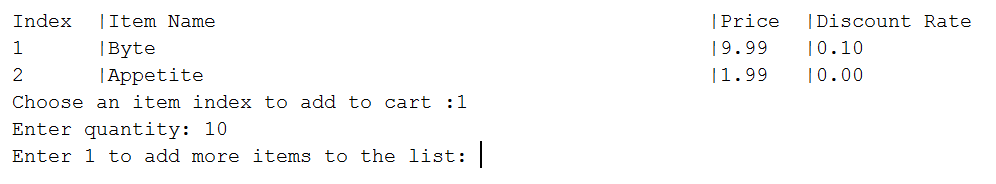


Figure 5: Selecting items and quantities

In figure 5, the user would be prompted to enter their desired item quantity after selecting the item

If the user enter “1” at the last prompt, they would be redirected to the add item screen at figure 3.

If the user enters any other number, they would be redirected to figure 6

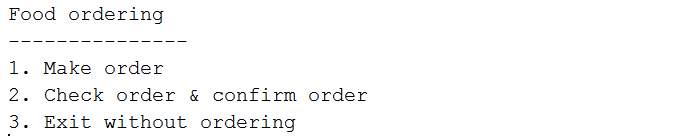


Figure 6: Check order and confirm order

Once at least one order has been made in this session, the option “2” will appear in the menu of figure 2 as shown in figure 6.

The user is required to enter this menu in order to confirm their order

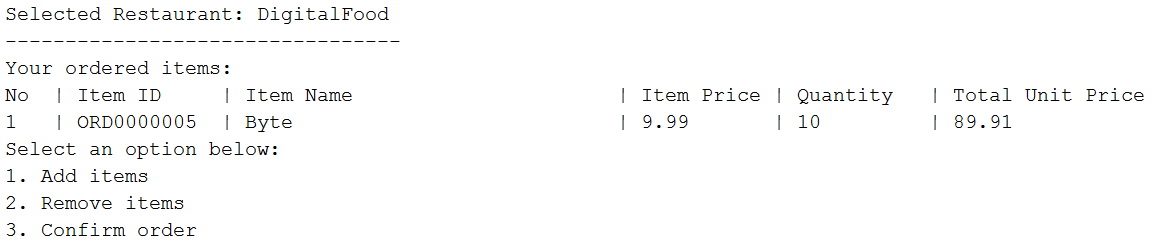


Figure 7: Order confirmation menu

In this menu shown in figure 7, the user can choose to add or remove items from their cart by entering “1” or “2” as values. Once the user enters “3” as a response, the order is considered finalised and the user will be prompted o enter their details as shown in Figure 8.

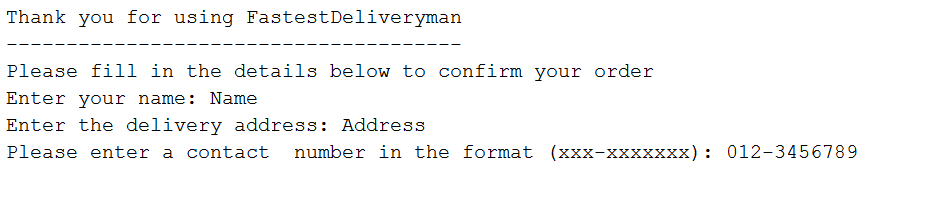


Figure 8: Customer Details

In this section, user is required to enter their details as prompted by the system. Once all information is entered, the order will be finalised and deliveries can be made

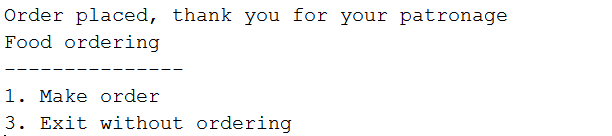


Figure 9: Completed Order

Once the user have completed the order process, they will be redirected to the same menu in Figure 2. The 2nd option will no longer be available until the user repeat the process in figure 3 - 5 again

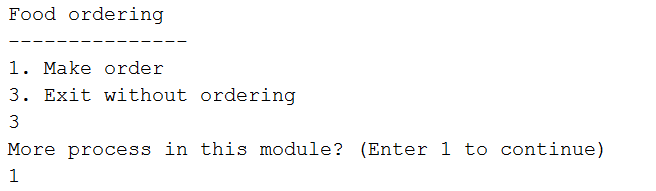


Figure 10: leaving customer menu

In order to leave the customer menu, the user would have to enter “3” as the input and then “1” in order to stay in Module C, entering other values will move the user to the system’s main menu

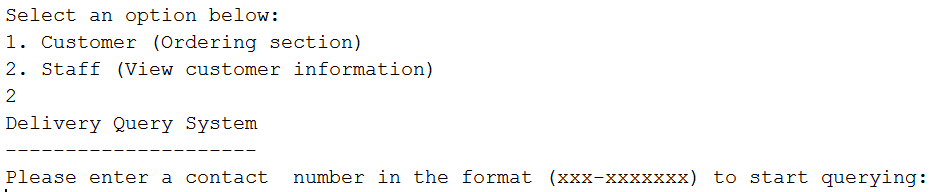


Figure 11: Staff Query

In figure 11, the user may check the delivery details of a customer by inserting their contact number

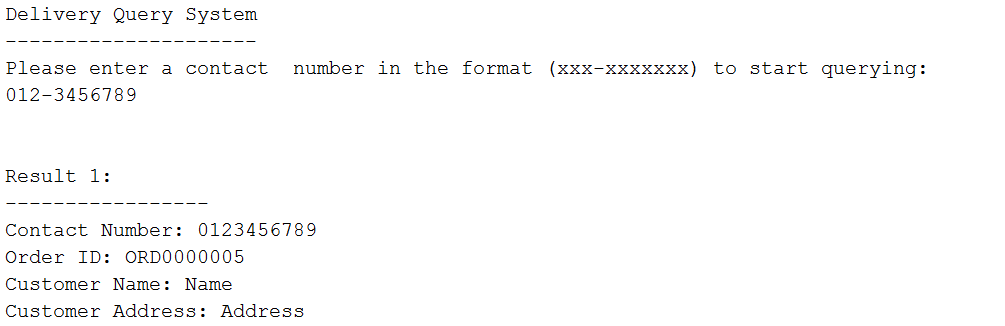


Figure 12: Contact with result

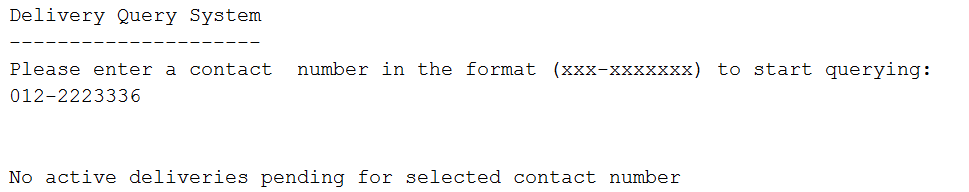


Figure 13: Contact with no result

In both figure 12 and 13, are the results shown if the contact number matches any **pending** delivery. Deliveries which are completed would not show up in this area

Entity class

MenuData Class

001 package Module\_C;

002

003 import java.sql.Connection;

004 import java.sql.DriverManager;

005 import java.sql.ResultSet;

006 import java.sql.SQLException;

007 import java.sql.Statement;

008 import java.sql.PreparedStatement;

009

010 public class MenuData {

011

012 private String ID;

013 private String name;

014 private double price;

015 private boolean availabilityStatus;

016

017 private String restaurantName;

018 private double discountRate;

019

020 //Empty constructor that sets values to default of empty

021 public MenuData(){

022 this.ID = "";

023 this.name = "";

024 this.price = 0.00;

025 this.availabilityStatus = false;

026 this.restaurantName = "";

027 this.discountRate = 0.00;

028 }

029

030 public MenuData(String ID, String name, double price, boolean status, String restaurantName, double discountRate){

031 this.ID = ID;

032 this.name = name;

033 this.price = price;

034 this.availabilityStatus = status;

035 this.restaurantName = restaurantName;

036 this.discountRate = discountRate;

037 }

038

039 //Information set methods

040 public void setID(String ID){

041 this.ID = ID;

042 }

043 public void setName(String name){

044 this.name = name;

045 }

046 public void setPrice(double price){

047 this.price = price;

048 }

049 public void setStatus(boolean status){

050 this.availabilityStatus = status;

051 }

052 public void setRestaurantName(String restaurantName){

053 this.restaurantName = restaurantName;

054 }

055 public void setDiscount(double discountRate){

056 this.discountRate = discountRate;

057 }

058

059 //Information getMethods

060 public String getID(){

061 return this.ID;

062 }

063 public String getName(){

064 return this.name;

065 }

066 public double getPrice(){

067 return this.price;

068 }

069 public boolean getStatus(){

070 return this.availabilityStatus;

071 }

072 public String getResturantName(){

073 return this.restaurantName;

074 }

075 public double getDiscount(){

076 return this.discountRate;

077 }

078

079 //Menu Database

080 public void addLocalDatabase(String ID, String name, double price, boolean status, String restaurantName, double discountRate){

081 try {

082 Connection conn = DriverManager.getConnection("jdbc:derby://localhost:1527/ModuleC", "ModuleC", "ModuleC");

083 PreparedStatement insert = conn.prepareStatement("Insert into Menu values (?, ?, ?, ?, ?, ?)");

084 insert.setString(1, ID);

085 insert.setString(2, name);

086 insert.setDouble(3, price);

087 insert.setBoolean(4, status);

088 insert.setString(5, restaurantName);

089 insert.setDouble(6, discountRate);

090 insert.execute();

091 }

092 catch (Exception ex){

093 System.out.println("Database connection failed");

094 }

095 }

096 public void deleteLocalDatabase(String ID){

097 try {

098 Connection conn = DriverManager.getConnection("jdbc:derby://localhost:1527/ModuleC", "ModuleC", "ModuleC");

099 PreparedStatement delete = conn.prepareStatement("delete from Menu where ID=?");

100 delete.setString(1, ID);

101 delete.execute();

102 }

103 catch (Exception ex){

104 System.out.println("Database connection failed");

105 }

106 }

107 public void updateLocalDatabase(String ID, String name, double price, boolean status, String restaurantName, double discountRate){

108 try {

109 Connection conn = DriverManager.getConnection("jdbc:derby://localhost:1527/ModuleC", "ModuleC", "ModuleC");

110 PreparedStatement insert = conn.prepareStatement("update Menu set ID=?, Name=?, Price=?, Status=?, RestaurantName=?, DiscountRate=?");

111 insert.setString(1, ID);

112 insert.setString(2, name);

113 insert.setDouble(3, price);

114 insert.setBoolean(4, status);

115 insert.setString(5, restaurantName);

116 insert.setDouble(6, discountRate);

117

118 insert.execute();

119 }

120 catch (Exception ex){

121 System.out.println("Database connection failed");

122 }

123 }

124

125 public String toString(){

126 return "ID: "+ID+" "+"Name: "+name+" "+"Price: "+price+" "+"Status: "+availabilityStatus+" "+"Restaurant Name: "+restaurantName+" "+"Discount Rate: "+discountRate+" ";

127 }

128 }

OrderData Class

001 package Module\_C;

002

003 import java.sql.Connection;

004 import java.sql.DriverManager;

005 import java.sql.ResultSet;

006 import java.sql.SQLException;

007 import java.sql.Statement;

008 import java.sql.PreparedStatement;

009

010 public class OrderData {

011

012 private String orderID;

013 private String itemID;

014 private String itemName;

015 private double unitPrice;

016 private int quantity;

017 private double totalUnitPrice;

018 private String restaurantName;

019 double discount;

020

021 public OrderData(){

022 this.orderID = null;

023 this.itemID = null;

024 this.itemName = null;

025 this.unitPrice = 0;

026 this.quantity = 0;

027 this.totalUnitPrice = 0;

028 this.restaurantName = null;

029 this.discount = 0;

030 }

031

032 public OrderData(String orderID, String itemID, String itemName, double unitPrice, int quantity, String restaurantName, double discount){

033 this.orderID = orderID;

034 this.itemID = itemID;

035 this.itemName = itemName;

036 this.unitPrice = unitPrice;

037 this.quantity = quantity;

038 this.totalUnitPrice = calculateUnitTotal(unitPrice, quantity, discount);

039 this.restaurantName = restaurantName;

040 this.discount = discount;

041 }

042

043 public OrderData(String itemID, String itemName, double unitPrice, int quantity, String restaurantName, double discount){

044 this.orderID = generateOrderID();

045 this.itemID = itemID;

046 this.itemName = itemName;

047 this.unitPrice = unitPrice;

048 this.quantity = quantity;

049 this.totalUnitPrice = calculateUnitTotal(unitPrice, quantity, discount);

050 this.restaurantName = restaurantName;

051 this.discount = discount;

052 }

053

054 //Data methods

055 public void setOrderID(String orderID){

056 this.orderID = generateOrderID();

057 }

058 public void setItemID(String itemID){

059 this.itemID = itemID;

060 }

061 public void setUnitPrice(double unitPrice){

062 this.unitPrice = unitPrice;

063 }

064 public void setQuantity(int quantity){

065 this.quantity = quantity;

066 }

067 public void setTotalPrice(){

068 this.totalUnitPrice = unitPrice \* quantity;

069 }

070 public void setRestaurantName(String restaurantName){

071 this.restaurantName = restaurantName;

072 }

073 public void setDiscount(double Discount){

074 this.discount = discount;

075 }

076 public void setName(String itemName){

077 this.itemName = itemName;

078 }

079

080 public String getOrderID(){

081 return this.orderID;

082 }

083 public String getItemID(){

084 return this.itemID;

085 }

086 public double getUnitPrice(){

087 return this.unitPrice;

088 }

089 public int getQuantity(){

090 return this.quantity;

091 }

092 public double getTotalUnitPrice(){

093 return this.totalUnitPrice;

094 }

095 public String getRestaurantName(){

096 return this.restaurantName;

097 }

098 public double getDiscount(){

099 return this.discount;

100 }

101 public String getItemName(){

102 return this.itemName;

103 }

104

105 //Supportive methods

106 public double calculateUnitTotal(double price, int quantity, double discount){

107 return ((price \* quantity) \* (1.00-discount));

108 }

109 public String generateOrderID(){

110 String orderID = "";

111 try {

112 Connection conn = DriverManager.getConnection("jdbc:derby://localhost:1527/ModuleC", "ModuleC", "ModuleC");

113 PreparedStatement pstmtRestaurant = conn.prepareStatement("Select orderid from ordering group by orderid");

114 ResultSet resultCount = pstmtRestaurant.executeQuery();

115 int count = 1;

116 for (;resultCount.next();count++){

117

118 }

119 orderID = orderIDSerialGenerator(count, 1, 6);

120 }

121 catch (SQLException ex){

122 System.out.println("SQL Exception has occured");

123 }

124 return orderID;

125 }

126 public String orderIDSerialGenerator(int orderIndex, int initial, int zero){

127 initial = initial\*10;

128 if(orderIndex < initial){

129

130 }

131 else {

132 orderIDSerialGenerator(orderIndex, initial, zero-1);

133 }

134 return "ORD"+zeroString(zero)+orderIndex;

135 }

136 public String zeroString(int number){

137 String zero = "";

138 for (int counter = 0; counter < number; counter++){

139 zero = zero + "0";

140 }

141 return zero;

142 }

143

144 //Database Ordering Table insert

145 public void insertNewOrder(OrderStack stack){

146 try {

147 OrderData[] order = new OrderData[stack.length+1];

148 int stackLength = stack.length+1;

149 Connection conn = DriverManager.getConnection("jdbc:derby://localhost:1527/ModuleC", "ModuleC", "ModuleC");

150

151 for (int count = 0; count < stackLength; count++){

152 order[count] = (OrderData)stack.pop();

153 PreparedStatement pstmtInsert = conn.prepareStatement("insert into ordering values ( ?, ?, ?, ?, ?, ?)");

154 pstmtInsert.setString(1, order[count].getOrderID());

155 pstmtInsert.setString(2, order[count].getItemID());

156 pstmtInsert.setDouble(3, order[count].getUnitPrice());

157 pstmtInsert.setInt(4, order[count].getQuantity());

158 pstmtInsert.setDouble(5, order[count].getTotalUnitPrice());

159 pstmtInsert.setString(6, order[count].getRestaurantName());

160

161 pstmtInsert.execute();

162 }

163 }

164 catch (SQLException ex){

165 System.err.println(ex);

166 }

167

168 }

169 }

CustomerDeliveryInfo Class

001 package Module\_C;

002

003 import java.util.Scanner;

004 import java.sql.\*;

005

006 public class CustomerDeliveryInfo {

007 private String name;

008 private String orderID;

009 private String address;

010 private String contactNumber;

011 private boolean delivered;

012

013 public CustomerDeliveryInfo(){

014 name = "";

015 orderID = "";

016 address = "";

017 contactNumber = "";

018 delivered = false;

019 }

020 public CustomerDeliveryInfo(String name, String orderID, String address, String contactNumber, boolean deliveryStatus){

021 this.name = name;

022 this.orderID = orderID;

023 this.address = address;

024 this.contactNumber = contactNumber;

025 this.delivered = deliveryStatus;

026 }

027

028 //information method

029 public String getName() {

030 return name;

031 }

032 public String getOrderID() {

033 return orderID;

034 }

035 public String getAddress() {

036 return address;

037 }

038 public String getContactNumber() {

039 return contactNumber;

040 }

041 public boolean isDelivered() {

042 return delivered;

043 }

044

045 public void setName(String name) {

046 this.name = name;

047 }

048 public void setOrderID(String orderID) {

049 this.orderID = orderID;

050 }

051 public void setAddress(String address) {

052 this.address = address;

053 }

054 public void setContactNumber(String contactNumber) {

055 this.contactNumber = contactNumber;

056 }

057 public void setDelivered(boolean delivered) {

058 this.delivered = delivered;

059 }

060

061 public void addIntoLocalDatabase(){

062 try {

063 Connection conn = DriverManager.getConnection("jdbc:derby://localhost:1527/ModuleC", "ModuleC", "ModuleC");

064 PreparedStatement insert = conn.prepareStatement("Insert into customer\_delivery values (?,?,?,?,?)");

065 insert.setString(1, name);

066 insert.setString(2, address);

067 insert.setString(3, contactNumber);

068 insert.setBoolean(4, delivered);

069 insert.setString(5, orderID);

070

071 insert.execute();

072 }

073 catch (Exception ex){

074 System.err.println(ex);

075 }

076 }

077 }

Client Application Program Class

CustomerInfoRetrieval Class

001 package Module\_C;

002

003 import java.util.Scanner;

004

005 import java.sql.Connection;

006 import java.sql.DriverManager;

007 import java.sql.ResultSet;

008 import java.sql.SQLException;

009 import java.sql.Statement;

010

011 import java.sql.PreparedStatement;

012

013 public class CustomerInfoRetrieval {

014 Scanner scanner = new Scanner(System.in);

015

016 public CustomerInfoRetrieval(){

017 try {

018 int redo;

019

020 do {

021 redo = 0;

022 Connection conn = DriverManager.getConnection("jdbc:derby://localhost:1527/ModuleC", "ModuleC", "ModuleC");

023 String contactNumber;

024

025 boolean invalid = false;

026 do{

027 invalid = false;

028 System.out.println("Delivery Query System");

029 System.out.println("---------------------");

030 System.out.println("Please enter a contact number in the format (xxx-xxxxxxx) to start querying: ");

031 contactNumber = scanner.nextLine();

032 if (contactNumber.length() != 11){

033 invalid = true;

034 System.out.println("\n");

035 System.out.println("\n");

036 System.out.println("\n");

037 System.out.println("Invalid contact number, try again\n");

038 }

039 }while (invalid == true);

040

041 String firstThree = contactNumber.substring(0, 3);

042 String lastSegment = contactNumber.substring(4);

043

044 PreparedStatement statementQuery = conn.prepareStatement("Select \* from customer\_delivery where contactnumber = ? and delivered = false");

045 statementQuery.setString(1, firstThree.concat(lastSegment));

046 System.out.println("\n");

047

048 ResultSet results = statementQuery.executeQuery();

049

050 if (results.next() == false){

051 System.out.println("No active deliveries pending for selected contact number"); //Retieve customer details based on contact number so that i can proceed with my work

052 }

053 else {

054 System.out.println("Result 1:");

055 System.out.println("-----------------");

056 System.out.println("Contact Number: "+results.getString("Customer\_ContactNumber"));

057 System.out.println("Order ID: "+results.getString("Order\_ID"));

058 System.out.println("Customer Name: "+results.getString("Customer\_Name"));

059 System.out.println("Customer Address: "+results.getString("Customer\_DELIVERY\_Address"));

060 System.out.println("Enter any value to proceed");

061 String valuePlaceholder = scanner.nextLine();

062

063 for (int index = 2;results.next()==true; index++){

064 System.out.println("\n");

065 System.out.println("\n");

066 System.out.println("Result "+index+":");

067 System.out.println("-----------------");

068 System.out.println("Contact Number: "+results.getString("ContactNumber"));

069 System.out.println("Order ID: "+results.getString("Order\_ID"));

070 System.out.println("Customer Name: "+results.getString("Name"));

071 System.out.println("Customer Address: "+results.getString("Address"));

072 System.out.println("Enter any value to proceed");

073 valuePlaceholder = scanner.nextLine();

074 }

075 }

076 int errorFree;

077 do {

078 errorFree = 1;

079 System.out.println("Is there more queries?: (Enter 1 for yes)");

080 String input = scanner.nextLine();

081 System.out.println("\n");

082 System.out.println("\n");

083 try {

084 redo = Integer.parseInt(input);

085 }

086 catch (Exception ex){

087 errorFree = 0;

088 System.out.println("Only integers are allowed");

089 }

090 } while (errorFree == 0);

091

092 }while(redo == 1);

093 }

094 catch (Exception ex){

095 ex.printStackTrace();

096 }

097

098 }

099 }

Menu\_Ordering Class

001 package Module\_C;

002

003 import java.util.Scanner;

004

005 import java.sql.Connection;

006 import java.sql.DriverManager;

007 import java.sql.ResultSet;

008 import java.sql.SQLException;

009 import java.sql.Statement;

010 import java.sql.PreparedStatement;

011

012 public class Menu\_Ordering {

013 OrderStack order = new OrderStack();

014 OrderStack stack;

015 Scanner scanner = new Scanner(System.in);

016

017 public Menu\_Ordering(){

018

019 }

020

021 public void classMenu(){

022 int orderMore;

023 int option;

024 String newLineEater;

025

026 do {

027 orderMore = 0;

028 do {

029 System.out.println("Food ordering");

030 System.out.println("---------------");

031 System.out.println("1. Make order");

032 if (order.length != -1){

033 System.out.println("2. Check order & confirm order");

034 }

035 System.out.println("3. Exit without ordering");

036 option = scanner.nextInt();

037 newLineEater = scanner.nextLine();

038

039 if (option == 1){

040 displayRestaurant\_Order();

041 }

042 if (order.length != -1){

043 if (option == 2){

044 displayItems();

045 }

046 }

047

048 }while (option != 3);

049

050

051 }while (orderMore == 1);

052

053 }

054 public void displayRestaurant\_Order(){

055 String orderID;

056 OrderData initiate = new OrderData();

057 orderID = initiate.generateOrderID();

058

059 order(orderID);

060 }

061 public void order(String orderID){

062 int restaurantNumber; //Stores restaurant selected by user

063 String newLineEater; //Eats new lines for breakfast

064

065 try {

066 int restaurantCount = 1;

067 Connection conn = DriverManager.getConnection("jdbc:derby://localhost:1527/ModuleC", "ModuleC", "ModuleC");

068 PreparedStatement pstmtRestaurant = conn.prepareStatement("Select RestaurantName from Menu group by restaurantName");

069 ResultSet resultCount = pstmtRestaurant.executeQuery();

070

071 if (resultCount.next()== true){

072 for (;resultCount.next();restaurantCount++){

073

074 }

075 }

076 //Displaying dynamic restaurant name

077 ResultSet resultSearch = pstmtRestaurant.executeQuery();

078 String[] availableRestaurants = new String[restaurantCount];

079 int index;

080 for(index = 0;resultSearch.next() == true;index++){

081 availableRestaurants[index] = resultSearch.getString("RestaurantName");

082 }

083 System.out.println("Select a restaurant from the list below");

084 for(int counter = 0; counter < index; counter++){

085 System.out.println((counter+1)+". "+availableRestaurants[counter]);

086 }

087 restaurantNumber = scanner.nextInt();

088 newLineEater = scanner.nextLine();

089 restaurantNumber = restaurantNumber - 1;

090

091 //Displaying dynamic item data

092 PreparedStatement pstmtRestaurantMenu = conn.prepareStatement("Select \* from menu where restaurantName = ? and status = 'true'");

093 pstmtRestaurantMenu.setString(1, availableRestaurants[restaurantNumber]);

094 ResultSet MenuSearchResult = pstmtRestaurantMenu.executeQuery();

095 int itemCount = 1;

096 if (MenuSearchResult.next()== true){

097 for (;MenuSearchResult.next();itemCount++){

098

099 }

100 }

101 MenuData[] data = new MenuData[itemCount];

102 MenuSearchResult = pstmtRestaurantMenu.executeQuery();

103

104

105 for(index = 0; MenuSearchResult.next()==true;index++){

106 data[index] = new MenuData(MenuSearchResult.getString("ID"), MenuSearchResult.getString("Name"), MenuSearchResult.getDouble("Price"), MenuSearchResult.getBoolean("Status"), MenuSearchResult.getString("RestaurantName"),MenuSearchResult.getDouble("Discount"));

107 }

108

109 int more = 0;

110 do {

111 int itemToAdd;

112 System.out.println("----------------------------------------------------------------------");

113 System.out.printf("%-7s|%-50s|%-7s|%-15s\n", "Index", "Item Name", "Price", "Discount Rate");

114 for (int counter = 0; counter < index; counter++){

115 System.out.printf("%-7d|%-50s|%-7.2f|%-15.2f\n", counter+1, data[counter].getName(), data[counter].getPrice(), data[counter].getDiscount());

116 }

117

118 //data[counter].getName(), data[counter].getPrice(), data[counter].getDiscount()

119 System.out.print("Choose an item index to add to cart :");

120 itemToAdd = scanner.nextInt();

121 newLineEater = scanner.nextLine();

122

123 System.out.print("Enter quantity: ");

124 int quantity = scanner.nextInt();

125 newLineEater = scanner.nextLine();

126

127 //

128

129 boolean condition = false; //stores value of whether item matches in check

130 stack = new OrderStack();

131 try {

132 //If true, means order stack has at least 1 record

133 int length = order.length+1;

134 if (length > 0){

135 OrderData[] record = new OrderData[length];

136 for (int start = 0; start < length; start++){

137 record[start] = (OrderData)order.pop();

138 if (record[start].getItemID().equals(data[itemToAdd-1].getID())){

139 int newQuantity = record[start].getQuantity() + quantity;

140 record[start].setQuantity(newQuantity);

141

142 condition = true;

143 }

144 else {

145 condition = false;

146 }

147 stack.push(record[start]);

148 }

149 order = stack;

150 }

151 if (condition != true)

152 {

153 OrderData entry = new OrderData(orderID, data[itemToAdd-1].getID(), data[itemToAdd-1].getName(), data[itemToAdd-1].getPrice(), quantity, data[itemToAdd-1].getResturantName(), data[itemToAdd-1].getDiscount());

154 order.push(entry);

155 }

156

157 }

158 catch(Exception ex){

159 System.err.println(ex);

160 }

161 System.out.print("Enter 1 to add more items to the list: ");

162 more = scanner.nextInt();

163 newLineEater = scanner.nextLine();

164 }while (more == 1);

165 }

166 catch (SQLException ex){

167 System.err.println("SQL Exception");

168 }

169 }

170 public void displayItems(){

171 //Manipulation

172 int option = 0; //Stores direct user input

173 String newLineEater; //Removes excess newline

174

175 //Generate a duplicate stack for editing

176 int ItemCount;

177 OrderStack temp = new OrderStack();

178

179 //Peek for restaurant name

180 OrderData peekResult = (OrderData)order.peek();

181 String restaurantName = peekResult.getRestaurantName();

182

183 do{

184 ItemCount = order.length+1;

185 if (order.length != -1){

186 OrderData[] data = new OrderData[order.length+1];

187

188 System.out.println("Selected Restaurant: "+restaurantName);

189 System.out.println("---------------------------------");

190 System.out.println("Your ordered items: ");

191 System.out.printf("%-3s | %-11s | %-30s | %-10s | %-10s | %-10s\n", "No","Item ID", "Item Name", "Item Price", "Quantity", "Total Unit Price");

192 for (int index = 0; index < ItemCount; index++){

193 data[index] = (OrderData)order.pop();

194 temp.push((OrderData)data[index]);

195 System.out.printf("%-3d | %-11s | %-30s | %-10.2f | %-10d | %-10.2f\n", index+1, data[index].getOrderID(), data[index].getItemName(), data[index].getUnitPrice(), data[index].getQuantity(), data[index].getTotalUnitPrice());

196 }

197 order = temp;

198 System.out.println("Select an option below: ");

199 System.out.println("1. Add items");

200 System.out.println("2. Remove items");

201 System.out.println("3. Confirm order");

202 option = scanner.nextInt();

203 newLineEater = scanner.nextLine();

204 if (option == 1){

205 order(peekResult.getOrderID());

206 }

207 else if (option == 2){

208 System.out.print("Select an item to remove: ");

209 int index = scanner.nextInt();

210

211 stack = new OrderStack();

212 try {

213 //If true, means order stack has at least 1 record

214 int length = order.length+1;

215 if (length > 0){

216 OrderData[] record = new OrderData[length];

217 for (int start = 0; start < length; start++){

218 record[start] = (OrderData)order.pop();

219 if (record[start].getItemID().equals(data[index-1].getItemID())){

220

221 }

222 else {

223 stack.push(record[start]);

224 }

225 }

226 order = stack;

227 }

228 }

229 catch (Exception ex){

230 System.out.println("Error");

231 }

232

233 newLineEater = scanner.nextLine();

234 }

235 else if (option == 3){

236 //Insert customer info

237 customerInformation(data[0].getOrderID());

238

239 OrderData updateDB = new OrderData();

240 order = temp;

241 updateDB.insertNewOrder(order);

242

243 System.out.println();

244 System.out.println();

245 System.out.println();

246 System.out.println("Order placed, thank you for your patronage");

247 }

248 }

249 }while (option != 3);

250 }

251

252 public void customerInformation(String orderID){

253 CustomerDeliveryInfo info = new CustomerDeliveryInfo();

254 System.out.println("Thank you for using FastestDeliveryman");

255 System.out.println("--------------------------------------");

256 System.out.println("Please fill in the details below to confirm your order");

257 System.out.print("Enter your name: ");

258 info.setName(scanner.nextLine());

259

260 System.out.print("Enter the delivery address: ");

261 info.setAddress(scanner.nextLine());

262

263 String contactNumber;

264 boolean invalid;

265 do{

266 invalid = false;

267 System.out.print("Please enter a contact number in the format (xxx-xxxxxxx): ");

268 contactNumber = scanner.nextLine();

269 if (contactNumber.length() != 11){

270 invalid = true;

271 System.err.println("Invalid contact number, try again\n");

272 }

273 }while (invalid == true);

274 String firstThree = contactNumber.substring(0, 3);

275 String lastSegment = contactNumber.substring(4);

276 info.setContactNumber(firstThree.concat(lastSegment));

277

278 info.setOrderID(orderID);

279 info.setDelivered(false);

280

281 info.addIntoLocalDatabase();

282 }

283 }

ModuleC\_MainMenu class

001 /\*

002 \* To change this license header, choose License Headers in Project Properties.

003 \* To change this template file, choose Tools | Templates

004 \* and open the template in the editor.

005 \*/

006 package Module\_C;

007

008 import java.util.Scanner;

009 /\*\*

010 \*

011 \* @author lenovo

012 \*/

013 public class ModuleC\_MainMenu {

014 Scanner scanner = new Scanner(System.in);

015 Menu\_Ordering ordering = new Menu\_Ordering();

016

017 public ModuleC\_MainMenu(){

018 int redo = 0;

019 do {

020 System.out.println("Select an option below: ");

021 System.out.println("1. Customer (Ordering section)");

022 System.out.println("2. Staff (View customer information)");

023 int option = scanner.nextInt();

024 String newLineEater = scanner.nextLine();

025 if (option == 1){

026 ordering.classMenu();

027 }

028 else if (option == 2){

029 CustomerInfoRetrieval info = new CustomerInfoRetrieval();

030 }

031

032 System.out.println("More process in this module? (Enter 1 to continue)");

033 redo = scanner.nextInt();

034 }while (redo == 1);

035

036 }

037 }

Module D

Part 1

ADT List

A list is a linear collection of entries of a type T which allows duplicate elements. An entry may be added at a specified position or at the end of the list.

**addDeliveryMan(T newEntry)**

Description : Adds newEntry to the end of the list.

Postcondition : newEntry has been added to the end of the list.

**T listDeliveryMan()**

Description : Removes the entry within the list.

Postcondition : The entry has been removed from the list.

**T getLatest()**

Description : Take a look of the end of the list.

Postcondition : The entry of the end of the list has been seen.

**int getLength()**

Description : Get the length of the list.

Postcondition : The list remains unchanged.

Returns : return the value of the length.

**public T getEntry(int givenPosition)**

Description : Retrieves the entry at position givenPosition in the list.

Precondition : newPosition must be between1 to total entries.

Postcondition : The list remains unchanged.

Returns : The entry at position givenPosition.

**boolean isEmpty()**

Description : Determines whether the list is empty.

Postcondition : The list remains unchanged.

Returns : true if the list is empty, or false if not.

**clear()**

Description : Removes all entries from the list.

Postcondition : The list is now empty.

Part 2

Interface class

1 package ModuleD;

2

3

4 public interface QueueInterface<T> {

5

6 public void addDeliveryMan(T newEntry);

7

8 public T listDeliveryMan();

9

10 public T getLatest();

11

12 public int getLength();

13

14 public T getEntry(int givenPosition);

15

16 public boolean isEmpty();

17

18 public void clear();

Implementation class

1 package ModuleD;

2

3

4 public class LinkedQueue<T> implements QueueInterface<T> {

5

6 private Node lastNode;

7 private int length;

8

9 public LinkedQueue() {

10 lastNode = null;

11 length = 0;

12 }

13

14 @Override

15 public void addDeliveryMan(T newElement) {/\*add deliveryman to the list\*/

16 if (isEmpty()) {

17 Node newNode = new Node(newElement);

18 lastNode = newNode;

19 lastNode.next = newNode;

20 lastNode.previous = newNode;

21

22 } else {

23 Node newNode = new Node(newElement, lastNode.next, lastNode);

24 lastNode.next = newNode;

25 lastNode = newNode;

26 lastNode.next.previous = lastNode;

27 }

28 length++;

29 }

30

31 public T listDeliveryMan() {/\*remove deliveryman from the list\*/

32 if (isEmpty()) {

33 return null;

34 } else if (lastNode.next == lastNode) {

35 T temp = lastNode.next.data;

36 lastNode = null;

37 return temp;

38 } else {

39 T temp = lastNode.next.data;

40 lastNode.next = lastNode.next.next;

41 lastNode.next.previous = lastNode;

42 return temp;

43 }

44

45 }

46

47 @Override

48 public T getLatest() {/\*get the lastest deliveryman on the list\*/

49 if (isEmpty()) {

50 return null;

51 } else {

52 return lastNode.next.data;

53 }

54 }

55

56 public int getLength() {//get the length of the list

57 return length;

58 }

59

60 public T getEntry(int givenPosition) {//get the specific position of entry in the list

61 T result = null;

62

63 if ((givenPosition >= 1) && (givenPosition <= length)) {

64 //currentNode to the firstNode

65 Node currentNode = lastNode;

66 for (int i = 0; i < givenPosition - 1; ++i) {

67 //advance currentNode to next node

68 currentNode = currentNode.next;

69 }

70 //current is pointing to the Node at givenPosition

71 result = currentNode.data;

72 }

73 return result;

74 }

75

76 @Override

77 public boolean isEmpty() {/\*check list isn't empty\*/

78 return lastNode == null;

79 }

80

81 @Override

82 public void clear() {/\*clear list\*/

83 lastNode = null;

84 }

85

86 private class Node {

87

88 private T data;

89 private Node next;

90 private Node previous;

91

92 public Node(T data) {

93 this.data = data;

94 this.next = null;

95 this.previous = null;

96 }

97

98 public Node(T data, Node next, Node previous) {

99 this.data = data;

100 this.next = next;

101 this.previous = previous;

102 }

103

104 public T getData() {

105 return this.data;

106 }

107

108 public Node getNext() {

109 return this.previous;

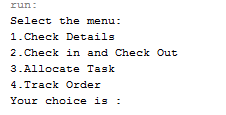
110 }

111 }

Part 3

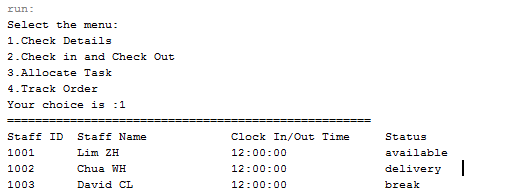
1. Main menu

This is where user select the function in the menu.



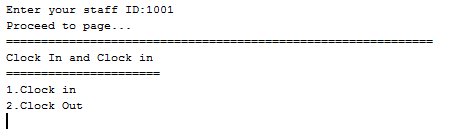
2. Check Details

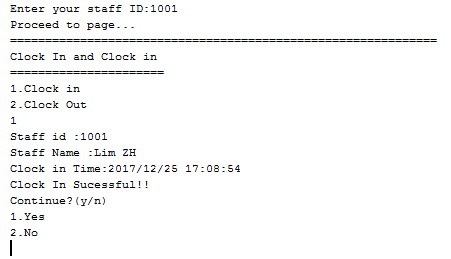
This function is to let manager to keep track of deliveryman status such as, available, delivery, break, and the details is show as a report form.



3. Clock In and Out

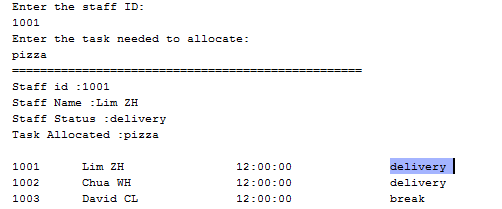
This function is to let deliveryman clock in and clock out during the working hours. First, a message is prompt to user to enter his/her staff ID and a menu display to let user to choose clock in and clock out.



Then, a messages will show the details of the check in or out and the time is based on real-time local time. A messages will ask the user whether they need to enter again.

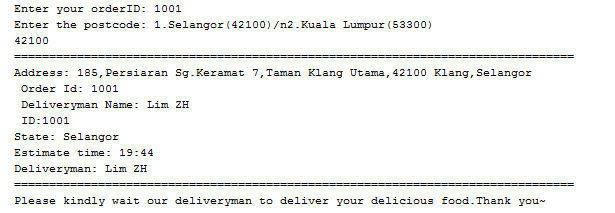
4. Allocate Task

This function is to allocate deliveryman a task and change his/her status into current situation. As the diagram below show, a messages is prompt to let user to enter staff ID to proceed to enter the content of the task. The module will update the status of the deliveryman status on the same time as showed as highlight part of the diagram below and display all the deliveryman in report form.



5. Track Order

This function is to let customer to track his/her order. This also will let customer to know the remaining time of the order arrive and the details of order. A messages will prompt to let enter order ID, postcode to verify the order and details of order and remaining time is displayed as shown as diagram below.



Entity class

1 package ModuleD;

2

3 public class Delivery {

4

5 private int staffID;

6 private String staff\_name;

7 private String break\_time;

8 private String available;

9

10 public Delivery(int staffID, String staff\_name, String break\_time, String available) {

11 this.staffID = staffID;

12 this.staff\_name = staff\_name;

13 this.break\_time = break\_time;

14 this.available = available;

15 }

16

17 public String getAvailable() {

18 return available;

19 }

20

21 public int getStaffID() {

22 return staffID;

23 }

24

25 public String getStaff\_name() {

26 return staff\_name;

27 }

28

29 public String getBreak\_time() {

30 return break\_time;

31 }

32

33 public void setAvailable(String available) {

34 this.available = available;

35 }

36

37 public void setStaffID(int staffID) {

38 this.staffID = staffID;

39 }

40

41 public void setStaff\_name(String staff\_name) {

42 this.staff\_name = staff\_name;

43 }

44

45 public void setBreak\_time(String break\_time) {

46 this.break\_time = break\_time;

47 }

48

49 @Override

50 public String toString() {

51 return String.format("%-9d %-21s %-21s %-21s" ,staffID ,staff\_name ,break\_time,available);

52 }

53

54}

Client application program class

1 package ModuleD;

2

3 import java.awt.Label;

4 import java.text.DateFormat;

5 import java.text.SimpleDateFormat;

6 import java.time.LocalTime;

7 import java.time.format.DateTimeFormatter;

8 import java.util.\*;

9

10 /\*\*

11 \*

12 \* @author Lim Zhi Hui 17WMR12382

13 \*/

14 public class ModuleD {

15 public String order;

16 private QueueInterface<Delivery> deliverylist = new LinkedQueue<>();

17 Delivery deliveryman1 = new Delivery(1001, "Lim ZH", "12:00:00", "available");

18 Delivery deliveryman2 = new Delivery(1002, "Chua WH", "12:00:00", "delivery");

19 Delivery deliveryman3 = new Delivery(1003, "David CL", "12:00:00", "break");

20

21 public static void main(String args[]) {

22 ModuleD module = new ModuleD();

23 module.mainmenu();

24 }

25

26 public void mainmenu() {

27 Scanner scanner = new Scanner(System.in);

28 System.out.println("Select the menu: ");

29 System.out.println("1.Check Details");

30 System.out.println("2.Check in and Check Out");

31 System.out.println("3.Allocate Task");

32 System.out.println("4.Track Order");

33 System.out.printf("Your choice is :");

34 int choice = scanner.nextInt();

35 if (choice == 1) {

36 CheckDetails();

37 } else if (choice == 2) {

38 checkinout();

39 } else if (choice == 3) {

40 AllocateTask();

41 } else if (choice == 4) {

42 RemainingTime();

43 }

44 }

45

46 public void AllocateTask() {

47 Scanner scanner = new Scanner(System.in);

48 deliverylist.addDeliveryMan(deliveryman1);

49 deliverylist.addDeliveryMan(deliveryman2);

50 deliverylist.addDeliveryMan(deliveryman3);

51 String available = "available";

52 String delivery = "delivery";

53 String breaktime = "break";

54 System.out.println("Enter the staff ID: ");

55 int staffid = scanner.nextInt();

56 for (int a = 0; a > deliverylist.getLength(); a++) {

57 if (deliverylist.getEntry(a).getAvailable() == available) {

58 deliverylist.getEntry(a).setAvailable("delivery");

59 } else if (deliverylist.getEntry(a).getAvailable() == delivery) {

60 System.out.println("Current deliveryman is in delivery task.");

61

62 } else if (deliverylist.getEntry(a).getAvailable() == breaktime) {

63 System.out.println("Current deliveryman is on break.");

64 }

65

66 }

67

68 if (staffid == 1001) {

69 System.out.println("Enter the task needed to allocate:");

70 String task = scanner.next();

71 deliveryman1.setAvailable("delivery");

72 Delivery delivery2 = new Delivery(deliveryman1.getStaffID(), deliveryman1.getStaff\_name(), deliveryman1.getBreak\_time(), deliveryman1.getAvailable());

73 System.out.println("==================================================");

74 System.out.println("Staff id :" + delivery2.getStaffID());

75 System.out.println("Staff Name :" + delivery2.getStaff\_name());

76 System.out.println("Staff Status :" + delivery2.getAvailable());

77 System.out.println("Task Allocated :"+task);

78 System.out.println("");

79 } else if (staffid == 1002) {

80 System.out.println("Cuurent Deliveryman is cuurently on task. Please select another deliveryman.");

81 AllocateTask();

82 } else if (staffid == 1003) {

83 System.out.println("Current deliveryman is in breaktme.Please select another deliveryman.");

84 AllocateTask();

85 } else {

86 System.out.println("Invalid StaffID.Please Enter again");

87 AllocateTask();

88 }

89 while (!deliverylist.isEmpty()) {

90 System.out.println(deliverylist.listDeliveryMan());

91 }

92

93 }

94

95 public void RemainingTime() {

96 String state;

97 Scanner scanner = new Scanner(System.in);

98 System.out.printf("Enter your orderID: ");

99 order = scanner.nextLine();

100 postcode();

101 System.out.println("================================================================================");

102 System.out.println("Please kindly wait our deliveryman to deliver your delicious food.Thank you~");

103 }

104

105 public void postcode() {

106 LocalTime localTime = LocalTime.now();

107 DateTimeFormatter dtf = DateTimeFormatter.ofPattern("HH:mm");

108 deliverylist.addDeliveryMan(deliveryman1);

109 deliverylist.addDeliveryMan(deliveryman2);

110 deliverylist.addDeliveryMan(deliveryman3);

111 String state;

112 String address1 = "185,Persiaran Sg.Keramat 7,Taman Klang Utama,42100 Klang,Selangor"+"\n Order Id: "+order+"\n Deliveryman Name: "+ deliveryman1.getStaff\_name() + "\n ID:"+deliveryman1.getStaffID();

113 String address2 = "A-5-9,PV 10 Platinum Lake Condo No.2,Jalan Danau Sauja,53300 Kuala Lumpur"+"\n Order Id: "+order+"\n Deliveryman Name: "+ deliveryman1.getStaff\_name() + "\n ID:"+deliveryman1.getStaffID();

114 Scanner scanner = new Scanner(System.in);

115 System.out.printf("Enter the postcode: ");

116 System.out.println("1.Selangor(42100)/n2.Kuala Lumpur(53300)");

117 int postcode = scanner.nextInt();

118 if (postcode == 42100) {

119 state = "Selangor";

120 System.out.println("================================================================================");

121 System.out.println("Address: " + address1);

122 System.out.println("State: " + state);

123 String selangor = localTime.plusMinutes(10).format(dtf);

124 System.out.println("Estimate time: " + selangor);

125 System.out.println("Deliveryman: " + deliveryman1.getStaff\_name());

126 } else if (postcode == 53300) {

127 state = "Kuala Lumpur";

128 System.out.println("================================================================================");

129 System.out.println("Address: " + address2);

130 System.out.println("State: " + state);

131 String kuala = localTime.plusMinutes(20).format(dtf);

132 System.out.println("Estimate time: " + kuala);

133 System.out.println("Deliveryman: " + deliveryman1.getStaff\_name());

134 } else {

135 System.out.println("Invalid postcode Please enter again.");

136 postcode();

137 }

138 }

139

140 public void CheckDetails() {

141 deliverylist.addDeliveryMan(deliveryman1);

142 deliverylist.addDeliveryMan(deliveryman2);

143 deliverylist.addDeliveryMan(deliveryman3);

144 System.out.println("====================================================");

145 System.out.println("Staff ID Staff Name\t\tClock In/Out Time Status");

146 while (!deliverylist.isEmpty()) {

147 System.out.println(deliverylist.listDeliveryMan());

148 }

149 }

150

151 public void checkinout() {

152 deliverylist.addDeliveryMan(deliveryman1);

153 deliverylist.addDeliveryMan(deliveryman2);

154 deliverylist.addDeliveryMan(deliveryman3);

155 int staffid;

156 Scanner scanner = new Scanner(System.in);

157 System.out.printf("Enter your staff ID:");

158 staffid = scanner.nextInt();

159 System.out.println("Proceed to page...");

160 System.out.println("=============================================================");

161 DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");

162 Date date = new Date();

163 String break\_time = dateFormat.format(date);

164 if (staffid == 1001) {

165 deliveryman1.setBreak\_time(break\_time);

166 Delivery delivery = new Delivery(deliveryman1.getStaffID(), deliveryman1.getStaff\_name(), deliveryman1.getBreak\_time(), deliveryman1.getAvailable());

167 deliverylist.addDeliveryMan(delivery);

168 int condition;

169 int yes = 1;

170 int no = 2;

171 System.out.println("Clock In and Clock in");

172 System.out.println("======================");

173 System.out.println("1.Clock in");

174 System.out.println("2.Clock Out");

175 int decision = scanner.nextInt();

176 if (decision == 1) {

177 System.out.println("Staff id :" + delivery.getStaffID());

178 System.out.println("Staff Name :" + delivery.getStaff\_name());

179 System.out.println("Clock in Time:" + delivery.getBreak\_time());

180 System.out.println("Clock In Sucessful!!");

181 System.out.println("Continue?(y/n)");

182 System.out.println("1.Yes");

183 System.out.println("2.No");

184 condition = scanner.nextInt();

185 if (no == condition) {

186 System.out.println("Thank You");

187 } else if (yes == condition) {

188 checkinout();

189 }

190 } else if (decision == 2) {

191 System.out.println("Staff id :" + delivery.getStaffID());

192 System.out.println("Staff Name :" + delivery.getStaff\_name());

193 System.out.println("Clock Out Time:" + delivery.getBreak\_time());

194 System.out.println("Clock Out Sucessful!!");

195 System.out.println("Continue?(y/n)");

196 System.out.println("1.Yes");

197 System.out.println("2.No");

198 condition = scanner.nextInt();

199 if (no == condition) {

200 System.out.println("Thank You");

201 } else if (yes == condition) {

202 checkinout();

203 }

204 } else {

205 System.out.println("Invalid Menu Number.");

206 checkinout();

207 }

208 } else if (staffid == 1002) {

209 deliveryman2.setBreak\_time(break\_time);

210 Delivery delivery = new Delivery(deliveryman2.getStaffID(), deliveryman2.getStaff\_name(), deliveryman2.getBreak\_time(), deliveryman2.getAvailable());

211 deliverylist.addDeliveryMan(delivery);

212 int condition;

213 int yes = 1;

214 int no = 2;

215 System.out.println("Clock In and Clock in");

216 System.out.println("======================");

217 System.out.println("1.Clock in");

218 System.out.println("2.Clock Out");

219 int decision = scanner.nextInt();

220 if (decision == 1) {

221 System.out.println("Staff id :" + delivery.getStaffID());

222 System.out.println("Staff Name :" + delivery.getStaff\_name());

223 System.out.println("Clock in Time:" + delivery.getBreak\_time());

224 System.out.println("Clock In Sucessful!!");

225 System.out.println("Continue?(y/n)");

226 System.out.println("1.Yes");

227 System.out.println("2.No");

228 condition = scanner.nextInt();

229 if (no == condition) {

230 System.out.println("Thank You");

231 } else if (yes == condition) {

232 checkinout();

233 }

234 } else if (decision == 2) {

235 System.out.println("Staff id :" + delivery.getStaffID());

236 System.out.println("Staff Name :" + delivery.getStaff\_name());

237 System.out.println("Clock Out Time:" + delivery.getBreak\_time());

238 System.out.println("Clock Out Sucessful!!");

239 System.out.println("Continue?(y/n)");

240 System.out.println("1.Yes");

241 System.out.println("2.No");

242 condition = scanner.nextInt();

243 if (no == condition) {

244 System.out.println("Thank You");

245 } else if (yes == condition) {

246 checkinout();

247 }

248 } else {

249 System.out.println("Invalid Menu Number.");

250 checkinout();

251 }

252 } else if (staffid == 1003) {

253 deliveryman3.setBreak\_time(break\_time);

254 Delivery delivery = new Delivery(deliveryman3.getStaffID(), deliveryman3.getStaff\_name(), deliveryman3.getBreak\_time(), deliveryman3.getAvailable());

255 deliverylist.addDeliveryMan(delivery);

256 int condition;

257 int yes = 1;

258 int no = 2;

259 System.out.println("Clock In and Clock in");

260 System.out.println("======================");

261 System.out.println("1.Clock in");

262 System.out.println("2.Clock Out");

263 int decision = scanner.nextInt();

264 if (decision == 1) {

265 System.out.println("Staff id :" + delivery.getStaffID());

266 System.out.println("Staff Name :" + delivery.getStaff\_name());

267 System.out.println("Clock in Time:" + delivery.getBreak\_time());

268 System.out.println("Clock In Sucessful!!");

269 System.out.println("Continue?(y/n)");

270 System.out.println("1.Yes");

271 System.out.println("2.No");

272 condition = scanner.nextInt();

273 if (no == condition) {

274 System.out.println("Thank You");

275 } else if (yes == condition) {

276 checkinout();

277 }

278 } else if (decision == 2) {

279 System.out.println("Staff id :" + delivery.getStaffID());

280 System.out.println("Staff Name :" + delivery.getStaff\_name());

281 System.out.println("Clock Out Time:" + delivery.getBreak\_time());

282 System.out.println("Clock Out Sucessful!!");

283 System.out.println("Continue?");

284 System.out.println("1.Yes");

285 System.out.println("2.No");

286 condition = scanner.nextInt();

287 if (no == condition) {

288 System.out.println("Thank You");

289 } else if (yes == condition) {

290 checkinout();

291 }

292 } else {

293 System.out.println("Invalid Menu Number.");

294 checkinout();

295 }

296 } else {

297 System.out.println("Invalid Staff Id");

298 checkinout();

299

300 }

301 }

302

303}