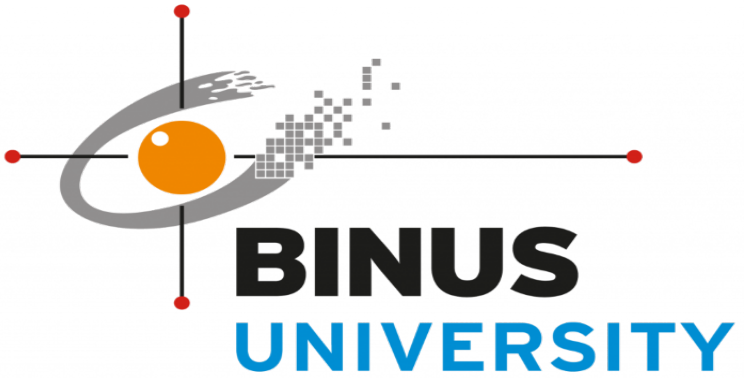
Object_Oriented Project

Event Management System Report



Lecturer:
Jude J.L. Martinez

By: Abas Iman - 2602192853

BINUS UNIVERSITY INTERNATIONAL
2023

Table of Contents

Project Description

The Event Management System (EMS) is a Java-based CLI-application developed to streamline and automate the process of organizing and managing events. It provides a user-friendly CLI interface for participants and administrators to efficiently handle various aspects of event planning, registration, and reservation of different event types. This is a broad system that can work for movie events, sports watching events, and other types of events. Users can easily create, track, and update event details, manage participant registrations. The system aims to enhance the overall event management experience by centralizing information and reducing manual efforts.

Object-Oriented Designs/OOP used

Total of 12 classes have been used in both the management and the user section of the project.

.Inheritance: The following class inherits properties from the class Event.

```
class ReservedEvent extends Event {
    private String location;

    public ReservedEvent(String eventname, String seattype, String location) {
        super(eventname, seattype);
        this.location = location;
    }

    public String getLocation() {
        return location;
    }
```

.**Encapsulation:**

Objects declared private and getter and setter methods have been
Used in order to access those objects/variables

```
private String eventname;
```

```
    private String seattype;

    public Event(String eventname, String seattype) {
        this.eventname = eventname;
        this.seattype = seattype;
    }

    public String getEvent() {
        return eventname;
    }

    public String getSeat() {
        return seattype;
    }
```

.**Polymorphism:**
overriding different methods: example getevent() method is used in the Event class()

```
 for (ReservedEvent newseats : reservelist) {
                                          LinkedList<ReservedEvent>   seatslist   =
hashlist.get(newseats.getEvent());
                if (seatslist == null) {
                    seatslist = new LinkedList<>();
                    hashlist.put(newseats.getEvent(), seatslist);
                }
                seatslist.add(newseats);
            }
```

. **Abstract class**
Class been declared as an abstract class where it cannot be instantiated on its own and serves as a blueprint for two other classes.

```
abstract class Event {
    private String eventname;
    private String seattype;

    public Event(String eventname, String seattype) {
        this.eventname = eventname;
        this.seattype = seattype;
    }

    public String getEvent() {
        return eventname;
    }

    public String getSeat() {
        return seattype;
    }
```

```
}
```

Modularity:

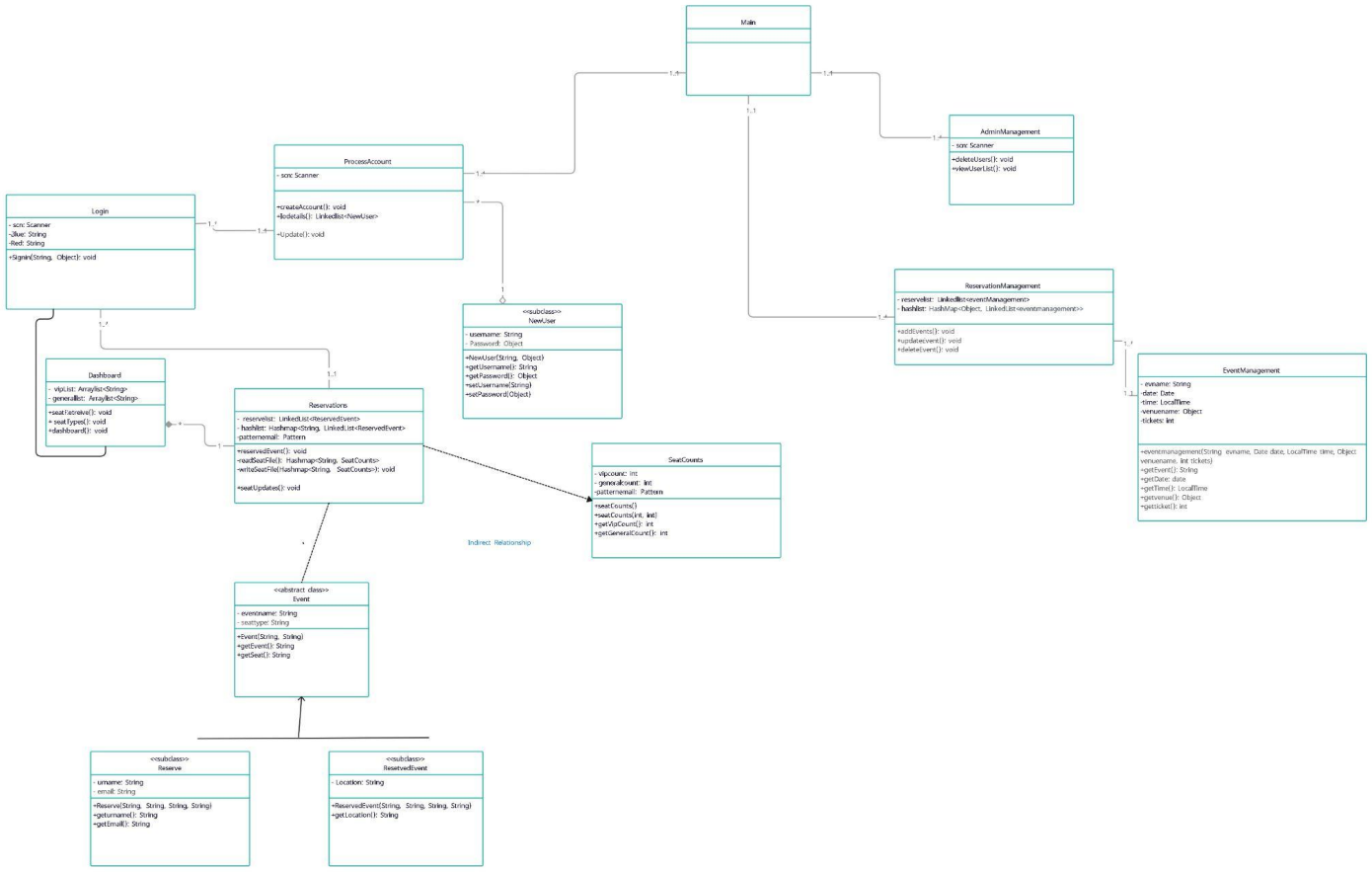
The code is organized into separate classes, each responsible for a specific aspect of the reservation system. The classes Reservations, Event, Reserve, ReservedEvent, and SeatCounts represent modular units with well-defined responsibilities.

Constructor Overloading: The classes event and SeatCounts demonstrate constructor overloading by providing multiple constructors, with different parameter lists and this allows flexibility when creating instances of these classes

Database Used:

I used a file system(I/O) to store data. I made use of the Java API of BufferedWrite to write the date into the file and BufferedReader to read data from the file. The bufferedreader is also very useful when reading from the file as I use a while loop to iterate through all the content in the file and retrieve the desidered item from the file content.

Solution Design (Class Diagram)



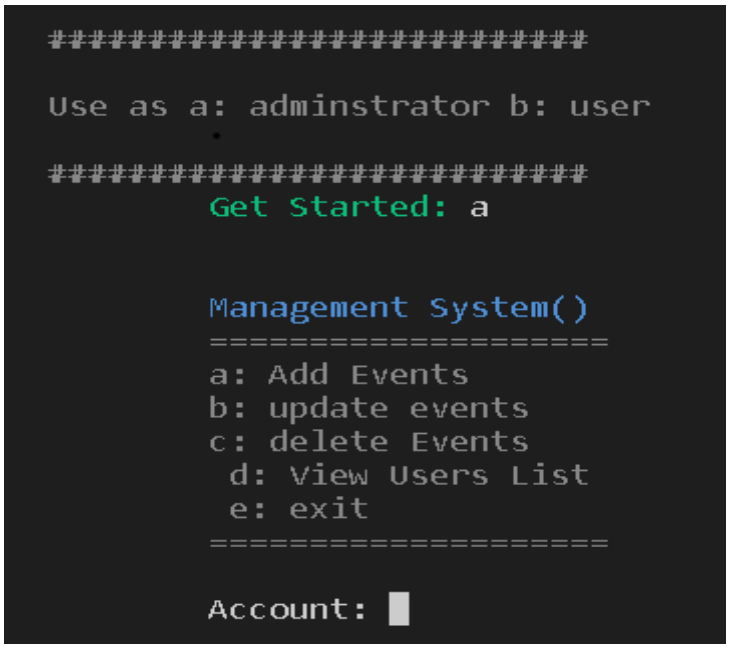
There are a total 13 classes that are used to implement this program, and as can be seen in the above diagram, the relationship between most of the classes is association class relationship, two have composition relationship while dependency and aggregation relationships among classes exist as well. There is one abstract class that's used as a blueprint for other classes.

Discussion of Implementation

Event management system was implemented using command line interface in the following steps:

The landing page of the program

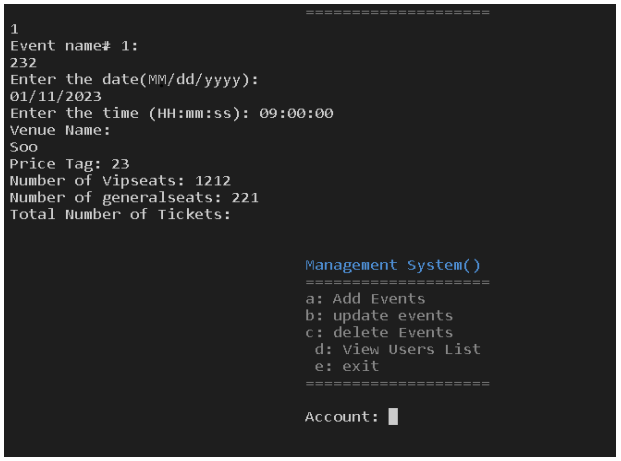
a.



The user will land on this page where they have the option to run the program as either administrator or user. If a: administrator is selected, that means the user will land on the next page which is the Management system or the administration page. The options in the administration page includes adding events and the general management of the program such as viewing users, updating events and so on.

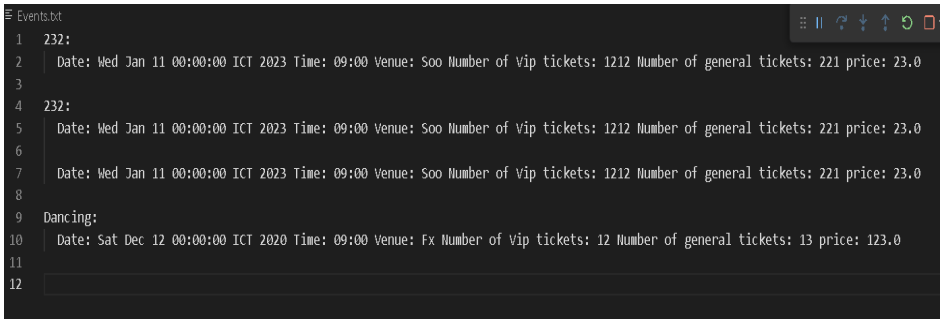
If a: add events is selected:

b.

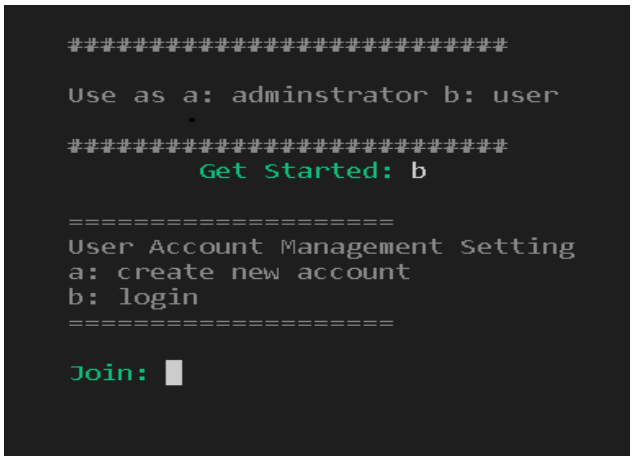


This page is the page that allows the administrator to add new events to their system, so that those events are accessible to the users. Those events will be saved in database (File system file(I/O):

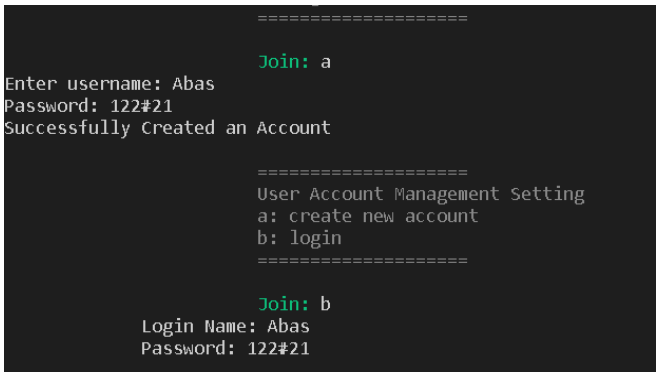
C. this is the file in which the events have been saved (Event.txt).



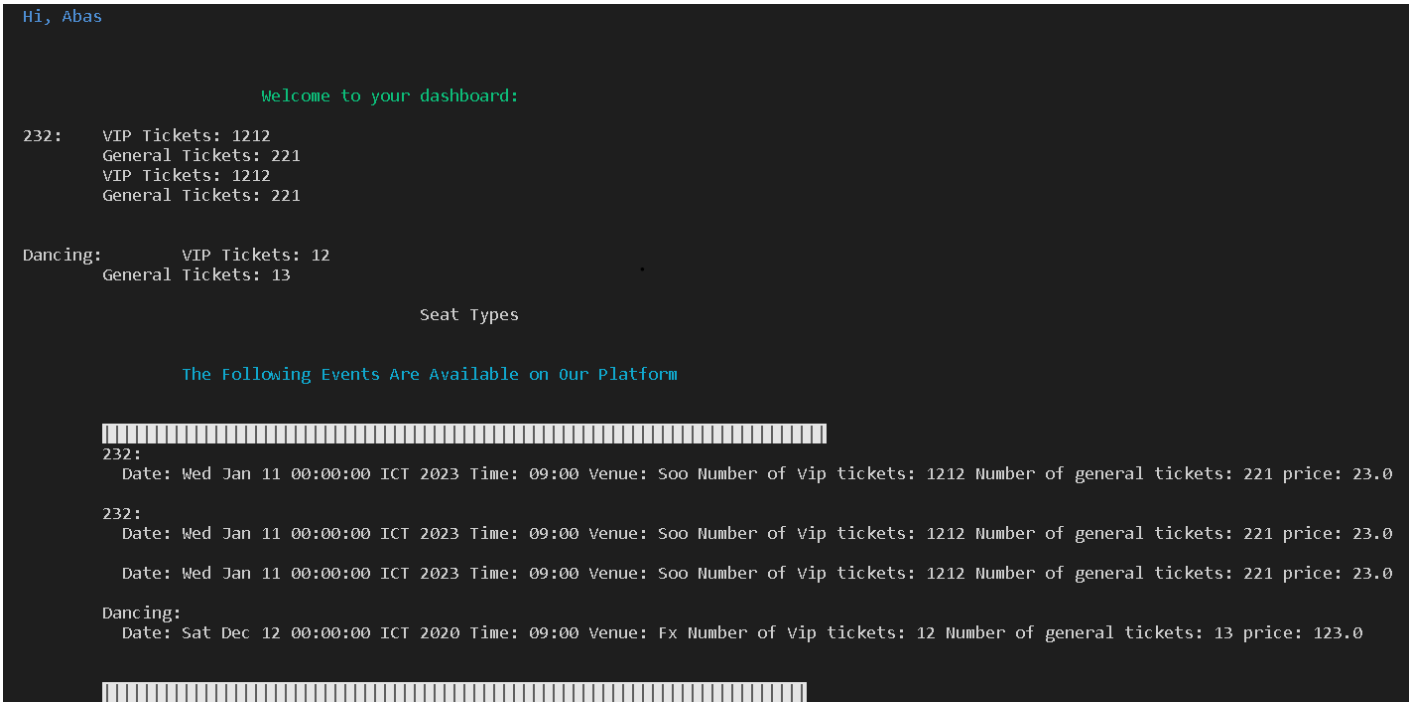
D. The other important section is the participant section where the participants of events can reserve seats either VIP or general seats.



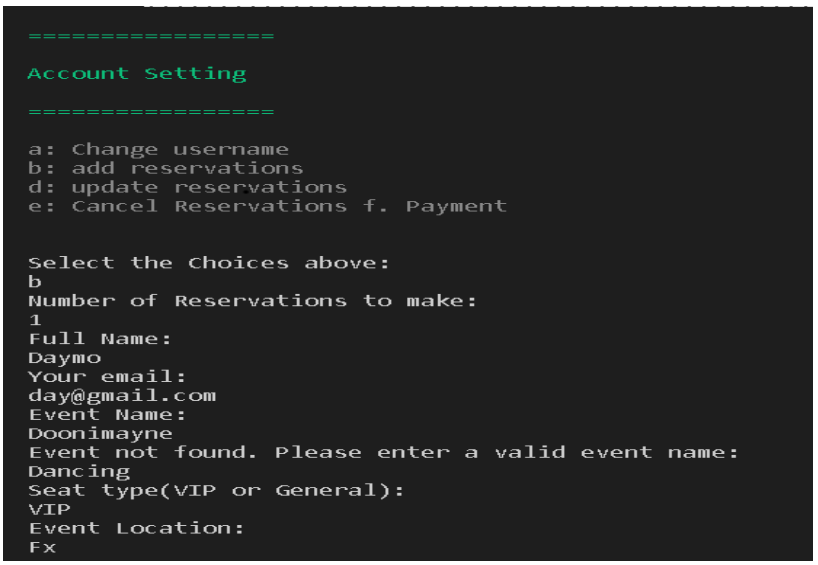
The first step the user should take is to either create an account or login if they have an account.



Once the user login they will land on their dashboard where they can automatically see the available events and the number of seats reserved so far for each category: VIP or General



The users have access to a lot of features including booking events, updating events, and deleting events.



The user books an event and the event which was created by the administrator has now been written into a new file called reservations.txt.

totaltickets.txt

```
232:
VIP Tickets: 1212
General Tickets: 221
VIP Tickets: 1212
General Tickets: 221

Dancing:
VIP Tickets: 12
General Tickets: 13

```

Notable Code Snippets

This is the method that tracks reserved seats and their categories:

```
 HashMap<String, SeatCounts> seatCountsMap = readSeatCountsFromFile();

            for (String eventName : hashlist.keySet()) {
                LinkedList<ReservedEvent> lisSeats = hashlist.get(eventName);

                   SeatCounts existingCounts = seatCountsMap.getOrDefault(eventName,
new SeatCounts());

                int evpCount = existingCounts.getVipCount();
                int generalCount = existingCounts.getGeneralCount();

                for (ReservedEvent reserveEvent : lisSeats) {

                    if (reserveEvent.getSeat().equalsIgnoreCase("VIP")) {
                        evpCount++;
                      } else if (reserveEvent.getSeat().equalsIgnoreCase("General"))
{
                        generalCount++;
                    }
                }

                // Update the seat counts in the seatCountsMap
                            seatCountsMap.put(eventName,  new  SeatCounts(evpCount,
generalCount));
            }
```

.It uses hashmap to store the Event name as the key of the date and the Seat type as the value of the map, and it counts each type as either generalcount or VIP count.

The following is the search method which instead of using data structure uses bufferedreader to search the events from the file:

```
while ((line = reader.readLine()) != null) {
```

```
                if (!line.isEmpty()) {
                    eventBuilder.append(line).append("\n");
                } else {
                    String event = eventBuilder.toString();
                    if (event.contains(categoryName)) {
                        int categoryStart = event.indexOf(categoryName);
                           int priceStart = event.indexOf("price:", categoryStart) +
7;
                        int priceEnd = event.indexOf("\n", priceStart);
                                  String priceString = event.substring(priceStart,
priceEnd).trim();
                        double price = Double.parseDouble(priceString);
                        foundEvents.add(categoryName + ": " + price);
                    }
```

Data Structure Used

- **Hashmap**: I used hashmap to store the event names as the key and the associated content for each event as the value of the hashmap. So that the data can be searched by the key which the event name and the associated data can be retrieved by just searching the event name.
- **Linkedlist:** I used linked list to store the data information and pass the linkedlist as the value for the hashmap.

- **ArrayList:** I used arraylist to store the retrieved data from the file in the search method so that I can print the retrieved data as arraylist.