



UNIVERSITI
TEKNOLOGI
PETRONAS

TEB 1043 / TFB 1033
Object-Oriented Programming

FINAL PROJECT GROUP JAVA CHIP

Lecturer: Dr Nordin Zakaria

No .	Name	ID No	Program
1	INSYIRAH BINTI HAMZAH	24000157	Bachelor in Computer Science
2	HADHINA SYAHIRA BINTI MUHAMMAD HILMI	24000291	Bachelor in Computer Engineering
3	NUR ALYA SYAZWINA BINTI SAMSUL BAHRIN	24000088	Bachelor in Computer Engineering
4	NUR FATIHAH BINTI MOHD NOOR	24000227	Bachelor in Computer Science

I. INTRODUCTION

1. Purpose

The purpose of this proposal is to outline the development of an object-oriented coffee shop management system named "Java Chip." This system aims to streamline the operations of a coffee shop, enhancing efficiency, customer experience, and overall business management through the use of modern software development practices.

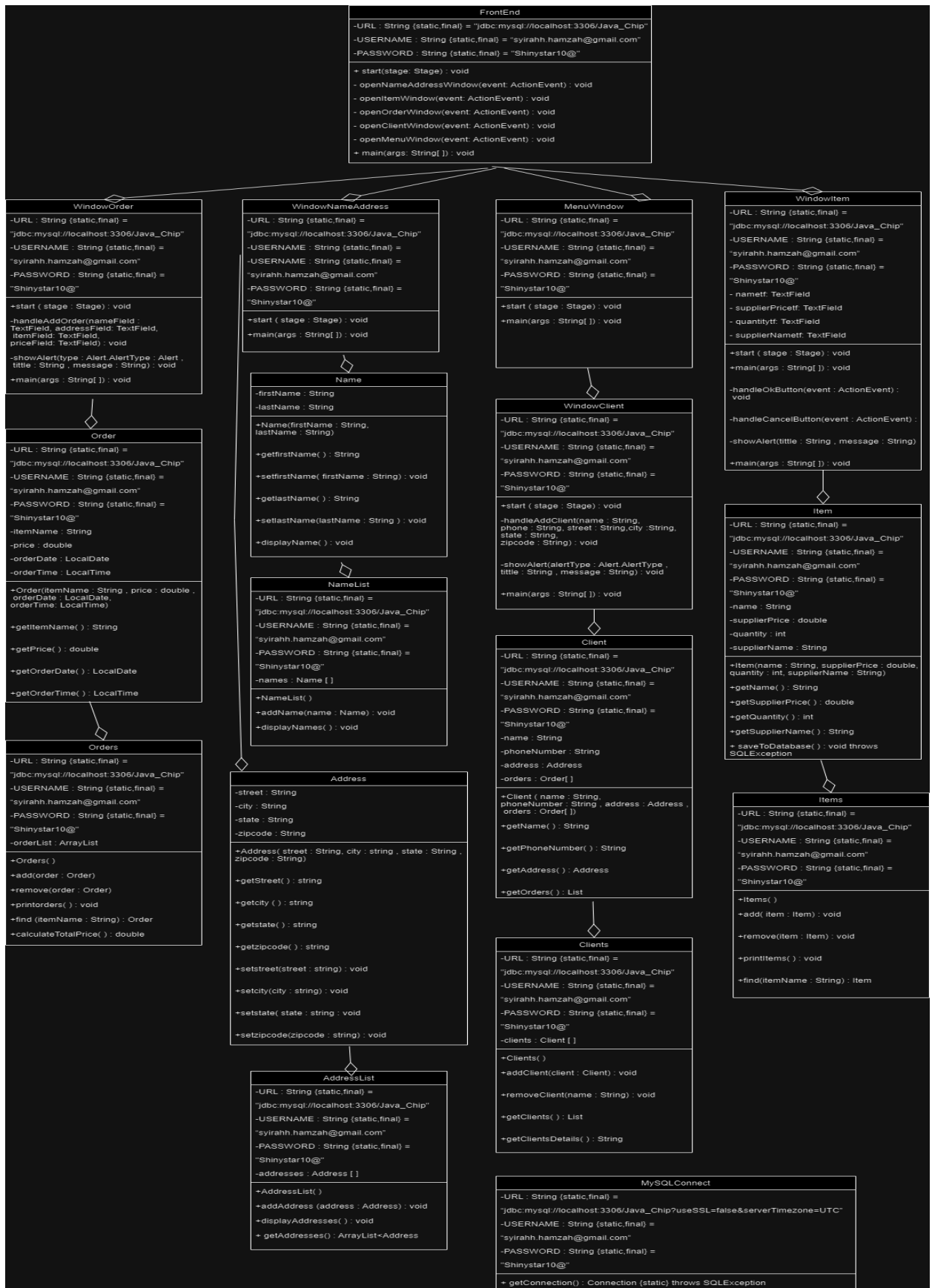
2. Problem Statement

Running a coffee shop involves managing numerous tasks such as order processing, inventory management, staff scheduling, and customer loyalty programs. Traditional manual methods or disjointed software solutions can lead to inefficiencies, errors, and customer dissatisfaction. Java Chip aims to address these challenges by providing a unified and robust management system.

3. Solution

Java Chip will be developed as an object-oriented software application, leveraging the principles of encapsulation, inheritance, and polymorphism to create a modular, maintainable, and scalable system. This system will include features such as order management, inventory tracking, employee scheduling, customer management, and sales reporting. By integrating these functionalities into a single platform, Java Chip will improve operational efficiency and enhance the customer experience.

II. UML DIAGRAM

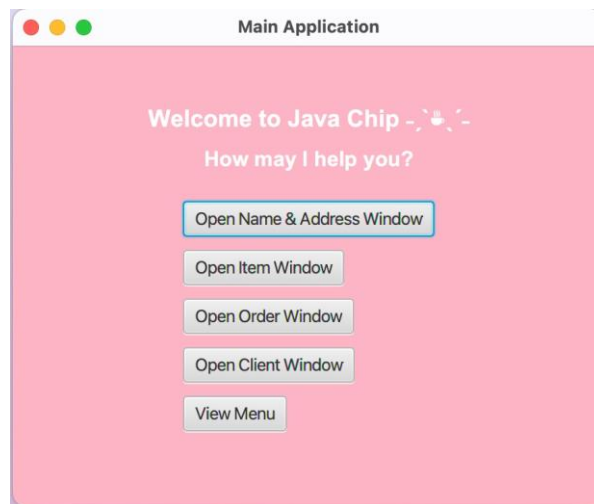


III.CODING DESCRIPTION

1. Frontend

a. Main Application

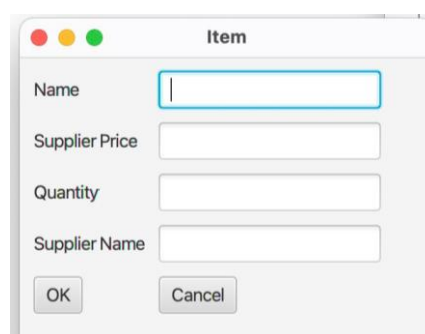
The interface that the user will see when opening the application.



User's interface

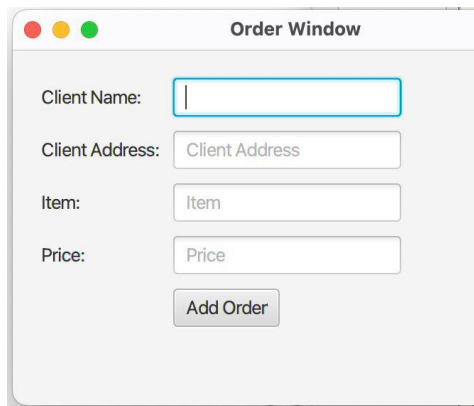
This allows users to directly choose which page they want to redirect to. Order window, item window, client window, name & address window or menu window.

b. Item window

A screenshot of a macOS-style window titled "Item". The window has a light gray background. It contains four text input fields labeled "Name", "Supplier Price", "Quantity", and "Supplier Name". At the bottom, there are two buttons: "OK" and "Cancel".

After clicking “Open Item Window”, users will see this window. They can fill up the details needed.

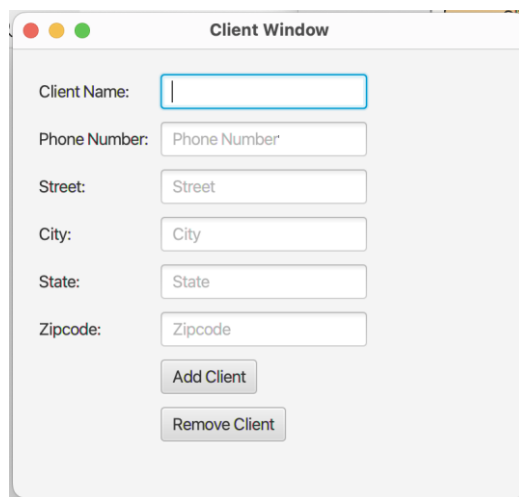
c. Order window



The "Order Window" is a macOS-style window with a title bar containing three colored buttons (red, yellow, green). The window has a light gray background. It contains four text input fields, each with a label to its left: "Client Name:" followed by an empty text box, "Client Address:" followed by a text box containing the placeholder "Client Address", "Item:" followed by a text box containing the placeholder "Item", and "Price:" followed by a text box containing the placeholder "Price". Below these fields is a single button labeled "Add Order".

If users choose to click “Open Order Window”, they will be redirected to this window. Users can fill up the details needed regarding order.

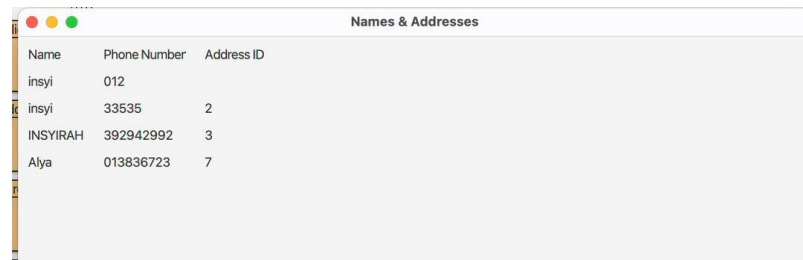
d. Client window



The "Client Window" is a macOS-style window with a title bar containing three colored buttons (red, yellow, green). The window has a light gray background. It contains seven text input fields, each with a label to its left: "Client Name:" followed by an empty text box, "Phone Number:" followed by a text box containing the placeholder "Phone Number", "Street:" followed by a text box containing the placeholder "Street", "City:" followed by a text box containing the placeholder "City", "State:" followed by a text box containing the placeholder "State", and "Zipcode:" followed by a text box containing the placeholder "Zipcode". Below these fields are two buttons: "Add Client" and "Remove Client".

If users click the “Open Client Window” button, this window will appear. Users can fill up the client details, add, and remove it into the client list database.

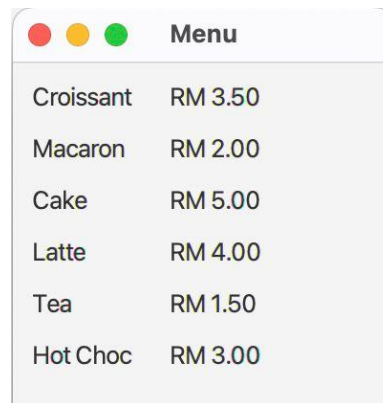
e. Name & Address window



Name	Phone Number	Address ID
insyi	012	
insyi	33535	2
INSYIRAH	392942992	3
Alya	013836723	7

If users click the “Open Name & Address Window” button, this window will appear. This list is used to store and manage contact details for clients.

f. Menu window



Croissant	RM 3.50
Macaron	RM 2.00
Cake	RM 5.00
Latte	RM 4.00
Tea	RM 1.50
Hot Choc	RM 3.00

If users click the “View Menu” button, this window will appear. User may choose to order.

2. Backend

The data and infrastructure that makes the coffee shop application works.

a. Item class

In this class, we build the backend of the Item Window. Each object will have these variables to store its name, supplier price, quantity, and supplier name.

```
private static final String URL = "jdbc:mysql://localhost:3306/Java_Chip";
private static final String USERNAME = "syirahh.hamzah@gmail.com";
private static final String PASSWORD = "Shinystar10@";

private String name;
private double supplierPrice;
private int quantity;
private String supplierName;
```

These are the attributes or properties of the Item class.

```
public Item(String name, String supplierPrice, String quantity, String
    this.name = name;
    this.supplierPrice = supplierPrice;
    this.quantity = quantity;
    this.supplierName = supplierName;
}
```

This is a constructor method and used to initialize the name, supplier price, quantity and supplier name.

```
public String getName() {
    return name;
}
```

These methods allow external code to set the values of the name and return the current values when called.

```
public double getSupplierPrice() {
    return supplierPrice;
}
```

These methods allow external code to set the values of the supplier price and return the current values when called.

```
public int getQuantity() {
    return quantity;
}
```

These methods allow external code to set the values of the quantity and return the current values when called.

```
public String getSupplierName() {
    return supplierName;
}
```

These methods allow external code to set the values of the supplier name and return the current values when called.

```
public void saveToDatabase() throws SQLException {
    String sql = "INSERT INTO items (name, supplier_price, quantity, supplier_name) VALUES (?, ?, ?, ?)";
    try (Connection conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
        PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, this.name);
        pstmt.setDouble(2, this.supplierPrice);
        pstmt.setInt(3, this.quantity);
        pstmt.setString(4, this.supplierName);
        pstmt.executeUpdate();
    }
}
```

The saveToDatabase method adds data to a database. The methods set the values for each data using variable name, supplierPrice, quantity and supplierName.

b. Items class

```
private static final String URL = "jdbc:mysql://localhost:3306/Java_Chip";  
private static final String USERNAME = "syirahh.hamzah@gmail.com";  
private static final String PASSWORD = "Shinystar10@";  
private List<Item> itemList;
```

Declare a variable named 'itemList' and it will hold a collection of 'Item' objects

```
public Items() {  
    this.itemList = new ArrayList<>();  
}
```

This constructor is used to initialize itemList as a type of ArrayList. This creates an empty list that will store 'Item' objects.

```
public void add(Item item) {  
    itemList.add(item);  
}
```

This method is used to add new value of an Item to the itemList.

```
public void remove(Item item) {  
    itemList.remove(item);  
}
```

This method is used to remove the value of Item from the itemList.

```
public void printItems() {  
    for (Item item : itemList) {  
        System.out.println("Name: " + item.getName() +  
            ", Price: RM " + item.getSupplierPrice() +  
            ", Quantity: " + item.getQuantity() +  
            ", Supplier: " + item.getSupplierName());  
    }  
}
```

This method is used to print new Item to the console.

```
public Item find(String itemName) {  
    return itemList.stream()  
        .filter(item -> item.getName().equalsIgnoreCase(itemName))  
        .findFirst()  
        .orElse(null);  
}
```

This method is used to find itemName in the itemList.

c. Order class

In this class, we build the backend of the Order Window. This class serves to hold data of item, price, date, and time of order taken.

```
private static final String URL = "jdbc:mysql://localhost:3306/Java_Chip";
private static final String USERNAME = "syirahh.hamzah@gmail.com";
private static final String PASSWORD = "Shinystar10@";

private String itemName;
private double price;
private LocalDate orderDate;
private LocalTime orderTime;
```

The first three lines of code define constants for connecting to a MySQL database using JDBC (Java Database Connectivity). The next lines are to declare the date type of the attributes.

```
public Order(String itemName, double price, LocalDate orderDate, LocalTime orderTime) {
    this.itemName = itemName;
    this.price = price;
    this.orderDate = orderDate;
    this.orderTime = orderTime;
}
```

This constructor is used to initialize the item, price, date, and time attributes. As well as to allow pass value from users to the parameters in the ().

```
public String getItemName() {
    return itemName;
}
```

This method is used to retrieve value of itemName from the user.

```
public double getPrice() {
    return price;
}
```

This method is used to set value of the price and retrieve value of price from the user.

```
public LocalDate getOrderDate() {
    return orderDate;
}
```

This method is used to retrieve value of date from the user.

```
public LocalTime getOrderTime() {
    return orderTime;
}
```

This method is used to retrieve value of time from the user.

d. Orders class

```
private static final String URL = "jdbc:mysql://localhost:3306/Java_Chip";
private static final String USERNAME = "syirahh.hamzah@gmail.com";
private static final String PASSWORD = "Shinystar10@";

private ArrayList<Order> orderList = new ArrayList<>();
```

The first three lines of code define constants for connecting to a MySQL database using JDBC (Java Database Connectivity). Declaring orderlist as an array of list that holds the values of Order.

```
public Orders() {
}
```

This default constructor serves to create instances of Orders class.

```
public void add(Order order) {
    orderList.add(order);
}
```

This method is used to add the new value of order into the orderList.

```
public void remove(Order order) {
    orderList.remove(order);
}
```

This method is used to remove the existing value of order from the orderList.

```
public void printOrders() {
    for (Order order : orderList) {
        System.out.println("Item: " + order.getItemName() + ", Price: RM " + order.getPrice() + ", Date: " + order.getOrderDate() + ", Time: " + order.getOrderTime());
    }
}
```

This method is used to compare the recently order input added with the existing data in orderList. If there is no match, this method will display the order added.

```
public Order find(String itemName) {  
    for (Order order : orderList) {  
        if (order.getItemName().equalsIgnoreCase(itemName)) {  
            return order;  
        }  
    }  
    return null;  
}
```

This method is used to find the name of order added in the orderList by comparing it with the existing itemName.

```
public double calculateTotalPrice() {  
    double total = 0;  
    for (Order order : orderList) {  
        total += order.getPrice();  
    }  
    return total;  
}
```

This method is used to calculate the total price of the orders and display it on Order Window.

e. Name class

```
protected String firstName, lastName;
```

This line declares instances variable firstname and lastname of type String.

```
public Name(String firstName, String lastName)
{
    this.firstName = firstName;
    this.lastName = lastName;
}
```

This is a constructor method and used to initialize the firstname and lastname.

```
//getter
public String getFirstName() {
    return firstName;
}

public String getLastName() {
    return lastName;
}
```

These getter methods allow external code to set the values of the firstname and lastname and retrieve the value of firstname and lastname from the user.

```
//setter  
public void setFirstName(String firstName) {  
    this.firstName = firstName;  
}  
  
public void setLastName(String lastName) {  
    this.lastName = lastName;  
}
```

Setter methods are used to set or update the value of the firstname and lastname.

```
public void displayName()  
{  
    System.out.println("Client's name: " + firstName + lastName);  
}
```

This method prints the firstname and lastname of the client to the console.

f. NameList class

```
private static final String URL = "jdbc:mysql://localhost:3306/Java_Chip";
private static final String USERNAME = "syirahh.hamzah@gmail.com";
private static final String PASSWORD = "Shinystar10@";
private ArrayList<Name> names;
```

Declare a private variable named 'names' and it will hold a collection of 'Name' objects.

```
public NameList() {
    this.names = new ArrayList<>();
}
```

Declare a constructor for the nameList class and initialize 'names' as a type of ArrayList. This creates an empty list that will store 'Name' objects.

```
public void addName(Name name) {
    names.add(name);
}
```

This method is used to add new Name to the names list.

```
public void displayNames() {
    for (Name name : names) {
        name.displayName();
    }
}
```

This method is used to show the name of the clients

g. Address class

```
private String street;  
private String city;  
private String state;  
private String zipcode;
```

This line declares instances variable city, state and zipcode of type String.

```
public Address(String city,String state,String zipcode)  
{  
    this.city = city;  
    this.state = state;  
    this.zipcode = zipcode;  
}
```

This is a constructor method and used to initialize the city, state and zipcode.

```
// Getters  
public String getStreet() {  
    return street;  
}  
public String getCity() {  
    return city;  
}  
public String getState() {  
    return state;  
}  
public String getZipcode() {  
    return zipcode;  
}
```

These getter methods allow external code to set the values of the street, city, state and zipcode and retrieve the value of street, city, state and zipcode from the user.

```
// Setters
public void setStreet(String street) {
    this.street = street;
}
public void setCity(String city) {
    this.city = city;
}
public void setState(String state) {
    this.state = state;
}
public void setZipcode(String zipcode) {
    this.zipcode = zipcode;
}
```

Setter methods are used to set or update the value of the street, city, state and zipcode.

h. AddressList class

```
private static final String URL = "jdbc:mysql://localhost:3306/Java_Chip";  
private static final String USERNAME = "syirahh.hamzah@gmail.com";  
private static final String PASSWORD = "Shinystar10@";  
  
private ArrayList<Address> addresses;
```

The first three lines of code define constants for connecting to a MySQL database using JDBC (Java Database Connectivity). Declaring addresses as an array of list that holds the values of Address.

```
public AddressList() {  
    this.addresses = new ArrayList<>();  
}
```

This constructor is used to initialize addresses as an ArrayList.

```
public void addAddress(Address address) {  
    addresses.add(address);  
}
```

This method is used to add address input into the addresses arraylist.

```
public void displayAddresses() {  
    for (Address address : addresses) {  
        System.out.println(address);  
    }  
}
```

This method is used to display the address after comparing it with the existing address in addresses arraylist.

```
public ArrayList<Address> getAddresses() {  
    return addresses;  
}
```

This method is used to retrieve the list of address in addresses.

i. Client class

```
private static final String URL = "jdbc:mysql://localhost:3306/Java_Chip"
private static final String USERNAME = "syirahh.hamzah@gmail.com";
private static final String PASSWORD = "Shinystar10@";

private String name;
private String phoneNumber;
private Address address;
private List<Order> orders;
```

The first three lines of code define constants for connecting to a MySQL database using JDBC (Java Database Connectivity). The next lines are to declare the data type of the attributes.

```
public Client(String name, String phoneNumber, Address address, List<Order> orders) {
    this.name = name;
    this.phoneNumber = phoneNumber;
    this.address = address;
    this.orders = orders;
}
```

This constructor is used to pass value from input through the parameters and to initialize the attributes.

```
public String getName() {
    return name;
}
```

This method is used to retrieve the name value.

```
public String getPhoneNumber() {
    return phoneNumber;
}
```

This method is used to retrieve the phoneNumber value.

```
public Address getAddress() {
    return address;
}
```

This method is used to retrieve the address value.

```
public List<Order> getOrders() {
    return orders;
}
```

This method is used to retrieve the value orders.

j. Clients class

```
private static final String URL = "jdbc:mysql://localhost:3306/Java_Chip";  
private static final String USERNAME = "syirahh.hamzah@gmail.com";  
private static final String PASSWORD = "Shinystar10@";  
private List<Client> clients;
```

The first three lines of code define constants for connecting to a MySQL database using JDBC (Java Database Connectivity). Declaring clients as an array of list that holds the values of Client.

```
public Clients() {  
    clients = new ArrayList<>();  
}
```

This constructor is used to initialize clients as an arraylist.

```
public void addClient(Client client) {  
    clients.add(client);  
}
```

This method is used to add client input into the clients arraylist.

```
public void removeClient(String name) {  
    clients.removeIf(client -> client.getName().equals(name));  
}
```

This method is used to remove client from the clients arraylist if same as name.

```
public List<Client> getClients() {  
    return new ArrayList<>(clients);  
}
```

This method is used to return the new list of the clients.






```
public String getClientsDetails() {  
    StringBuilder details = new StringBuilder();  
    for (Client client : clients) {  
        details.append("Client Name: ").append(client.getName()).append("\n");  
        details.append("Phone Number: ").append(client.getPhoneNumber()).append("\n");  
  
        details.append("Orders: ");  
        if (client.getOrders() != null && !client.getOrders().isEmpty()) {  
            for (Order order : client.getOrders()) {  
                details.append(order.getItemName()).append(" (Price: RM ").append(order.getPrice()).append(", ");  
            }  
            // Remove the trailing comma and space  
            details.setLength(details.length() - 2);  
        } else {  
            details.append("No orders");  
        }  
        details.append("\n\n");  
    }  
    return details.toString();  
}
```

This method is used to return a detailed string representation of all clients and their orders.

3. Database Table

A database table is a collection of data organized in rows and columns. Each table represents a specific type of data. Tables are fundamental to the structure of a relational database and are used to store and manage data efficiently.

a. Address

<div><div>←T→</div><div></div></div>						id	city	state	zipcode	street	
<input type="checkbox"/>		Edit		Copy		Delete	1	UTP	SI	36201	123 Blok v2c
<input type="checkbox"/>		Edit		Copy		Delete	2	42	24	2424	fr
<input type="checkbox"/>		Edit		Copy		Delete	3	si	perak	23242	V2 UTP
<input type="checkbox"/>		Edit		Copy		Delete	7	SI	Perak	388372	utp

This table allows the user to manage information by editing item details, duplicating entries, or deleting them as needed. The interface is typical for managing a list of addresses, including cities, zip codes and streets.

b. Clients

<div><div>←</div><div>T</div><div>→</div></div>					<div>▼</div>	id	name	phone_number	address_id
<div><div><input type="checkbox"/></div></div>	<div><div></div><div>Edit</div></div>	<div><div></div><div>Copy</div></div>	<div><div></div><div>Delete</div></div>	1	insyi	012	NULL		
<div><div><input type="checkbox"/></div></div>	<div><div></div><div>Edit</div></div>	<div><div></div><div>Copy</div></div>	<div><div></div><div>Delete</div></div>	2	insyi	33535	2		
<div><div><input type="checkbox"/></div></div>	<div><div></div><div>Edit</div></div>	<div><div></div><div>Copy</div></div>	<div><div></div><div>Delete</div></div>	3	INSYIRAH	392942992	3		
<div><div><input type="checkbox"/></div></div>	<div><div></div><div>Edit</div></div>	<div><div></div><div>Copy</div></div>	<div><div></div><div>Delete</div></div>	7	Alya	013836723	7		






















This table allows the user to manage contact information by editing details, duplicating entries, or deleting them as needed. The interface is typical for managing a list of contacts, including phone numbers and associated addresses. The presence of NULL in the address_id column indicates that some entries may not have an address assigned.

c. Items

						id	name	supplier_price	quantity	supplier_name
<input type="checkbox"/>	 Edit	 Copy	 Delete	1	latte	5.00	100	teha		
<input type="checkbox"/>	 Edit	 Copy	 Delete	2	fsdfs	5.00	24	slfkf		
<input type="checkbox"/>	 Edit	 Copy	 Delete	3	Insyirah	100.00	20	abc124		
<input type="checkbox"/>	 Edit	 Copy	 Delete	4	Alya	100.00	250	abc263		

This table allows the user to manage inventory by editing item details, duplicating entries, or deleting them as needed. The interface is typical for managing a list of products or supplies, including tracking supplier information and stock quantities.

d. Menu

← T →				id	item_name	price
<input type="checkbox"/>	 Edit	 Copy	 Delete	1	Croissant	3.50
<input type="checkbox"/>	 Edit	 Copy	 Delete	2	Macaron	2.00
<input type="checkbox"/>	 Edit	 Copy	 Delete	3	Cake	5.00
<input type="checkbox"/>	 Edit	 Copy	 Delete	4	Latte	4.00
<input type="checkbox"/>	 Edit	 Copy	 Delete	5	Tea	1.50
<input type="checkbox"/>	 Edit	 Copy	 Delete	6	Hot Choc	3.00
<input type="checkbox"/>	 Edit	 Copy	 Delete	8	Green Tea	5.50

This table allows the user to manage items by editing their details, duplicating entries, or deleting them as needed. The interface is typical for managing a menu or product list in a small business, such as a cafe or bakery.