SLIIT
Discover Your Future

BSc (Hons) in Information Technology

Object Oriented Concepts – IT1050

Assignment 2

Year 1, Semester 2 **2023-July-Dec**

Cover Page:

Topic : Online Dry Cleaning and Laundry Services
Group no : Y1S2_23_MTR_Gr02
Campus: Matara

Submission Date: 31st of October 2023

We declare that this is our own work, and this Assignment does not incorporate without acknowledgment any material previously submitted by anyone else in SLIIT or any other university/Institute. And we declare that each one of us equally contributed to the completion of this Assignment.

Registration No	Name	Contact Number	Individual Contribution
It22253958	W.P.R. Nethmina	0703318808	Marketer, Campaign
IT22296078	Sarithmal K.D	0788724258	Service, Report
IT22226532	D.V.D Hashan	0741750500	Unregistered User, Complaint
IT22635952	Abeywickrama A.S.	0779195607	Registered User, Administrator
IT22244352	Hewahalpage	0760702357	Dry Cleaner, Manager
IT22371522	G.H.P Iroshan	0719366028	Payment, Order

Contents

1. Description of the Requirements. 3
2. Identified Classes. 4
3. CRC Cards. 5
4. Class Diagram 9
5. Coding for the Classes. 10
    1. Main.cpp 10
    2. RegisteredUser.h 13
    3. RegisteredUser.cpp 14
    4. Manager.h 16
    5. Manager.cpp 17
    6. DryCleaner.h 18
    7. DryCleaner.cpp 19
    8. Administrator.h 20
    9. Administrator.cpp 21
    10. Service.h 22
    11. Service.cpp 23
    12. Order.h 24
    13. Order.cpp 25
    14. Report.h 26
    15. Report.cpp 27
    16. UnregisteredUser.h 28
    17. UnregisteredUser.cpp 29
    18. Marketer.h 30
    19. Marketer.cpp 31
    20. Campaign.h 32
    21. Campaign.cpp 33
    22. Complaint.h 34
    23. Complaint.cpp 35
    24. Payment.h 36
    25. Payment.cpp 37

1. Description of the Requirements.

- All the users can browse available services and prices, view general information about the company, view terms and conditions.
- Registered users can Log in to the system.
- Registered users can place an order for the relevant service.
- Registered users can view order history.
- Registered users can update profile information and contact support.
- Unregistered users can browse available services and prices.
- Unregistered users can view general information about the company.
- Managers can view and manage orders.
- Managers can assign orders to dry cleaners.
- Managers can Generate reports and Manage user accounts.
- Administrator can manage system settings and configurations.
- Administrator can Add, edit, or remove services and pricing.
- Administrators can Monitor and manage user accounts.
- Administrators can Handle customer complaints and disputes.
- Dry Cleaner can Accept and fulfill orders, Update order status (e.g., in-progress, completed)
- Dry Cleaner can Request assistance.
- Marketer can Create and manage marketing campaigns, Analyze customer data for targeted advertising.
- Marketer can Track the success of marketing efforts.

2. Identified Classes.

1. Registered user.
2. Unregistered user.
3. Manager.
4. Administrator.
5. Dry Cleaner.
6. Marketer.
7. Service
8. Order
9. Report
10. Payment
11. Campaign.
12. Complaint

3. CRC Cards.

Registered User	
Responsibilities	**Collaborations**
Log in to the system	
Browse services and prices.	Service, payment
View order history.	order
Update Profile	
Contact Support	
Place order	service

Unregistered User	
Responsibilities	**Collaborations**
Browse services and prices	service
Register	

Manager	
Responsibilities	**Collaborations**
View Order	Order
Manage orders	
Assign Orders to dry cleaner	Dry cleaner
Generate reports	report
Manage user accounts	

Administrator	
Responsibilities	**Collaborations**
Manage system settings	
Edit services	service
Handle complaints	Complaints

Dry Cleaner	
Responsibilities	**Collaborations**
Accept order	order
Fulfil order	
Update Order Status	
Request assistance	Manager

Bank	
Responsibilities	**Collaborations**
Authorize payment	payment
Generate financial reports	report

Marketer	
Responsibilities	**Collaborations**
Create marketing campaign	campaign
Manage marketing campaign	
Analyze customer data	Registered user

Service	
Responsibilities	**Collaborations**
Browse services and prices	Registered user
Place order	Registered user
Accept order	
Fulfil an order	Dry cleaner

Order	
Responsibilities	**Collaborations**
Update order status	Dry cleaner
Get order details	
Calculate order cost	payment
View order history	Registered user

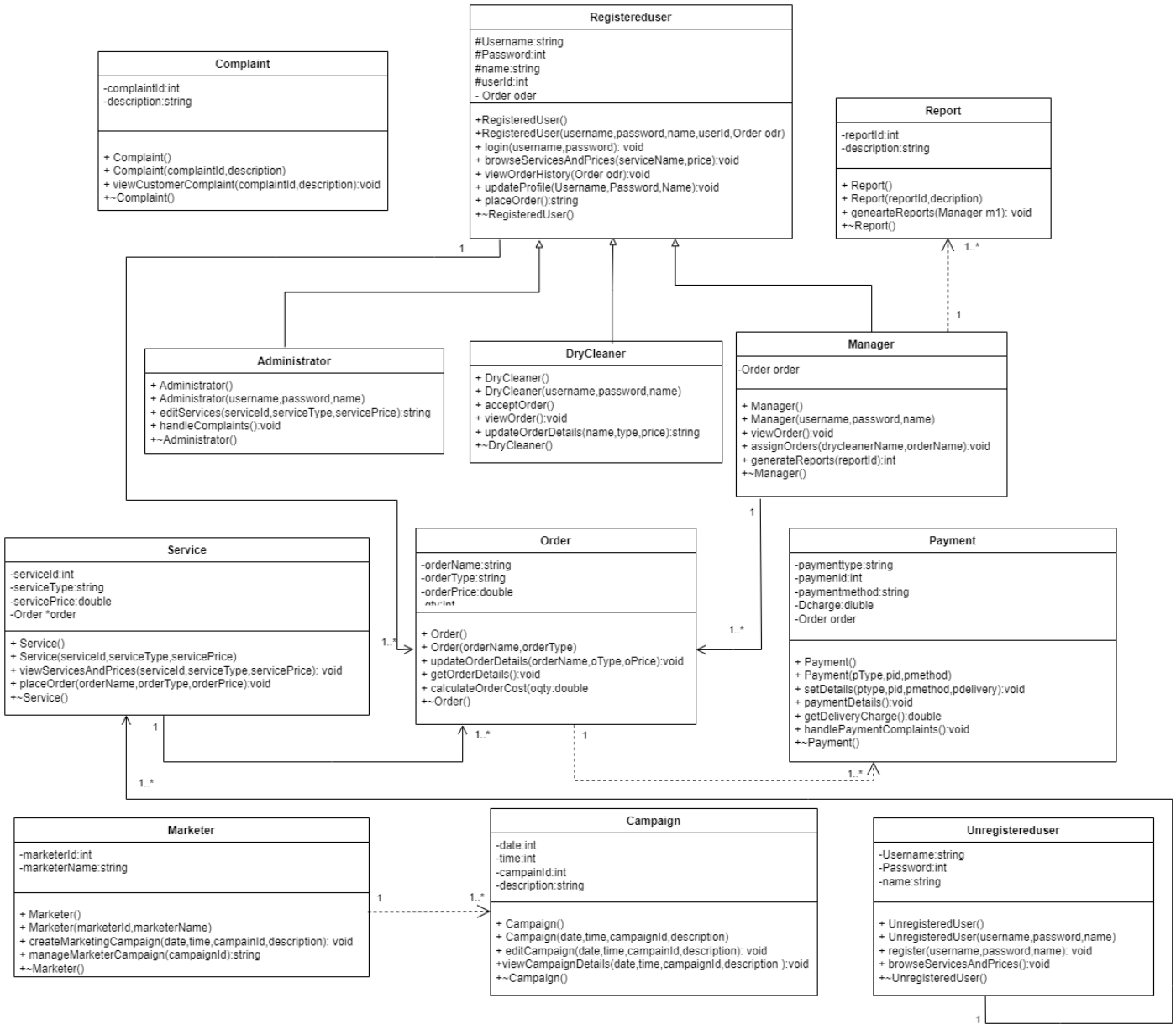
Report	
Responsibilities	**Collaborations**
Generate a report	manager
View a report	
Email report	

Payment	
Responsibilities	**Collaborations**
Browse price	Registered user
Calculate order cost	order
Process payments	
Handle payment complaints	Complaints

Campaign	
Responsibilities	**Collaborations**
Create a campaign	marketer
Edit Campaign	
Launch campaign	
View campaign details	

Complaint	
Responsibilities	**Collaborations**
Handle customer complaints	administrator
Handle payment complaints	payments

4. Class Diagram



5. Coding for the Classes.

1. Main.cpp

```
#include <iostream>
#include "Campaign.h"
#include "Marketer.h"
#include "RegisteredUser.h"
#include "Administrator.h"
#include "Complaint.h"
#include "DryCleaner.h"
#include "Manager.h"
#include "Order.h"
#include "Payment.h"
#include "Report.h"
#include "Service.h"
#include "UnRegisteredUser.h"


using namespace std;

int main() {

//Marketer and Campaign class
 // Create a Campaign
    Campaign campaign1(20231029, 1400, 1, "New Product Launch");

    // View Campaign Details
    campaign1.viewCampaignDetails();

    // Create a Marketer
    Marketer marketer1(1, "John Doe");

    // Create a campaign and assign it to the Marketer
    marketer1.createMarketingCampaign(20231029, 701, 2, "New
Session");

    // Attempt to create another Campaign
    marketer1.createMarketingCampaign(20231101, 1500, 3, "Holiday
Promotion");

//registered user
```

```
    RegisteredUser user1("Ravindu",1234,"RN",1);
    user1.login("Ravindu" ,1234);
    user1.browseServicesAndPrices();
    user1.viewOrderHistory();
    user1.placeOrder("Bob", "Clean", 1000);
//manager

    Manager m1;
    m1.viewOrder();
    m1.assignOrder("Bob", "Clean", 1000);
    m1.generateReports(1, "Report About Me");

//Dry Cleaner

    DryCleaner dc1;
    dc1.updateOrderDetails("Bob", "Clean", 1000);
    dc1.acceptOrder("none", "press", 2000);
    dc1.viewOrder();

//Administrator

    Administrator admin;
    admin.editServices(1, "Normal", 1000);
    admin.handleComplaint();

//Complaint

    Complaint com1(1,"About Service");
    com1.viewCustomerComplaints();

 //Order

    Order oder1("Normal", "Pressing", 5000);
    oder1.getOrderDetails();
    oder1.calculateOrderCost(10);

//Payment

    Payment pay1("Online", 1, "Credit Card");
    pay1.paymentdetails();
    pay1.getDeliveryCharge();
    pay1.paymentcomplaint();

//Report
```

```
    Report r1(1,"Payment Report");
    r1.generateReports(2, "About Salary");

//Service

    Service service1(1,"Cleaning",1500);
    service1.placeOrder("Daily", "Customer", 3200);
    service1.ViewServicesAndPrices();

//Unregistered User

    UnRegisteredUser ur1("RV", 1234, "Ravindu Nethmina");
    ur1.register("RV", 1234, "Ravindu Nethmina");
    ur1.browseServicesAndPrices();

     return 0;
}
```

2. RegisteredUser.h

```
#pragma once
#include <iostream>
#include "Administrator.h"
#include "DryCleaner.h"
#include "Manager.h"
#include "Order.h"
#include "Service.h"

using namespace std;

class RegisteredUser
{
protected:
	string Username;
	int password;
	string name;
	int userId;

private:
	Order order;

public:
	RegisteredUser();
	RegisteredUser(string uName, int pw, string rName, int
uId);
	void login(string uName, int pw);
	void browseServicesAndPrices();//view services and
prices from services class
	void updateProfile(string uName, int pw, string rName,
int uId);
	void viewOrderHistory();
	string placeOrder(string orderName, string orderType,
double orderPrice); //create a new order by using this
function
	~RegisteredUser();
};
```

3. RegisteredUser.cpp

```
#include "RegisteredUser.h"
#include "Order.h"

#include <iostream>

using namespace std;

RegisteredUser::RegisteredUser() {
	Username = "";
	password = 0;
	name = "";
	userId = 0;
}
RegisteredUser::RegisteredUser(string uName, int pw, string
rName, int uId) {
    Username = uName;
	password = pw;
	name = rName;
	userId = uId;
}
void RegisteredUser::login(string uName, int pw) {

	if (Username == uName && password == pw) {
		cout << "Login Successful!" << endl;
	}
	else if (Username != uName || password != pw) {
		cout << "Login Unsucceful!" << endl;
	}
}
void RegisteredUser::browseServicesAndPrices() {
	Service service1;
	service1.ViewServicesAndPrices();
}
void RegisteredUser::updateProfile(string uName, int pw,
string rName, int uId) {
	Username = uName;
	password = pw;
	name = rName;
	userId = uId;
}
void RegisteredUser::viewOrderHistory() {
	Order order;
	order.getOrderDetails();
```

```
}
string RegisteredUser::placeOrder(string orderName, string
orderType, double orderPrice){

     try {
          Order order1(orderName,orderType,orderPrice);
          // If the Order constructor completes without
exceptions,
          // it means the order has been successfully placed.
          return "Order placed successfully.";
     }
     catch (const exception& e) {
          // If an exception is thrown during the
construction of the Order object,
          // catch it and display an error message.
          return "Order Placing Unsucceful!";
     }

}
RegisteredUser::~RegisteredUser() {

}
```

4. **Manager.h**

```
#pragma once
#include "RegisteredUser.h"
#include "Order.h"
#include "Report.h"
#include "DryCleaner.h"
#include <iostream>

using namespace std;

class Manager : public RegisteredUser
{
private:
	Order order;
public:
	Manager();
	Manager(string uName, int pw, string rName, int uId);
	void viewOrder();
	void assignOrder(string oName, string oType, double
oPrice);
	void generateReports(int reportId, string rDes);
	~Manager();

};
```

5. Manager.cpp

```
#include "Manager.h"
#include "RegisteredUser.h"
#include "Order.h"
#include "Report.h"
#include "DryCleaner.h"
#include <iostream>

using namespace std;


Manager::Manager() {
	Username = "";
	password = 0;
	name = "";
	userId = 0;
}
Manager::Manager(string uName, int pw, string rName, int uId)
{
	Username = uName;
	password = pw;
	name = rName;
	userId = uId;
}
void Manager::viewOrder() {
	Order order;
	order.getOrderDetails();
}
void Manager::assignOrder(string oName, string oType, double
oPrice) {
	DryCleaner drycln;
	drycln.acceptOrder(oName,oType,oPrice);
}
void Manager::generateReports(int reportId, string rDes) {
	Report report;
	report.generateReports(reportId, rDes);
}
Manager::~Manager() {

}
```

6. DryCleaner.h

```
#pragma once
#include <iostream>
#include "RegisteredUser.h"
#include "Order.h"

using namespace std;

class DryCleaner : public RegisteredUser
{
public:
	DryCleaner();
	DryCleaner(string uName, int pw, string rName, int uId);
	void viewOrder();
	string updateOrderDetails(string oName, string oType,
double oPrice);
	void acceptOrder(string oName, string oType, double
oPrice);
	~DryCleaner();

};
```

7. DryCleaner.cpp

```
#include "DryCleaner.h"
#include <iostream>
#include "RegisteredUser.h"
#include "Order.h"

using namespace std;

DryCleaner::DryCleaner() {
	Username = "";
	password = 0;
	name = "";
	userId = 0;
}
DryCleaner::DryCleaner(string uName, int pw, string rName, int
uId)  {
	Username = uName;
	password = pw;
	name = rName;
	userId = uId;

}
void DryCleaner::viewOrder() {
	Order order;
	order.getOrderDetails();
}
string DryCleaner::updateOrderDetails(string oName, string oType,
double oPrice) {
	Order order1(oName, oType, oPrice);
	return "Successfull!";
}
void DryCleaner::acceptOrder(string oName, string oType, double
oPrice) {
	Order odr(oName,oType,oPrice);
}
DryCleaner:: ~DryCleaner() {

}
```

8. Administrator.h

```
#pragma once
#include "RegisteredUser.h"
#include "Service.h"
#include "Complaint.h"
#include <iostream>

using namespace std;

class Administrator : public RegisteredUser
{
public:
	Administrator();
	Administrator(string uName, int pw, string rName, int uId);
	string editServices(int sId, string sType, double sPrice);
	void handleComplaint();
	~Administrator();
};
```

9. Administrator.cpp

```
#include "Administrator.h"
#include <iostream>
#include "Service.h"
#include "Complaint.h"

using namespace std;

Administrator::Administrator() {
    Username = "";
     password = 0;
     name = "";
     userId = 0;
}
Administrator::Administrator(string uName, int pw, string rName,
int uId) {
     Username = uName;
     password = pw;
     name = rName;
     userId = uId;
}
string Administrator::editServices(int sId, string sType, double
sPrice) {

     try {
          Service service(sId,sType,sPrice);
          return "Service Edit Successful!";
     }
     catch (const exception& e) {
          return "Service Edit Unsucceful!";
     }

}
void Administrator::handleComplaint() {
     Complaint complain;
     complain.viewCustomerComplaints();
}
Administrator::~Administrator() {

}
```

10.Service.h

```
#pragma once

#include <iostream>
#include "Order.h"

using namespace std;
class Service
{
private:
	int serviceId;
	string serviceType;
	double servicePrice;
	Order order;

public:
	Service();
	Service(int sId, string sType, double sPrice);
	void ViewServicesAndPrices();
	void placeOrder(string oName,string oType,int oPrice);
	~Service();

};
```

11.Service.cpp

```
#include "Service.h"
#include <iostream>

using namespace std;

Service::Service() {
	serviceId = 0;
	serviceType = "";
	servicePrice = 0;
}
Service::Service(int sId, string sType, double sPrice) {
	serviceId = sId;
	serviceType = sType;
	servicePrice = sPrice;
}
void Service::ViewServicesAndPrices() {
	cout << endl;
	cout << "Service ID : " << serviceId << endl;
	cout << "Service Type : " << serviceType << endl;
	cout << "Servive Price : " << servicePrice << endl;
}
void Service::placeOrder(string oName, string oType, int oPrice) {
	Order order(oName,oType,oPrice);
}
Service::~Service() {

}
```

12.Order.h

```
#pragma once
#include <iostream>
#include "Order.h"

using namespace std;

class Order
{
private:
	string orderName;
	string orderType;
	double orderPrice;
	int qty;

public:
	Order();
	Order(string oName,string oType,double oPrice);
	void updateOrderDetails(string oName, string oType, double
oPrice);
	void getOrderDetails();
	double calculateOrderCost(int oQty);
	~Order();
};
```

13.Order.cpp

```
#include "Order.h"
#include <iostream>

using namespace std;

Order::Order() {
	orderName = "";
	orderType = "";
	orderPrice = 0;
}
Order::Order(string oName, string oType, double oPrice) {
	orderName = oName;
	orderType = oType;
	orderPrice = oPrice;
}
void Order::updateOrderDetails(string oName, string oType, double
oPrice) {
	orderName = oName;
	orderType = oType;
	orderPrice = oPrice;
}

void Order::getOrderDetails() {
	cout << endl;
	cout << "Order Name : " << orderName << endl;
	cout << "Order Type : " << orderType << endl;
	cout << "Order Price : " << orderPrice << endl;
	cout << endl;

}
double Order::calculateOrderCost(int oQty) {
	return oQty * orderPrice;
}
Order::~Order(){

};
```

14.Report.h

```
#pragma once
#include <iostream>

using namespace std;

class Report
{
private:
	int reportId;
	string reportDes;

public:
	Report();
	Report(int rId, string rDes);
	void generateReports(int rId, string rDes);
	~Report();

};
```

15.Report.cpp

```
#include "Report.h"
#include <iostream>

using namespace std;


Report::Report() {
	reportId = 0;
	reportDes = "";

}
Report::Report(int rId, string rDes) {
	reportId = rId;
	reportDes = rDes;
}
void Report::generateReports(int rId, string rDes) {
	reportId = rId;
	reportDes = rDes;
}
Report::~Report() {

}
```

16.UnregisteredUser.h

```
#pragma once
#include <iostream>

using namespace std;

class UnRegisteredUser
{
public:
	string Username;
	int Password;
	string name;
private:
	UnRegisteredUser();
	UnRegisteredUser(string uName, int passw, string name);
	void Register(string uName, int passw, string name);
	void browseServicesAndPrices();
	~UnRegisteredUser();

};
```

17.UnregisteredUser.cpp

```
#include "UnRegisteredUser.h"
#include "Service.h"
#include <iostream>

using namespace std;

UnRegisteredUser::UnRegisteredUser() {
	Username = "";
	Password = 0;
	name = "";
}
UnRegisteredUser::UnRegisteredUser(string uName, int passw, string
name) {
	Username = uName;
	Password = passw;
	name = name;
}
void UnRegisteredUser::Register(string uName, int passw, string
name) {
	Username = uName;
	Password = passw;
	name = name;
}
void UnRegisteredUser::browseServicesAndPrices() {
	Service service;
	service.ViewServicesAndPrices();
}
UnRegisteredUser::~UnRegisteredUser() {


}
```

18.Marketer.h

```
#pragma once
#include "Campaign.h"
#include <iostream>

using namespace std;

class Marketer {
private:
    int marketerId;
    string marketerName;

public:
    Marketer();
    Marketer(int mId,string maName);
    void createMarketingCampaign(int cDate, int cTime, int cId,
string cDescription);
    string manageMarketerCampaign(int cId);
    ~Marketer();
};
```

19.Marketer.cpp

```
#include "Marketer.h"
#include "Campaign.h"

Campaign* campaign; //creating a pointer varible to create a
dynamic memory

Marketer::Marketer() {
    marketerId = 0;
    marketerName = "";
}

Marketer::Marketer(int mId, string maName) {
    marketerId = mId;
    marketerName = maName;
}

void Marketer::createMarketingCampaign(int cDate, int cTime, int
cId, string cDescription) {
    if (campaign == nullptr) {
        campaign = new Campaign(cDate, cTime, cId, cDescription);
    }
    else {
        cout << "A campaign already exists. Close the current
campaign to create a new one." << endl;
    }
}

string Marketer::manageMarketerCampaign(int cId) {
    if (campaign != nullptr) {
        return "Campaign managed successfully.";
    }
    else {
        return "Campaign not found or not assigned to this
marketer.";
    }
}

Marketer::~Marketer() {
    if (campaign != nullptr) {
        delete campaign;
    }
}
```

20.Campaign.h

```
#pragma once
#include <iostream>
#include "Marketer.h"

using namespace std;
class Campaign {
private:
    int date;
    int time;
    int campaignId;
    string description;
public:
    Campaign();
    Campaign(int cDate,int cTime, int cId,string cDescription);
    void editCampaign(int cDate, int cTime, int cId, string
cDescription);
    void viewCampaignDetails();
    ~Campaign();
};
```

21.Campaign.cpp

```
#include "Campaign.h"
#include "Marketer.h"

Campaign::Campaign() {
    date = 0;
    time = 0;
    campaignId = 0;
    description = "";
}

Campaign::Campaign(int cDate, int cTime, int cId, string
cDescription) {
    date = cDate;
    time = cTime;
    campaignId = cId;
    description = cDescription;
}

void Campaign::editCampaign(int cDate, int cTime, int cId, string
cDescription) {
    date = cDate;
    time = cTime;
    campaignId = cId;
    description = cDescription;
}

void Campaign::viewCampaignDetails() {
    cout << endl;
    cout << " Campaign Date : " << date << endl;
    cout << " Campaign Time : " << time << endl;
    cout << " Campaign ID : " << campaignId << endl;
    cout << " Campaign Discription : " << description << endl;
    cout << endl;

}

Campaign::~Campaign() {

}
```

22.Complaint.h

```
#pragma once
#include <iostream>

using namespace std;

class Complaint
{
private:
	int complaintId;
	string cDescription;

public:
	Complaint();
	Complaint(int cId, string cDes);
	void viewCustomerComplaints();
	~Complaint();
};
```

23.Complaint.cpp

```
#include "Complaint.h"

#include <iostream>

using namespace std;

Complaint::Complaint() {
	complaintId = 0;
	cDescription = "";
}
Complaint::Complaint(int cId, string cDes) {
	complaintId = cId;
	cDescription = cDes;
}
void Complaint::viewCustomerComplaints() {
	cout << endl;
	cout << "Complaint ID : " << complaintId;
	cout << "Complaint Description : " << cDescription;
	cout << endl;
}
Complaint::~Complaint() {

}
```

24.Payment.h

```
#pragma once
#include<iostream>
#include<string>

using namespace std;
#include"Order.h"

class Payment
{
private:
	string paymenttype;
	int paymentid;
	string paymentmethod;
	double Dcharge;
	Order order;
public:
	Payment();
	Payment(string Ptype,int Pid,string Pmethod);
	void setdetails(string Ptype, int Pid, string Pmethod, double
Pdelivery);
	void paymentdetails();
	double getDeliveryCharge() const;
	void paymentcomplaint();
	~Payment();
};
```

25.Payment.cpp

```
#include "Payment.h"
#include"Order.h"
#include<iostream>
using namespace std;


Payment::Payment() {
	paymenttype = "";
	paymentid = 0;
	paymentmethod = "";

}
Payment::Payment(string Ptype, int Pid, string Pmethod) {
	paymenttype = Ptype;
	paymentid = Pid;
	paymentmethod = Pmethod;

}
void Payment::setdetails(string Ptype, int Pid, string Pmethod,
double PDelivery)
{
	paymenttype = Ptype;
	paymentid = Pid;
	paymentmethod = Pmethod;
	Dcharge = PDelivery;
}
void Payment::paymentdetails()
{
	cout << "payment type : " << paymenttype << endl;
	cout << "payment id : " << paymentid << endl;
	cout << "payment method : " << paymentmethod << endl;
	cout << "delivery charge : " << Dcharge << endl;
}
double Payment::getDeliveryCharge() const
{
	return Dcharge;
}

void Payment::paymentcomplaint()
{
	cout << "card OTP validation error" << endl;
}
Payment::~Payment()}
```