THE GLOBAL EVENT MANAGEMENT SYSTEM LIMITED.

//Durga Sutharsan - 101285544

//Jorge De La Cruz - 101248516

//Juan Consuegra - 101216670

//Saloni Jagdishbhai Prajapati - 101283741

//Program class

using System;

using System.Collections.Generic;

using System.Text;

using System.Text.RegularExpressions;

namespace assignment2

{

class Program

{

static void Main(string[] args)

{

//creates a new object in EventCoordinator class (parameters includes seeds and capacity for event and customer class)

eCoord = new EventCoordinator(200, 1000, 101, 5000);

runProgram();

Console.WriteLine("Thank you for using Transformers Event Management Limited System");

Console.WriteLine("Press any key to exit.");

Console.ReadKey();

}

static EventCoordinator eCoord;

public static void deleteCustomer()

{

int id;

Console.Clear();

Console.WriteLine(eCoord.customerList());

Console.Write("Please enter a customer id to delete:");

id = getIntChoice();

if (eCoord.deleteCustomer(id))

{

Console.WriteLine("Customer with id {0} deleted..", id);

}

else

{

Console.WriteLine("Customer with id {0} was not found..", id);

}

Console.WriteLine("\nPress any key to continue return to the main menu.");

Console.ReadKey();

}

public static void viewCustomers()

{

Console.Clear();

Console.WriteLine(eCoord.customerList());

Console.WriteLine("\nPress any key to continue return to the main menu.");

Console.ReadKey();

}

public static void viewSpecificCustomer()

{

int id;

string cust;

Console.Clear();

Console.WriteLine(eCoord.customerList());

Console.Write("Please enter a customer id to View:");

id = getIntChoice();

Console.Clear();

cust = eCoord.getCustomerInfoById(id);

Console.WriteLine(cust);

Console.WriteLine("\nPress any key to continue return to the previous menu.");

Console.ReadKey();

}

public static void addCustomer()

{

string fname, lname, phone;

Console.Clear();

Console.WriteLine("-----------Add Customer----------");

Console.Write("Please enter the customer's first name:");

fname = IsAlphabets();

Console.Write("Please enter the customer's last name:");

lname = IsAlphabets();

Console.Write("Please enter the customer's phone:");

phone = formatPhoneNumber();

if (eCoord.addCustomer(fname, lname, phone))

{

Console.WriteLine("Customer successfully added..");

}

else

{

Console.WriteLine("Customer was not added..");

}

Console.WriteLine("\nPress any key to continue return to the main menu.");

Console.ReadKey();

}

public static void addEvent()

{

string eventName, venue;

Date eventDate;

int maxAttendees;

int day, month, year, hour, minute;

Console.Clear();

Console.WriteLine("-----------Add Event----------");

Console.Write("Please enter the name of the Event:");

eventName = Console.ReadLine();

Console.Write("Please enter venue for the event:");

venue = Console.ReadLine();

Console.Write("Please enter the Day of the event:");

day = getIntChoice();

Console.Write("Please enter the Month of the event (as an integer):");

month = getIntChoice();

Console.Write("Please enter the Year of the event:");

year = getIntChoice();

Console.Write("Please enter the Hour the event starts in 24 hour format:");

hour = getIntChoice();

Console.Write("Please enter the Minute the event starts:");

minute = getIntChoice();

eventDate = new Date(day, month, year, hour, minute);

Console.Write("Please enter the maximum number of attendees:");

maxAttendees = getIntChoice();

if (eCoord.addEvent(eventName, venue, eventDate, maxAttendees))

{

Console.WriteLine("Event successfully added..");

}

else

{

Console.WriteLine("The event was not added..");

}

Console.WriteLine("\nPress any key to continue return to the main menu.");

Console.ReadKey();

}

public static void viewEvents()

{

Console.Clear();

Console.WriteLine(eCoord.eventList());

Console.WriteLine("\nPress any key to continue return to the main menu.");

Console.ReadKey();

}

public static void viewSpecificEvent()

{

int id;

string ev;

Console.Clear();

Console.WriteLine(eCoord.eventList());

Console.Write("Please enter an event id to View:");

id = getIntChoice();

Console.Clear();

ev = eCoord.getEventInfoById(id);

Console.WriteLine(ev);

Console.WriteLine("\nPress any key to continue return to the previous menu.");

Console.ReadKey();

}

public static string customerMenu()

{

string s = "Transformers Event Management Limited\n";

s += "Customer Menu.\n";

s += "Please select a choice from the menu below:\n";

s += "1: Add Customer \n";

s += "2: View Customers \n";

s += "3: View Customer Details \n";

s += "4: Delete Customer\n";

s += "5: Return to the main menu.";

return s;

}

public static string eventMenu()

{

string s = "Transformers Event Management Limited\n";

s += "Event Menu.\n";

s += "Please select a choice from the menu below:\n";

s += "1: Add Event \n";

s += "2: View all Events \n";

s += "3: View Event Details \n";

s += "4: Return to the main menu.";

return s;

}

public static string registrationMenu()

{

string s = "Transformers Event Management Limited\n";

s += "Event Registration Menu.\n";

s += "Please select a choice from the menu below:\n";

s += "1: RSVP for event \n";

s += "2: View RSVPs \n";

s += "3: Return to the main menu.";

return s;

}

public static string mainMenu()

{

string s = "Transformers Event Management Limited.\n";

s += "Please select a choice from the menu below:\n";

s += "1: Customer Options \n";

s += "2: Event Options \n";

s += "3: RSVP for Event \n";

s += "4: Exit";

return s;

}

public static void runCustomerMenu()

{

string menu = customerMenu();

int choice = getValidChoice(5, menu);

while (choice != 5)

{

if (choice == 1) { addCustomer(); }

if (choice == 2) { viewCustomers(); }

if (choice == 3) { viewSpecificCustomer(); }

if (choice == 4) { deleteCustomer(); }

choice = getValidChoice(5, menu);

}

}

public static void runEventMenu()

{

string menu = eventMenu();

int choice = getValidChoice(4, menu);

while (choice != 4)

{

if (choice == 1) { addEvent(); }

if (choice == 2) { viewEvents(); }

if (choice == 3) { viewSpecificEvent(); }

choice = getValidChoice(4, menu);

}

}

public static void runRegistrationMenu()

{

string menu = registrationMenu();

int choice = getValidChoice(3, menu);

while (choice != 3)

{

if (choice == 1) { makeRsvp(); }

if (choice == 2) { viewRsvpList(); }

choice = getValidChoice(3, menu);

}

}

public static int getValidChoice(int max, string menu)

{

int choice;

Console.Clear();

Console.WriteLine(menu);

while (!int.TryParse(Console.ReadLine(), out choice) || (choice < 1 || choice > max))

{

Console.Clear();

Console.WriteLine(menu);

Console.WriteLine("Please enter a valid choice:");

}

return choice;

}

public static int getIntChoice()

{

int choice;

while (!int.TryParse(Console.ReadLine(), out choice))

{

Console.WriteLine("Please enter an integer:");

}

return choice;

}

public static void runProgram()

{

string menu = mainMenu();

int choice = getValidChoice(4, menu);

while (choice != 4)

{

if (choice == 1) { runCustomerMenu(); }

if (choice == 2) { runEventMenu(); }

if (choice == 3) { runRegistrationMenu(); }

choice = getValidChoice(4, menu);

}

}

//method to make RSVP for event

public static void makeRsvp()

{

Console.Clear();

Console.WriteLine(eCoord.customerList()); //displays the list of customers

Console.WriteLine("Enter a customer Id:");

int custId = Convert.ToInt32(Console.ReadLine()); //accepts customer id from user

if (eCoord.customerExist(custId)) //checks if the customer with that id exists

{

Console.WriteLine(eCoord.eventList()); //displays the list of events

Console.WriteLine("Please enter event id");

int eId = Convert.ToInt32(Console.ReadLine()); //accepts event id from user

if (eCoord.eventExist(eId)) //checks if the event with that id exists

{

if (eCoord.roomAvailable(eId)) //checks if there is a room available for the customer or not

{

eCoord.getEvent(eId).addAttendee(eCoord.getCustomer(custId)); //adds the customer into attendee's list

eCoord.makingRsvp(eCoord.getCustomer(custId).getFirstName() + " " + eCoord.getCustomer(custId).getLastName(), eId); //generates a string that confirms the RSVP

Console.WriteLine("Your registration is created successfully!");

}

else

{

Console.WriteLine("This Event is full!");

}

}

else

{

Console.WriteLine("This Event does not exist!");

}

}

else

{

Console.WriteLine("This customer does not exist!");

}

Console.WriteLine("\nPress any key to continue return to the main menu.");

Console.ReadKey();

}

//method to view RSVP list

public static void viewRsvpList()

{

Console.Clear();

Console.WriteLine(eCoord.getRsvpList());

Console.WriteLine("\nPress any key to continue return to the main menu.");

Console.ReadKey();

}

//Validates if the input string is alphabets and space only

public static string IsAlphabets()

{

string value = Console.ReadLine();

Regex r = new Regex("^[a-zA-Z ]+$");

while (!r.IsMatch(value))

{

Console.WriteLine("Please use alphabets and space only: ");

value = Console.ReadLine();

}

return value;

}

//validates if the input string is valid as a phone number

public static string formatPhoneNumber()

{

string ph = Console.ReadLine();

//checks if it's null or contains any non-digits

while(string.IsNullOrWhiteSpace(ph) || !Regex.IsMatch(ph, @"^[\d\-]+$") || Regex.IsMatch(ph, @"^[a-zA-Z]") || ph.Length < 10 || ph.Length > 12)

{

Console.WriteLine("Enter in the format of ###-###-####");

ph = Console.ReadLine();

}

//checks if it's in the format: ###-###-####

if (Regex.IsMatch(ph, @"^(\d{3}-\d{3}-\d{4})$"))

{

//returned if correct

return ph;

}

if (ph.Length == 10)

{

return string.Format("{0:###-###-####}", double.Parse(ph));

}

return ph;

}

}

}

//EventCoordinator class

using System;

using System.Collections.Generic;

using System.Text;

namespace assignment2

{

class EventCoordinator

{

CustomerManager custMan;

EventManager eventMan;

RsvpManager rvManager;

public EventCoordinator(int custIdSeed, int maxCust, int eventIdSeed, int maxEvents)

{

custMan = new CustomerManager(custIdSeed, maxCust);

eventMan = new EventManager(eventIdSeed, maxEvents);

rvManager = new RsvpManager(maxCust);

}

//customer related methods

public bool addCustomer(string fname, string lname, string phone)

{

return custMan.addCustomer(fname, lname, phone);

}

public string customerList()

{

return custMan.getCustomerList();

}

public string getCustomerInfoById(int id)

{

return custMan.getCustomerInfo(id);

}

public bool deleteCustomer(int id)

{

return custMan.deleteCustomer(id);

}

public Customer getCustomer(int cid)

{

return custMan.getCustomer(cid);

}

// Event related methods

public bool addEvent(string name, string venue, Date eventDate, int maxAttendee)

{

return eventMan.addEvent(name, venue, eventDate, maxAttendee);

}

public string eventList()

{

return eventMan.getEventList();

}

public string getEventInfoById(int id)

{

return eventMan.getEventInfo(id);

}

public bool customerExist(int cid)

{

return custMan.customerExist(cid);

}

public bool roomAvailable(int eventId)

{

return eventMan.roomAvailable(eventId);

}

public Event getEvent(int eid)

{

return eventMan.getEvent(eid);

}

public bool eventExist(int id)

{

return eventMan.eventExists(id);

}

// rsvp related

public bool makingRsvp(string cName, int eventId)

{

return rvManager.makingRsvp(cName, eventId);

}

public int getRsvp(int bNum)

{

return rvManager.getRsvp(bNum).getRsvpNum();

}

public string getRsvpList()

{

return rvManager.getRsvpList();

}

}

}

//Customer class

using System;

using System.Collections.Generic;

using System.Text;

namespace assignment2

{

class Customer

{

private int customerId;

private string firstName;

private string lastName;

private string phone;

private int bookings;

public Customer(int cId, string fname, string lname, string ph)

{

bookings = 0;

customerId = cId;

firstName = fname;

lastName = lname;

phone = ph;

}

public int getId() { return customerId; }

public string getFirstName() { return firstName; }

public string getLastName() { return lastName; }

public string getPhone() { return phone; }

public int getNumBookings() { return bookings; }

public override string ToString()

{

string s = "Customer " + customerId;

s = s + "\nName: " + firstName + " " + lastName;

s = s + "\nPhone: " + phone;

s = s + "\nBookings: " + bookings;

return s;

}

}

}

//CustomerManager class

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace assignment2

{

class CustomerManager

{

private static int currentCustNumber;

private int maxNumCustomers;

private int numCustomers;

private Customer[] customerList;

public CustomerManager(int ccn, int max)

{

currentCustNumber = ccn;

maxNumCustomers = max;

numCustomers = 0;

customerList = new Customer[maxNumCustomers];

}

public bool addCustomer(string fn, string ln, string ph)

{

if (numCustomers >= maxNumCustomers) { return false; }

Customer c = new Customer(currentCustNumber, fn, ln, ph);

currentCustNumber++;

customerList[numCustomers] = c;

numCustomers++;

return true;

}

public int findCustomer(int cid)

{

for (int x = 0; x < numCustomers; x++)

{

if (customerList[x].getId() == cid)

return x;

}

return -1;

}

public bool customerExist(int cid)

{

int loc = findCustomer(cid);

if (loc == -1) { return false; }

return true;

}

public Customer getCustomer(int cid)

{

int loc = findCustomer(cid);

if (loc == -1) { return null; }

return customerList[loc];

}

public string getCustomerInfo(int cid)

{

int loc = findCustomer(cid);

if (loc == -1) { return "There is no customer with id " + cid + "."; }

return customerList[loc].ToString();

}

public bool deleteCustomer(int cid)

{

int loc = findCustomer(cid);

if (loc == -1)

{

return false;

}

customerList = customerList.Where(x => x.getId() != cid).ToArray();

numCustomers--;

return true;

}

public string getCustomerList()

{

string s = "Customer List:";

s = s + "\nNumber \t Name \t \t Phone";

for (int x = 0; x < numCustomers; x++)

{

s = s + "\n" + customerList[x].getId() + "\t" + customerList[x].getFirstName() + "\t" + customerList[x].getLastName() + "\t" + customerList[x].getPhone();

}

return s;

}

}

}

//Event class

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace oopassignment2gui

{

public class Event

{

private int eventId;

private string eventName;

private string venue;

private Date eventDate;

private int maxAttendees;

private int numAttendees;

private string dateCreated;

private Customer[] attendeeList;

public Event(int eventId, string name, string venue, Date eventDate, int maxAttendees)

{

this.eventId = eventId;

this.eventName = name;

this.venue = venue;

this.eventDate = eventDate;

this.maxAttendees = maxAttendees;

numAttendees = 0;

attendeeList = new Customer[maxAttendees];

dateCreated = DateTime.Now.ToString(@"MM\/dd\/yyyy h\:mm tt");

}

public int getEventId() { return eventId; }

public string getEventName() { return eventName; }

public string getVenue() { return venue; }

public Date getDate() { return eventDate; }

public int getMaxAttendees() { return maxAttendees; }

public int getNumAttendees() { return numAttendees; }

//method to add a customer to the list of attendee once RSVP is done

public bool addAttendee(Customer c)

{

if (numAttendees >= maxAttendees) { return false; }

attendeeList[numAttendees] = c;

int curBook = c.getNumBookings();

curBook++;

c.setNumBookings(curBook);

numAttendees++;

return true;

}

//method to find attendee from the list

private int findAttendee(int custId)

{

for (int x = 0; x < maxAttendees; x++)

{

if (attendeeList[x].getId() == custId)

return x;

}

return -1;

}

//method to remove attendee from the list

public bool removeAttendee(int custId)

{

int loc = findAttendee(custId);

if (loc == -1) return false;

attendeeList[loc] = attendeeList[numAttendees - 1];

numAttendees--;

return true;

}

//method to get a list of attendees

public string getAttendees()

{

string s = "\nCustomers registered :";

for (int x = 0; x < numAttendees; x++)

{

s = s + "\n" + (x + 1) + ". " + attendeeList[x].getFirstName() + " " + attendeeList[x].getLastName();

}

return s;

}

public override string ToString()

{

string s = "Event: " + eventId;

s = s + "\nName: " + eventName;

s = s + "\nVenue: " + venue;

s = s + "\nDate: " + eventDate;

s = s + "\nRegistered Attendees: " + numAttendees;

s = s + "\nAvailable spaces: " + (maxAttendees - numAttendees);

s = s + getAttendees();

return s;

}

}

}

//EventManager class

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace oopassignment2gui

{

public class EventManager

{

private static int currentEventId;

private int maxEvents;

private int numEvents;

private Event[] eventList;

public EventManager(int idSeed, int max)

{

currentEventId = idSeed;

maxEvents = max;

numEvents = 0;

eventList = new Event[maxEvents];

}

public bool addEvent(string name, string venue, Date eventDate, int maxAttendees)

{

if (numEvents >= maxEvents) { return false; }

for (int x = 0; x < numEvents; x++)

{

if (eventList[x].getVenue().ToLower().Equals(venue.ToLower()) && eventList[x].getDate().day == eventDate.day && eventList[x].getDate().month == eventDate.month && eventList[x].getDate().year == eventDate.year)

{

return false;

}

}

Event e = new Event(currentEventId, name, venue, eventDate, maxAttendees);

eventList[numEvents] = e;

numEvents++;

currentEventId++;

return true;

}

private int findEvent(int eid)

{

for (int x = 0; x < numEvents; x++)

{

if (eventList[x].getEventId() == eid)

return x;

}

return -1;

}

public bool eventExists(int eid)

{

int loc = findEvent(eid);

if (loc == -1) { return false; }

return true;

}

public Event getEvent(int eid)

{

int loc = findEvent(eid);

if (loc == -1) { return null; }

return eventList[loc];

}

public bool deleteEvent(int eid)

{

int loc = findEvent(eid);

if (loc == -1) { return false; }

eventList[loc] = eventList[numEvents - 1];

numEvents--;

return true;

}

public string getEventInfo(int eid)

{

int loc = findEvent(eid);

if (loc == -1) { return "There is no event with id " + eid + "."; }

return eventList[loc].ToString();

}

public string getEventList()

{

string s = "Event List: ";

s += "\nEvent Id \t Event Name \t Venue";

for (int x = 0; x < numEvents; x++)

{

s = s + "\n" + eventList[x].getEventId() + " \t " + eventList[x].getEventName() + " \t " + eventList[x].getVenue();

}

return s;

}

//checks if there is a room for customer in the event

public bool roomAvailable(int id)

{

Event x = getEvent(id);

if (x.getMaxAttendees() > x.getNumAttendees())

{

return true;

}

else

{

return false;

}

}

}

}

//Rsvp class

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace oopassignment2gui

{

public class Rsvp

{

private int rsvpNumber;

private string rsvpDate;

private string customerName;

private int eventId;

//constructor for RSVP class

public Rsvp(int rsvpNum, string cName, int idNum)

{

rsvpNumber = rsvpNum;

rsvpDate = DateTime.Now.ToString(@"MM/dd/yyyy h:mm tt");

customerName = cName;

eventId = idNum;

}

//getters and setters

public int getRsvpNum()

{

return rsvpNumber;

}

public string getRsvpDate()

{

return rsvpDate;

}

public string getCustomerName()

{

return customerName;

}

public int getEventId()

{

return eventId;

}

//toString method for RSVP class

public string toString()

{

string s = "Rsvp Numer: " + rsvpNumber;

s = s + "\nRsvp Date: " + rsvpDate;

s = s + "\nCustomer Name: " + customerName;

s = s + "\nEvent Id: " + eventId;

return s;

}

}

}

//RsvpManager class

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace oopassignment2gui

{

public class RsvpManager

{

private int idSee;

private int numCustomer;

private int maxCustomers;

private Rsvp[] RsvpList;

public RsvpManager(int max)

{

idSee = 1;

numCustomer = 0;

maxCustomers = max;

RsvpList = new Rsvp[maxCustomers];

}

//method to find RSVP for customer using rsvpId

public int findRsvp(int rNum)

{

for (int i = 1; i < numCustomer; i++)

{

if (RsvpList[i].getRsvpNum() == i)

{

return i;

}

}

return -1;

}

//method to get rsvp for customer using rsvpId

public Rsvp getRsvp(int rNum)

{

int num = findRsvp(rNum);

if (num == -1) { return null; }

return RsvpList[num];

}

//method to create RSVP for customer

public bool makingRsvp(string cName, int eventId)

{

if (numCustomer >= maxCustomers) { return false; }

{

Rsvp temp = new Rsvp(idSee, cName, eventId);

RsvpList[numCustomer] = temp;

numCustomer++;

idSee++;

return true;

}

}

//generates a list of RSVPs

public string getRsvpList()

{

string s = "Rsvp List: " + "\nRsvp Number \tDate \t Customer Name \t Event Id";

for (int x = 0; x < numCustomer; x++)

{

s = s + "\n" + RsvpList[x].getRsvpNum() + " \t" + RsvpList[x].getRsvpDate()

+ " \t " + RsvpList[x].getCustomerName() + " \t " + RsvpList[x].getEventId();

}

return s;

}

}

}

//Date class

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace oopassignment2gui

{

public class Date

{

public int year;

public int month; // 1 Jan, 2, Feb....

public int day; // no error checking required for day

public int hour; //24 hour format

public int minute; //

public Date(int day, int month, int year, int hour, int minute)

{

this.day = day;

this.year = year;

this.month = month;

this.hour = hour;

this.minute = minute;

}

public string viewLongMonth()

{

switch (month)

{

case 1:

return "January";

case 2:

return "February";

case 3:

return "March";

case 4:

return "April";

case 5:

return "May";

case 6:

return "June";

case 7:

return "July";

case 8:

return "August";

case 9:

return "September";

case 10:

return "October";

case 11:

return "November";

case 12:

return "December"; ;

default:

return "-";

}

}

public string viewShortMonth()

{

switch (month)

{

case 1:

return "Jan";

case 2:

return "Feb";

case 3:

return "Mar";

case 4:

return "Apr";

case 5:

return "May";

case 6:

return "Jun";

case 7:

return "July";

case 8:

return "Aug";

case 9:

return "Sep";

case 10:

return "Oct";

case 11:

return "Nov";

case 12:

return "Dec"; ;

default:

return "-";

}

}

public override string ToString()

{

string s = day + " " + viewShortMonth() + " " + year;

s += " at " + hour + ":" + minute;

return s;

}

}

}