Topic : Online Bill and Event Reminder

Group No : MLB_04.01_04

Campus : Malabe

Submission Date : 20/05/2022

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
IT21259098	Dilshan K.H.T	0702793307
IT21257568	Panangala P.A.D.S.S	0770618812
IT21261732	Perera W.A.S.K	0776088263
IT21262104	Lakshani D. M. W. S	0714310688
IT21258626	Munasinghe J.R	0768472460

Table of Content

01. Description of the Requirements

- If the customers need to know the features and process of the system, they can enter the web application and explore the content and the features of the system which are on the home page.
- If they have any issues, they can contact the support service.
- If they are satisfied, they can register to the system by providing the email, username, contact number, country, and password.
- The system stores the user data and auto generates a user ID. It can be used to identify the users uniquely.
- Registered customers can log in to the system with their email and password. The system validates the user login credentials.
- Once the user logged in to the system, the user can add reminders to the upcoming events and bills and select the method to get notifications as via SMS or emails.
- User also can view upcoming events and bills, edit event or bill details, remove events or bills, give feedback to the system, and post issues.
- If the user is satisfied with the system, the user can give ratings.
- When adding new bills or events, the system generates a bill ID or an event ID to identify them uniquely.
- Administrators of the system have to provide name, email, password, and contact number in order to register to the system.
- Admins of the system can log in to the system with their email and password. The system validates the admin login credentials. Admins can be uniquely identified by the admin ID.
- Support service admin is an admin who manages issues. System validates the support service admin code upon login.

- The global events manager is an admin who adds new global events. System validates the global event manager code upon login. If there are outdated events, the global event manager can remove them.
- System admin is an admin who manages web content by removing outdated content, adding advertisements, managing feedback, and checking user accounts. System validates the system admin code upon login.
- If there are inactive accounts, the system admin can alert and notify a user about new features. The system admin can request a report from the database. The database generates the reports when the system admin requests them.
- If a user deactivates their account, all the data inside the user account including added events and bills will be erased from the system database.

02. Noun Verb Analysis

I. Classes

- Customer
- Event
- Bill
- Reminder
- Issues
- Feedback
- Support service admin
- Global event manager
- System admin
- Report
- Admin

II. Methods

- Customer → Customer registration
  Generate user ID
  Log in
  Validate customer details

- Event → View events
  Edit events
  Remove events
  Add new events
  Generate event ID
  Erase events

- Bill → View bills
  Edit bills
  Remove events
  Add new bills
  Generate bill ID
  Erase bills

- Issue → Post issues
  Manage issues

- Feedback → Give feedback
  Give ratings
  Manage feedback

- Support service admin → Contact support service
  Support service admin ID validation

- Global event manager → Global event manager ID validation

- Global event → Search events
  Add new global events
  Remove outdated events

- System admin → System admin ID validation

- Report → Request report
  Generate report

- Admin → Generate admin ID
  Register
  Log in
  Validate admin details

03. CRC Cards

Customer Class	
Responsibilities	**collaborations**
Register customer details	
Generate user ID	
Log in	
Validate customer details	

Event Class	
Responsibilities	**Collaborations**
Add events	
Remove events	
Update event details	
Set event reminder	
Generate event ID	

Global Event Class	
Responsibilities	**Collaborations**
Search global events	
Add new global events	Global event manager
Remove outdated events	Global event manager

Bill Class	
Responsibilities	**Collaborations**
Add bills	
Remove bills	
Update bill details	
Set bill reminder	
Generate bill ID	

Reminder Class	
Responsibilities	**Collaborations**
Set reminder	Bill, Event

Issue Class	
Responsibilities	**Collaborations**
Post issues	Customer
Manage issues	Support service admin

Feedback Class	
Responsibilities	**Collaborations**
Give feedback	Customer
Give ratings	
Manage feedback	

Support Service Admin Class	
Responsibilities	**Collaborations**
Contact support service	
Support service admin code validation	

Global Event Manager Class	
Responsibilities	**Collaborations**
Global event manager code validation	

System admin Class	
Responsibilities	**Collaborations**
System admin code validation	

Report Class	
Responsibilities	**Collaborations**
List of events	event
List of global events	Global event
List of bills	bill
List of registered users	Customer

Admin Class	
Responsibilities	**Collaborations**
Store admin details	
Generate admin ID	
Validate admin details	
Register admin details	

04. Class Diagram

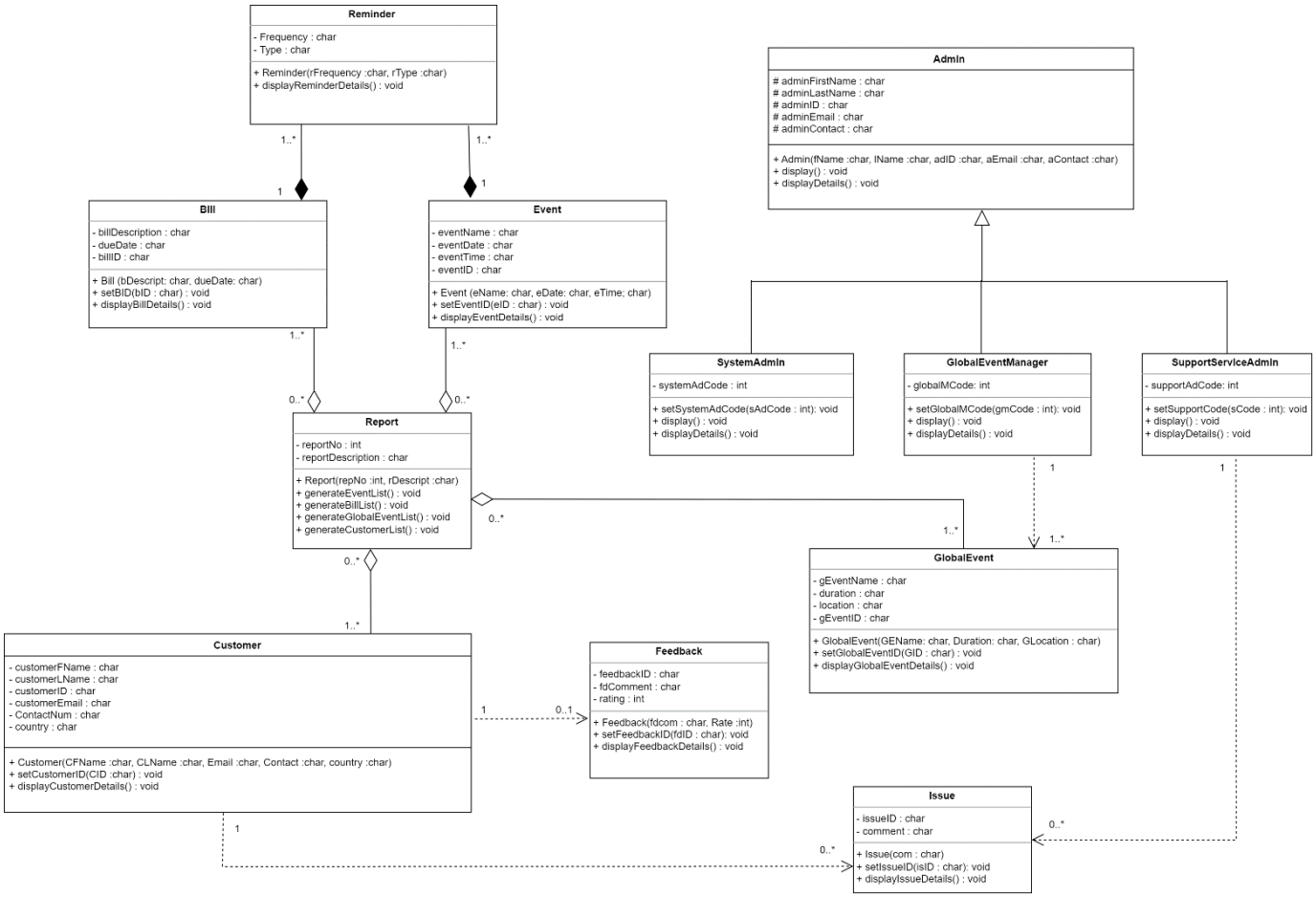

Diagram link :-
https://drive.google.com/file/d/1gzfP3NrtzNYR4h9uLLGORVJ8r_7l9tTy/view?usp=sharing

05. Coding for the Classes

```
#include <iostream>
#include <cstring>
#define SIZE1 2
using namespace std;

//classes implementation

//class feedback
class Feedback {
private:
    char feedbackID[10];
    char fdComment[30];
    int rating;

public:
    Feedback(const char fdcom[], int Rate);
    void setFeedbackID(const char fdID[]);
    void displayFeedbackDetails();
    ~Feedback();

};

//class issue
class Issue
{
private:
    char issueID[10];
    char comment[100];


public:
    Issue(const char com[]);
    void setIssueID(const char isID[]);
    void displayIssueDetails();
    ~Issue();
};

//class customer
class Customer {
private:
    char customerFName[10];
    char customerLName[10];
    char customerID[10];
```

```
    char customerEmail[30];
    char contactnumber[20];
    char country[20];

public:
    Customer(const char CFName[], const char CLName[], const
char Email[], const char Contact[], const char Country[]);
    void setCoustemerID(const char CID[]);
    void displayCustomerDetails();
    void addFeedback(const char CID[], Feedback* f);
    void addIssue(const char CID[], Issue* i);
    ~Customer();
};

//class reminder
class Reminder {
private:
    char Frequency[20];
    char Type[20];
    int rNumber;
public:
    Reminder(int rNo);
    void setFrequency(const char fre[]);
    void setType(const char type[]);
    void displayReminderDetails();
    void Display();
    ~Reminder();
};

//class event
class Event {
private:
    char eventName[20];
    char eventDate[20];
    char eventTime[20];
    char eventID[20];
    Reminder* RM[SIZE1];
public:
    Event();
    Event(const char eName[], const char eDate[], const char
eTime[], int r1, int r2);
    void setEventID(const char eID[]);
    void displayEventDetails();
    void displayEventReminders();
    ~Event();
};

//class bill
class Bill {
private:
```

```
    char billDescription[30];
    char dueDate[20];
    char billID[6];
    Reminder* rm[SIZE1];
public:
    Bill();
    Bill(const char bDescript[], const char dueD[], int R1, int
R2);
    void setBID(const char bID[]);
    void displayBillDetails();
    void displayBillReminders();
    ~Bill();
};

//class globalEvent
class GlobalEvent {
private:
    char gEventName[20];
    char duration[20];
    char location[10];
    char gEventID[10];

public:
    GlobalEvent(const char GEName[], const char Duration[],
const char GLocation[]);
    void setGlobalEventID(const char GID[]);
    void displayGlobalEventDetails();
    ~GlobalEvent();

};

//class report
class Report {
private:
    int reportNo;
    char reportDescription[50];
    Customer* cs[SIZE1];
    Event* eve[SIZE1];
    Bill* bl[SIZE1];
    GlobalEvent* gle[SIZE1];

public:
    Report(int repNo, const char rDescript[]);
    void addCustomerList(Customer* cs1, Customer* cs2);
    void addEventList(Event* eve1, Event* eve2);
    void addBillList(Bill* bl1, Bill* bl2);
    void addGlobalEventList(GlobalEvent* gle1, GlobalEvent*
gle2);
    void displayReportDetails();
```

```
    void displayCustomerList();
    void displayEventList();
    void displayBillList();
    void displayGlobalEventList();
    ~Report();
};

//class admin
class Admin {

protected:
    char adminFirstName[10];
    char adminLastName[10];
    char adminID[10];
    char adminEmail[30];
    char adminContact[20];


public:
    Admin(const char fName[], const char lName[], const char
adID[], const char aEmail[], const char aContact[]);
    void display();
    void displayDetails();

};

//class supportServiceAdmin
class SupportServiceAdmin : public Admin {

private:
    int supportAdminCode;

public:
    SupportServiceAdmin(const char fName[], const char lName[],
const char adID[], const char aEmail[], const char aContact[],
int sCode);
    void display();
    void displayDetails();
    void manageIssues(int sCode, Issue* i);
    ~SupportServiceAdmin();
};

//class systemAdmin
class SystemAdmin : public Admin {

private:
    int systemAdCode;

public:
```

```
    SystemAdmin(const char fName[], const char lName[], const
char adID[], const char aEmail[], const char aContact[], int
sAdCode);
    void display();
    void displayDetails();
    ~SystemAdmin();
};

//class globalEventManager
class GlobalEventManager : public Admin {

private:
    int globalMCode;

public:
    GlobalEventManager(const char fName[], const char lName[],
const char adID[], const char aEmail[], const char aContact[],
int gmCode);
    void display();
    void displayDetails();
    void manageGlobalEvents(int gmCode, GlobalEvent* ge);
    ~GlobalEventManager();
};
```

```
//Function implementations

//customer class functions implementation
Customer::Customer(const char CFName[], const char CLName[],
const char Email[], const char Contact[], const char Country[])
{
    strcpy(customerFName, CFName);
    strcpy(customerLName, CLName);
    strcpy(customerEmail, Email);
    strcpy(contactnumber, Contact);
    strcpy(country, Country);
}
void Customer::setCoustemerID(const char CID[])
{
    strcpy(customerID, CID);
}
void Customer::displayCustomerDetails()
{
    cout << "Customer Name:" << customerFName << " " <<
customerLName << endl;
    cout << "Customer ID:" << customerID << endl;
    cout << "Customer Email:" << customerEmail << endl;
    cout << "Customer Contact Number:" << contactnumber << endl;
    cout << "Customer Country:" << country << endl;
}
Customer::~Customer()
{
    cout << "Customer details deleted" << endl;
}

//issue class functions implementation
Issue::Issue(const char com[])
{
    strcpy(comment, com);
}
void Issue::setIssueID(const char isID[])
{
    strcpy(issueID, isID);
}
void Issue::displayIssueDetails()
{
    cout << "Issue id:" << issueID << endl;
    cout << "Issue comment:" << comment << endl;
}
Issue::~Issue()
{
    cout << "Issue details deleted " << endl;
}
```

```
//Event class functions implementation
Event::Event()
{
    RM[0] = new Reminder(0);
    RM[1] = new Reminder(0);
}
Event::Event(const char eName[], const char eDate[], const char
eTime[], int r1, int r2)
{
    strcpy(eventName, eName);
    strcpy(eventDate, eDate);
    strcpy(eventTime, eTime);
    RM[0] = new Reminder(r1);
    RM[1] = new Reminder(r2);
}

void  Event::displayEventReminders()
{
    for (int i = 0; i < SIZE1; i++)
    {
        RM[i]->Display();
        cout << endl;
    }
}

void Event::setEventID(const char eID[])
{
    strcpy(eventID, eID);
}
void Event::displayEventDetails()
{
    cout << "Event Details" << endl;
    cout << "Event Name : " << eventName << endl;
    cout << "Event Date : " << eventDate << endl;
    cout << "Event Time : " << eventTime << endl;
    cout << "Event ID : " << eventID << endl;
}
Event::~Event()
{
    cout << "Event destructor deleted" << endl;
    for (int i = 0; i < SIZE1; i++)
    {
        delete RM[i];
    }
    cout << "Event and Event reminder details deleted" << endl;

}

//Reminder class functions implementation
Reminder::Reminder(int rNo)
```

```
{
    rNumber = rNo;
}

void Reminder::setFrequency(const char fre[])
{
    strcpy(Frequency, fre);
}

void Reminder::setType(const char type[])
{
    strcpy(Type, type);
}

void Reminder::displayReminderDetails()
{
    cout << "Reminder Details" << endl;
    cout << "Frequency : " << Frequency << endl;
    cout << "Type : " << Type << endl;
}

void Reminder::Display()
{
    cout << "Reminder number : " << rNumber;
}
Reminder::~Reminder()
{
    cout << "Reminder details deleted" << endl;
}

//Bill class functions implementation
Bill::Bill()
{
    rm[0] = new Reminder(0);
    rm[1] = new Reminder(0);
}

Bill::Bill(const char bDescript[], const char dueD[], int R1,
int R2)
{
    strcpy(billDescription, bDescript);
    strcpy(dueDate, dueD);
    rm[0] = new Reminder(R1);
    rm[1] = new Reminder(R2);
}

void  Bill::displayBillReminders()
{
    for (int i = 0; i < SIZE1; i++)
    {
```

```
        rm[i]->Display();
        cout << endl;
    }
}

void Bill::setBID(const char bID[])
{
    strcpy(billID, bID);
}

void Bill::displayBillDetails()
{
    cout << "Bill ID : " << billID << endl;
    cout << "Bill description : " << billDescription << endl;
    cout << "Due date : " << dueDate << endl;
}

Bill::~Bill()
{
    cout << "Bill Destructor was called." << endl;
    for (int i = 0; i < SIZE1; i++)
    {
        delete rm[i];
    }
    cout << "Bill and bill reminders deleted" << endl;
}

//Report class functions implementation
Report::Report(int repNo, const char rDescript[])
{
    reportNo = repNo;
    strcpy(reportDescription, rDescript);
}

void Report::displayReportDetails()
{
    cout << "Report number : " << reportNo << endl;
    cout << "Report description : " << reportDescription <<
endl;
}

void Report::addCustomerList(Customer* cs1, Customer* cs2)
{
    cs[0] = cs1;
    cs[1] = cs2;
}
void Report::addEventList(Event* eve1, Event* eve2)
{
    eve[0] = eve1;
```

```
    eve[1] = eve2;
}
void Report::addBillList(Bill* bl1, Bill* bl2)
{
    bl[0] = bl1;
    bl[1] = bl2;
}
void Report::addGlobalEventList(GlobalEvent* gle1, GlobalEvent*
gle2)
{
    gle[0] = gle1;
    gle[1] = gle2;
}

Report::~Report()
{
    cout << "Report details deleted." << endl;
}

void Report::displayBillList()
{
    for (int i = 0; i < SIZE1; i++)
    {
        bl[i]->displayBillDetails();
    }
}


void Report::displayEventList()
{
    for (int i = 0; i < SIZE1; i++)
    {
        eve[i]->displayEventDetails();
    }
}

void Report::displayCustomerList()
{
    for (int i = 0; i < SIZE1; i++)
    {
        cs[i]->displayCustomerDetails();
    }
}

void Report::displayGlobalEventList()
{
    for (int i = 0; i < SIZE1; i++)
    {
        gle[i]->displayGlobalEventDetails();
    }
```

```
}

//Global event class functions implementation
GlobalEvent::GlobalEvent(const char GEName[], const char
Duration[], const char GLocation[]) {
    strcpy(gEventName, GEName);
    strcpy(duration, Duration);
    strcpy(location, GLocation);

}

void GlobalEvent::setGlobalEventID(const char GID[]) {
    strcpy(gEventID, GID);

}

void GlobalEvent::displayGlobalEventDetails() {
    cout << "Global Event details " << endl << endl;
    cout << "Event ID :" << gEventID << endl;
    cout << "Event Name :" << gEventName << endl;
    cout << "Duration :" << duration << endl;
    cout << "Location :" << location << endl;
}

GlobalEvent::~GlobalEvent() {
    cout << "Global event details deleted" << endl;
}

//Feedback class functions implementation
Feedback::Feedback(const char fdcom[], int Rate) {
    strcpy(fdComment, fdcom);
    rating = Rate;
}

void Feedback::setFeedbackID(const char fdID[]) {
    strcpy(feedbackID, fdID);
}

void Feedback::displayFeedbackDetails() {
    cout << "Feedback details" << endl << endl;
    cout << "Feedback Id :" << feedbackID << endl;
    cout << "Comment :" << fdComment << endl;
    cout << "Rating :" << rating << endl;
}


Feedback::~Feedback() {
    cout << "Feedback details deleted" << endl;
}
```

```
//Admin class functions implementation
Admin::Admin(const char fName[], const char lName[], const char
adID[], const char aEmail[], const char aContact[])
{
    strcpy(adminFirstName, fName);
    strcpy(adminLastName, lName);
    strcpy(adminID, adID);
    strcpy(adminEmail, aEmail);
    strcpy(adminContact, aContact);
}

void Admin::display()
{
    cout << "Admin class called" << endl;
    cout << endl;
}

void Admin::displayDetails()
{
    cout << "Admin details" << endl;
    cout << "Admin ID : " << adminID << endl;
    cout << "Admin name : " << adminFirstName << "" <<
adminLastName << endl;
    cout << "Email : " << adminEmail << endl;
    cout << "Contact : " << adminContact << endl;
    cout << endl;
}

SupportServiceAdmin::SupportServiceAdmin(const char fName[],
const char lName[], const char adID[], const char aEmail[],
const char aContact[], int sCode) : Admin(fName, lName, adID,
aEmail, aContact)
{
    strcpy(adminFirstName, fName);
    strcpy(adminLastName, lName);
    strcpy(adminID, adID);
    strcpy(adminEmail, aEmail);
    strcpy(adminContact, aContact);
    supportAdminCode = sCode;
}

void SupportServiceAdmin::display()
{
    cout << "Support service admin class called" << endl;
    cout << endl;
}

void SupportServiceAdmin::displayDetails()
{
```

```
    cout << "Support service admin details" << endl;
    cout << "Admin ID : " << adminID << endl;
    cout << "Admin name : " << adminFirstName << "" <<
adminLastName << endl;
    cout << "Email : " << adminEmail << endl;
    cout << "Contact : " << adminContact << endl;
    cout << "Support service admin code : " << supportAdminCode
<< endl;
    cout << endl;
}


SupportServiceAdmin :: ~SupportServiceAdmin()
{
    cout << "Support service details deleted" << endl;
}

SystemAdmin::SystemAdmin(const char fName[], const char lName[],
const char adID[], const char aEmail[], const char aContact[],
int sAdCode) : Admin(fName, lName, adID, aEmail, aContact)
{
    strcpy(adminFirstName, fName);
    strcpy(adminLastName, lName);
    strcpy(adminID, adID);
    strcpy(adminEmail, aEmail);
    strcpy(adminContact, aContact);
    systemAdCode = sAdCode;
}

void SystemAdmin::display()
{
    cout << "System admin class called" << endl;
    cout << endl;
}

void SystemAdmin::displayDetails()
{
    cout << "System admin details" << endl;
    cout << "Admin ID : " << adminID << endl;
    cout << "Admin name : " << adminFirstName << "" <<
adminLastName << endl;
    cout << "Email : " << adminEmail << endl;
    cout << "Contact : " << adminContact << endl;
    cout << "System admin code : " << systemAdCode << endl;
    cout << endl;
}

SystemAdmin :: ~SystemAdmin()
{
    cout << "System admin details deleted" << endl;
```

```
}


GlobalEventManager::GlobalEventManager(const char fName[], const
char lName[], const char adID[], const char aEmail[], const char
aContact[], int gmCode) : Admin(fName, lName, adID, aEmail,
aContact)
{
    strcpy(adminFirstName, fName);
    strcpy(adminLastName, lName);
    strcpy(adminID, adID);
    strcpy(adminEmail, aEmail);
    strcpy(adminContact, aContact);
    globalMCode = gmCode;
}

void GlobalEventManager::display()
{
    cout << "Global event manager class called" << endl;

}

void GlobalEventManager::displayDetails()
{
    cout << "Global event manager details" << endl;
    cout << "Admin ID : " << adminID << endl;
}

GlobalEventManager :: ~GlobalEventManager()
{
    cout << "Global event manager details deleted" << endl;
}
```

```
//main programme

int main()
{
    cout << endl;
    cout << "=====================================" << endl;
    cout << endl;

    //customer object creation
    Customer* cus1 = new Customer("Amal", "Mendis",
"amal@123gmail.com", "0715422386", "Sri Lanka");

    cus1->setCoustemerID("C001");
    cus1->displayCustomerDetails();

    cout << endl;
    cout << "=====================================" << endl;
    cout << endl;

    //issue object creation
    Issue* issue1 = new Issue("System is not responding.");
    issue1->setIssueID("I001");
    issue1->displayIssueDetails();

    cout << endl;
    cout << "=====================================" << endl;
    cout << endl;
    //Event object creation
    Event* ev = new  Event("Holy", "2022-08-01", "15:30", 003,
004);
    ev->setEventID("545");
    ev->displayEventDetails();
    ev->displayEventReminders();
    cout << endl;
    cout << "=====================================" << endl;
    cout << endl;

    //Bill object creation
    Bill* bill = new Bill("This is a water bill.", "2022-06-10",
001, 002);
    bill->setBID("B005");
    bill->displayBillDetails();
    bill->displayBillReminders();
    cout << endl;
    cout << "=====================================" << endl;
    cout << endl;

    //Report object creation
```

```
    Report* rep = new Report(555, "This is a report about global
events.");

    rep->displayReportDetails();

    cout << endl;
    cout << "=====================================" << endl;
    cout << endl;

    //global event object creation
    GlobalEvent* ge1 = new GlobalEvent("Holi Festival", "10.00AM
to 12.00PM", "Colombo");
    ge1->setGlobalEventID("001");
    ge1->displayGlobalEventDetails();

    cout << endl;
    cout << "=====================================" << endl;
    cout << endl;

    //feedback object creation
    Feedback* fb1 = new Feedback("Very user friendly system :)",
5);
    fb1->setFeedbackID("fd001");
    fb1->displayFeedbackDetails();

    cout << endl;
    cout << "=====================================" << endl;
    cout << endl;

    //Support service admin object creation
    SupportServiceAdmin* ss1 = new
SupportServiceAdmin("Sunil", "Perera", "AD001",
"Sunil123@gmail.com", "0784569855", 001);
    ss1->display();
    ss1->displayDetails();

    cout << endl;
    cout << "=====================================" << endl;
    cout << endl;

    //system admin object creation
    SystemAdmin* sa1 = new         SystemAdmin("Amal", "Anjana",
"AD002", "Amal456@gmail.com", "0784586955", 001);
    sa1->display();
    sa1->displayDetails(); //Display system admin details

    cout << endl;
    cout << "=====================================" << endl;
    cout << endl;
```

```
    //Global event manager object creation
    GlobalEventManager* gm1 = new GlobalEventManager("kamal",
"Randiv", "AD003", "kamal1@gmail.com", "0774586965", 001);
    gm1->display();
    gm1->displayDetails(); //Display global manager details

    cout << endl;
    cout << "=======================================" << endl;
    cout << endl;

    //calling destructors
    delete fb1;
    delete ge1;
    delete rep;
    delete bill;
    delete ev;
    delete issue1;
    delete cus1;
    delete ss1;
    delete sa1;
    delete gm1;

    cout << endl;
    cout << "=======================================" << endl;
    cout << endl;

    return 0;
}
```

06. Individual contributions

IT21259098 K.H.T Dilshan	Feedback class Global event class
IT21257568 Panangala P.A.D.S.S	System admin class Global event manager class Support service admin class
IT21261732 Perera W.A.S.K	Customer class Issue class
IT21262104 Lakshani D. M. W. S	Report class Bill class
IT21258626 Munasinghe J.R	Event class Reminder class