Advanced Object Oriented Programming Final Assignment

(Group size : 4 people)

Due: **Friday** 4th December 2020 at 11:30 pm on Blackboard

Group Members:

ALEKSANDR KUDIN – 101258693

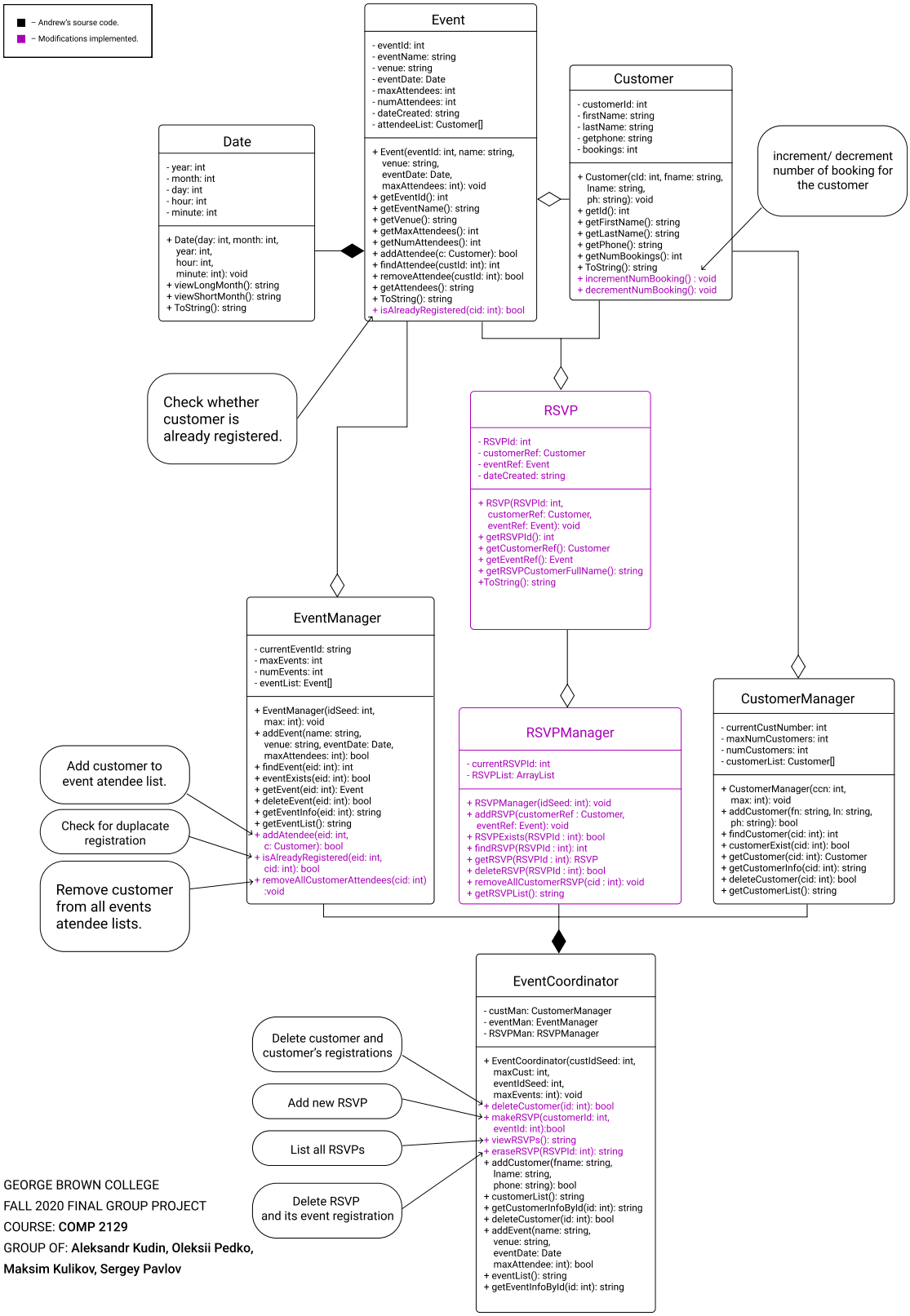
OLEKSII PEDKO – 101242285

SERGEY PAVLOV – 101288444

MAKSIM KULIKOV – 101278070

There are 2 archives in the submission: GUI version and CLI version.

FALL 2020 FINAL GROUP PROJECT – UML CLASS DIAGRAM



GEORGE BROWN COLLEGE
FALL 2020 FINAL GROUP PROJECT
COURSE: **COMP 2129**
GROUP OF: **Aleksandr Kudin, Oleksii Pedko, Maksim Kulikov, Sergey Pavlov**

Date Class

```
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)
   {
      if (day <= 0) { day = 1; }
      if (month <= 0) { month = 1; }
      if (day > 31) { day = 31; }
      if (month > 12) { month = 12; }
      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;
   }
}
```

Customer Class

```
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 void incrementNumBooking() { bookings++; } // MODIFICATION (ALEKSANDR KUDIN)
    public void decrementNumBooking() { bookings--; }
    public override string ToString()
    {
        string s = "Customer " + customerId;
        s = s + "\nName: " + firstName + " " + lastName;
        s = s + "\nPhone: " + phone;
        s = s + "\nBookings: " + bookings;

        return s;
    }

}
```

Event Class

```
class Event
  {
    private int eventId;
    private string eventName;
    private string venue;
    private Date eventDate;
    private int maxAttendees;
    private int numAttendees;
    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];
    }

    public int getEventId() { return eventId; }
```

```
public string getEventName() { return eventName; }
public string getVenue() { return venue; }

public int getMaxAttendees() { return maxAttendees; }
public int getNumAttendees() { return numAttendees; }

public bool addAttendee(Customer c)
{
   if (numAttendees >= maxAttendees) { return false; }
   attendeeList[numAttendees] = c;
   numAttendees++;
   return true;
}

private int findAttendee(int custId)
{
   for (int x = 0; x < numAttendees; x++)
   {
      if (attendeeList[x].getId() == custId)
         return x;
   }
   return -1;
}

public bool removeAttendee(int custId)
```

```
{
   int loc = findAttendee(custId);
   if (loc == -1) return false;
   attendeeList[loc] = attendeeList[numAttendees - 1];
   numAttendees--;
   return true;
}

public string getAttendees()
{
   string s = "\nCustomers registered :";
   for (int x = 0; x < numAttendees; x++)
   {
      s = s + "\n" + attendeeList[x].getFirstName() + " " + attendeeList[x].getLastName();
   }
   return s;
}

public override string ToString()
{
   string s = "Event: " + eventId + "\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;
    }

    // MODIFICATION (OLEKSII PEDKO) – IMPLEMENTED METHODS: isAlreadyRegistered().

    public bool isAlreadyRegistered(int cid)
    {
        if (numAttendees == 0) { return false; }
        if (findAttendee(cid) == -1) { return false; }
        return true;
    }

}
```

RSVP Class

```
    class RSVP
    {
        int RSVPId;
        Customer customerRef;
        Event eventRef;
        string dateCreated;

        public RSVP(int RSVPId, Customer customerRef, Event eventRef)
        {
            this.RSVPId = RSVPId;
            this.customerRef = customerRef;
            this.eventRef = eventRef;
            this.dateCreated = DateTime.Now.ToString(@"MM\/dd\/yyyy h\:mm tt");
        }

        public int getRSVPId() { return RSVPId; }
        public Customer getCustomerRef() { return customerRef; }
        public Event getEventRef() { return eventRef; }
        public string getRSVPCustomerFullName() { return customerRef.getFirstName() + " " +
customerRef.getLastName(); }

        public override string ToString()
        {
            string s = "RSVP ID: " + RSVPId + " | Customer: " + getRSVPCustomerFullName() + "
| Event ID: " + eventRef.getEventId() + " | Date Created: " + dateCreated.ToString();
            return s;
        }
    }
```

CustomerManager Class

```
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[loc] = customerList[numCustomers - 1];
        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;
}
```

```
}
```

EventManager Class

```
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; }
        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:";
            for (int x = 0; x < numEvents; x++)
            {
                s = s + "\n" + eventList[x].getEventId() + " \t " +
eventList[x].getEventName() + " \t " + eventList[x].getVenue();
            }
            return s;
        }

        //MODIFICATION (ALEKSANDR KUDIN) – IMPLEMENTED METHODS: addAtendee()

        public bool addAtendee(int eid, Customer c) // register customer for the event and
check if the event is not full.
        {
                return getEvent(eid).addAttendee(c);
        }

        // MODIFICATION (OLEKSII PEDKO) – IMPLEMENTED METHODS: isAlreadyRegistered()

        public bool isAlreadyRegistered(int eid, int cid) // check whether customer is already
registered or not.
        {
            return getEvent(eid).isAlreadyRegistered(cid);
        }

        // MODIFICATION (ALEKSANDR KUDIN) – IMPLEMENTED METHODS: removeAllCustomerAttendees()

        public void removeAllCustomerAttendees(int cid) // removing all customer's attendees.
        {
            for (int i =  0;  i < numEvents; i++)
            {
                eventList[i].removeAttendee(cid);
            }
        }

    }
```

RSVPManager Class

```
class RSVPManager
{
    private static int currentRSVPId;
    private ArrayList RSVPList = new ArrayList();

    public RSVPManager(int idSeed)
    {

        currentRSVPId = idSeed;
    }

    //MODIFICATION (ALEKSANDR KUDIN) – IMPLEMENTED METHODS: addRSVP().

    public void addRSVP(Customer customerRef, Event eventRef)
    {
        RSVP temp = new RSVP(currentRSVPId, customerRef, eventRef);
        RSVPList.Add(temp);
        currentRSVPId++;
    }

    //MODIFICATION (ALEKSANDR KUDIN) – IMPLEMENTED METHODS: RSVPExists().

    public bool RSVPExists(int RSVPId)
    {
        for (int i = 0; i < RSVPList.Count; i++)
        {
            RSVP temp = (RSVP)RSVPList[i];
            if (temp.getRSVPId() == RSVPId)
            {
                return true;
            }
        }
        return false;
    }

    //MODIFICATION (ALEKSANDR KUDIN) – IMPLEMENTED METHODS: findRSVP().

    private int findRSVP(int RSVPId)
    {
        for (int i = 0; i < RSVPList.Count; i++)
        {
            RSVP temp = (RSVP)RSVPList[i];
            if (temp.getRSVPId() == RSVPId)
                return i;
        }
        return -1;
    }

    //MODIFICATION (ALEKSANDR KUDIN) – IMPLEMENTED METHODS: getRSVP().

    public RSVP getRSVP(int RSVPId)
    {
        int loc = findRSVP(RSVPId);
```

```
            if (loc == -1) { return null; }
            return (RSVP)RSVPList[loc];
        }

        //MODIFICATION (ALEKSANDR KUDIN) – IMPLEMENTED METHODS: deleteRSVP().

        public bool deleteRSVP(int RSVPId)
        {
            int loc = findRSVP(RSVPId);
            if (loc == -1) { return false; }
            RSVPList.RemoveAt(loc);
            return true;
        }

        //MODIFICATION (MAKSIM KULIKOV) – IMPLEMENTED METHODS: getRSVPList().

        public string getRSVPList()
        {
            string s = "RSVP List:\n\n";

            foreach (RSVP RSVP in RSVPList)
            {
                s += RSVP.ToString() + "\n";
            }

            return s;
        }

        //MODIFICATION (SERGEY PAVLOV) – IMPLEMENTED METHODS: removeAllCustomerRSVP().

        public void removeAllCustomerRSVP(int cid) // removing all customer's RSVPs (removing
from the array backwards).
        {
            for (int i = RSVPList.Count - 1; i >= 0; i--)
            {
                RSVP temp = (RSVP)RSVPList[i];
                if (temp.getCustomerRef().getId() == cid)
                {
                    RSVPList.RemoveAt(i);
                }
            }
        }
    }
```

EventCoordinator Class

```
    class EventCoordinator
    {
        CustomerManager custMan;
        EventManager eventMan;
        RSVPManager RSVPMan;


        public EventCoordinator(int custIdSeed, int maxCust, int eventIdSeed, int maxEvents,
int RSVPIdSeed)
        {
            custMan = new CustomerManager(custIdSeed, maxCust);
            eventMan = new EventManager(eventIdSeed, maxEvents);
            RSVPMan = new RSVPManager(RSVPIdSeed);
        }

        // 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)
        {
            if(custMan.customerExist(id))
            {
                RSVPMan.removeAllCustomerRSVP(id); // MODIFICATION (SERGEY PAVLOV) – removing
all RSVPs asociated with the customer.
                eventMan.removeAllCustomerAttendees(id); // MODIFICATION (ALEKSANDR KUDIN) –
removing all customer's attendees from events.
                custMan.deleteCustomer(id);
                return true;
            }
            return false;
        }

        // 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);
        }

        // MODIFICATION (ALEKSANDR KUDIN) – IMPLEMENTED METHODS: makeRSVP().
        // MODIFICATION (MAKSIM KULIKOV) – IMPLEMENTED METHODS: viewRSVPs().
        // MODIFICATION (ALEKSANDR KUDIN) – IMPLEMENTED METHODS: eraseRSVP().

        public string makeRSVP(int eid, int cid)
        {
            if (!eventMan.eventExists(eid)) { return "There is no event with id " + eid + ".";
} // event id check.
            if (!custMan.customerExist(cid)) { return "There is no customer with id " + cid +
"."; } // customer id check.
            Customer c = custMan.getCustomer(cid); // puts customer reference in the variable.
            string customerFullName = custMan.getCustomer(cid).getFirstName() + " " +
custMan.getCustomer(cid).getLastName(); // puts full name of the customer into the
customerFullName variable.
            if (eventMan.isAlreadyRegistered(eid, cid)) { return customerFullName + " is
already registered for the event with id " + eid + "."; } // duplicate check.
            if (!eventMan.addAtendee(eid, c)) { return "Event with id " + eid + " is full."; }
// event space avaliability check -> register customer for the event.
            custMan.getCustomer(cid).incrementNumBooking(); // incrementing number of booking
for the customer.
            RSVPMan.addRSVP(c, eventMan.getEvent(eid)); // add a record of RSVP to RSVP list.
            return customerFullName + " is registered for the event with id " + eid + "."; //
display the successful message to the user
        }

        public string viewRSVPs()
        {
            return RSVPMan.getRSVPList();
        }

        public string eraseRSVP(int RSVPId)
        {
            if (!RSVPMan.RSVPExists(RSVPId)) { return "There is no RSVP with id " + RSVPId +
"."; }
            RSVPMan.getRSVP(RSVPId).getCustomerRef().decrementNumBooking(); // decrement
number of booking for the castomer which RSVP is deleted.

RSVPMan.getRSVP(RSVPId).getEventRef().removeAttendee(RSVPMan.getRSVP(RSVPId).getCustomerRef().
getId()); // Remove customer registration from the event.
            if (!RSVPMan.deleteRSVP(RSVPId)) { return "Error occured. This RSVP with ID " +
RSVPId + " can not be deleted"; } // handling error. should not appear.
            return "RSVP with ID " + RSVPId + " has been deleted";


        }

    }
```

Program Class

```
// ALEKSANDR KUDIN 101258693
// OLEKSII PEDKO 101242285
// SERGEY PAVLOV 101288444
// MAKSIM KULIKOV 101278070
class Program
{
   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 = Console.ReadLine();
    Console.Write("Please enter the customer's last name:");
    lname = Console.ReadLine();
    Console.Write("Please enter the customer's phone:");
    phone = Console.ReadLine();
    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();
}

//MODIFICATION (ALEKSANDR KUDIN) – IMPLEMENTED METHODS ( makeRSVP() )
//MODIFICATION (ALEKSANDR KUDIN) – IMPLEMENTED METHODS ( viewRSVPs() )
//MODIFICATION (ALEKSANDR KUDIN) – IMPLEMENTED METHODS ( eraseRSVP() )

public static void makeRSVP()
{
   int eventId, customerId;
   Console.Clear();
   Console.WriteLine(eCoord.eventList() + "\n");
```

```
    Console.WriteLine(eCoord.customerList() + "\n");
    Console.WriteLine("-----------Make RSVP----------");
    Console.Write("Please enter an event id to make a RSVP:");
    eventId = getIntChoice();
    Console.Write("Please enter a customer id make a RSVP:");
    customerId = getIntChoice();
    Console.WriteLine(eCoord.makeRSVP(eventId, customerId));
    Console.WriteLine("\nPress any key to continue return to the main menu.");
    Console.ReadKey();
}

public static void viewRSVPs()
{
    Console.Clear();
    Console.WriteLine(eCoord.viewRSVPs());
    Console.WriteLine("\nPress any key to continue return to the main menu.");
    Console.ReadKey();
}

public static void eraseRSVP()
{
    int RSVPId;
    Console.Clear();
    Console.WriteLine("-----------Erasing RSVP----------");
    Console.WriteLine(eCoord.viewRSVPs() + "\n");
```

```
    Console.Write("Please enter RSVP id to erase:");
    RSVPId = getIntChoice();
    Console.WriteLine(eCoord.eraseRSVP(RSVPId));
    Console.WriteLine("\nPress any key to continue return to the main menu.");
    Console.ReadKey();

}

public static string customerMenu()
{
    string s = "Andrew's Modified 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 = "Andrew's Modified 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 = "Andrew's Modified 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: Erase RSVP \n";
    s += "4: Return to the main menu.";
    return s;
}

public static string mainMenu()
{
    string s = "Andrew's Modified 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(4, menu);
        while (choice != 4)
        {
            if (choice == 1) { makeRSVP(); } // MODIFICATION (ALEKSANDR KUDIN) – CALLING makeRSVP() METHOD
            if (choice == 2) { viewRSVPs(); } // MODIFICATION (ALEKSANDR KUDIN) – CALLING viewRSVPs() METHOD
            if (choice == 3) { eraseRSVP(); } // MODIFICATION (ALEKSANDR KUDIN) – CALLING eraseRSVP() METHOD

            choice = getValidChoice(4, 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) || choice < 0)
   {
      if (choice <= 0) { Console.WriteLine("Integer must be positive:"); }
      else { 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);
   }
}

static void Main(string[] args)
{
   eCoord = new EventCoordinator(200, 1000, 101, 5000, 1);
   Date d1 = new Date(07, 07, 2022, 13, 00);
   Date d2 = new Date(15, 02, 2020, 10, 00);
   Date d3 = new Date(25, 12, 2020, 18, 00);
   eCoord.addEvent("Convocation Ceremony", "GBC St. James Campus", d1, 100);
   eCoord.addEvent(".NET Worshop", "GBC Waterfront Campus", d2, 20);
   eCoord.addEvent("Christmas", "City Hall", d3, 100);
   eCoord.addCustomer("Aleksandr", "Kudin", "+1 000 000 0000");
```

```
            eCoord.addCustomer("Oleksii", "Pedko", "+1 000 000 0001");
            eCoord.addCustomer("Maksim", "Kulikov", "+1 000 000 0002");
            eCoord.addCustomer("Sergey", "Pavlov", "+1 000 000 0003");
            eCoord.addCustomer("Andrew", "Rudder", "+1 000 000 0004");
            eCoord.addCustomer("Customer", "Test", "+1 000 000 0005");
            runProgram();
            Console.WriteLine("Thank you for using Andrew's Modified Event Management Limited System.
");
            Console.WriteLine("Press any key to exit.");
            Console.ReadKey();
        }


    }
```