|  |
| --- |
| Mohamed Ashraf 24P0024  Ahmed Hesham 24P0029  Faris Mohamed 24P0233 |

|  |
| --- |
| Ain Shams University, Faculty of Engineering |
| Event Management System |
| To: Eng/ Mazen |

Table of Contents

[**Class Diagram** 1](#_Toc196338772)

[**Detailed Description** 2](#_Toc196338773)

[ **Database (abstract)** 2](#_Toc196338774)

[**Data Fields:** 2](#_Toc196338775)

[**Functions:** 2](#_Toc196338776)

[ **DataType (Enum)** 4](#_Toc196338777)

[**Data Fields:** 4](#_Toc196338778)

[**Functions:** 4](#_Toc196338779)

[ **User (Abstract)** 4](#_Toc196338780)

[**Data Fields:** 4](#_Toc196338781)

[**Functions:** 4](#_Toc196338782)

[ **Gender (Enum)** 4](#_Toc196338783)

[**Data Fields:** 4](#_Toc196338784)

[**Functions:** 4](#_Toc196338785)

[ **Admin (extends User)** 4](#_Toc196338786)

[**Data Fields:** 4](#_Toc196338787)

[**Functions:** 5](#_Toc196338788)

[ **Organizer (extends User)** 5](#_Toc196338789)

[**Data Fields:** 5](#_Toc196338790)

[**Functions:** 5](#_Toc196338791)

[ **Attendee (extends User)** 7](#_Toc196338792)

[**Data Fields:** 7](#_Toc196338793)

[**Functions:** 7](#_Toc196338794)

[ **Room** 7](#_Toc196338795)

[**Data Fields:** 7](#_Toc196338796)

[**Functions:** 7](#_Toc196338797)

[ **Event** 8](#_Toc196338798)

[**Data Fields:** 8](#_Toc196338799)

[**Functions:** 8](#_Toc196338800)

[ **Status (Enum)** 9](#_Toc196338801)

[ **Category** 9](#_Toc196338802)

[**Data Fields:** 9](#_Toc196338803)

[**Functions:** 9](#_Toc196338804)

[ **Wallet** 9](#_Toc196338805)

[**Data Fields:** 9](#_Toc196338806)

[**Functions:** 9](#_Toc196338807)

[ **TimeRange** 9](#_Toc196338808)

[**Data Fields:** 9](#_Toc196338809)

[**Functions:** 10](#_Toc196338810)

[ **Hours** 10](#_Toc196338811)

[**Data Fields:** 10](#_Toc196338812)

[**Functions:** 10](#_Toc196338813)

[ **Day (Enum)** 11](#_Toc196338814)

[**Data Fields:** 11](#_Toc196338815)

[**Functions:** 11](#_Toc196338816)

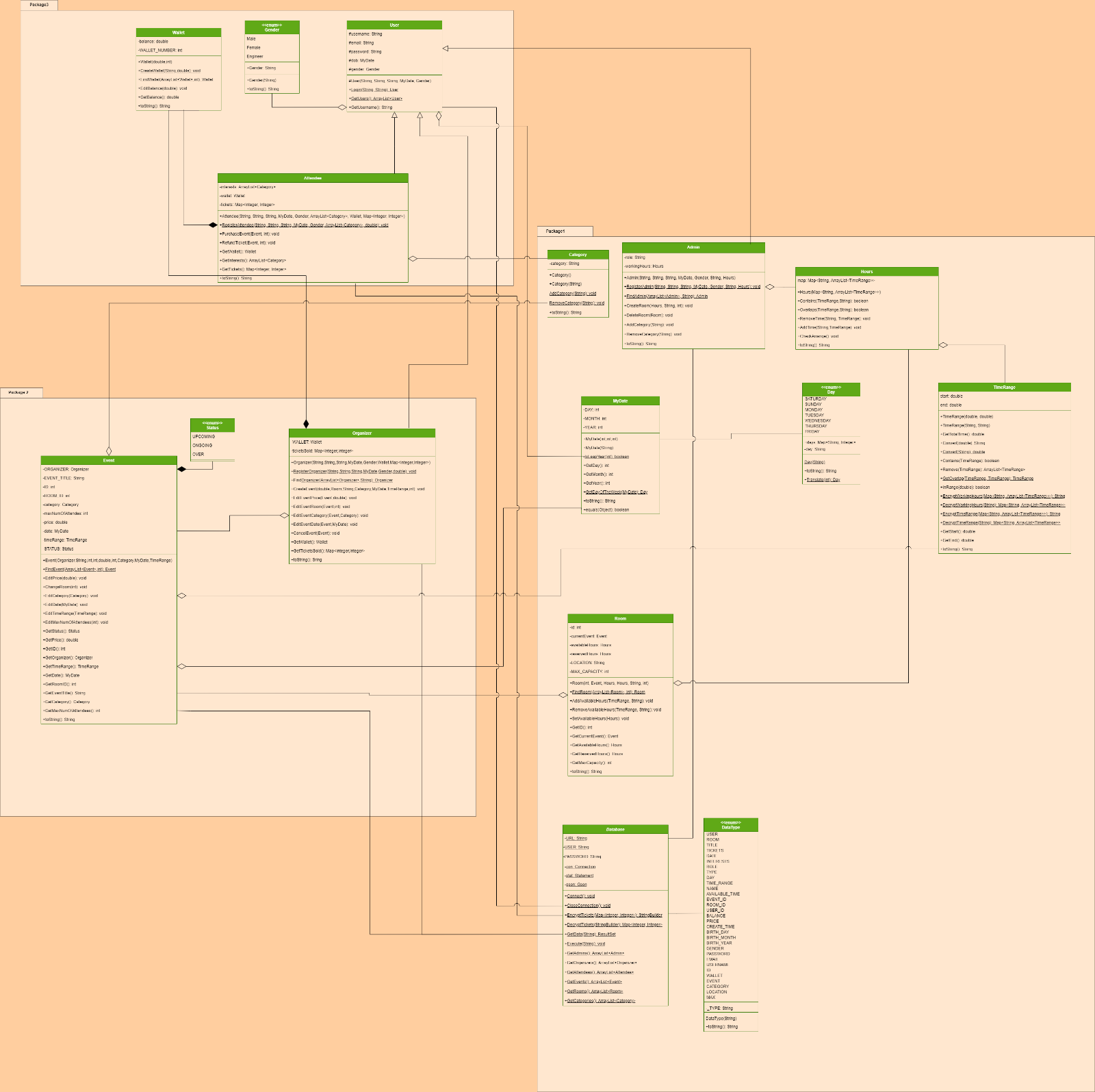
[ **MyDate** 11](#_Toc196338817)

[**Data Fields:** 11](#_Toc196338818)

[**Functions:** 11](#_Toc196338819)

[**GitHub Link** 12](#_Toc196338820)

# **Class Diagram**

****

# **Detailed Description**

This application is an Event Management System, made with OOP principles such as encapsulation and inheritance, and designed to store the data of users with different types (Admins, Organizers, Attendees), and store the data of rooms and events and allow for Admins to add rooms, and Organizers to add events, and Attendees to purchase tickets to those events. The System is designed to be safe, scalable and maintainable.

Required Modules: javafx.controls, javafx.fxml, org.controlsfx.controls, com.dlsc.formsfx, com.google.gson, java.sql

External Dependencies:

From Maven: com.google.code.gson

From JAR: mysqlconnector-j-9.3.0 (provided in the GitHub)

* **Database (abstract)**

### **Data Fields:**

* Private static final String URL: Holds the url of the mysql database.
* Private static final String USER: Holds the user of the mysql database.
* Private static final String PASSWORD: Holds the password of the mysql database.
* Private static Connection con: Holds the connection to the mysql database that is initialized when starting the app.
* Private static Statement stat: Holds the statement for the mysql database to execute commands.
* Private static Gson gson: Initializes Gson which helps in storing json lists in the mysql database.

### **Functions:**

* Public static Connect(): Takes no parameters and returns void, it initializes the connection to the mysql database.
* Public static CloseConnection(): Takes no parameters and returns void, it terminates the connection to the database.
* Public static EncryptTickets(): Takes parameter of type Map<Integer, Integer> and returns StringBuilder, it encrypts the tickets map (from the attendee and organizer) in a specific string format to save it in the database.
* Public static DecryptTickets(): Takes parameter of type StringBuilder and returns Map<Integer, Integer>, reverses the encryption done by EncryptTickets function.
* Public static GetData(): Takes parameter String and returns ResultSet (from mysql), it takes the string and performs a mysql query to get the data needed (this makes use of the DataType enum to prevent any typos in the mysql commands.
* Public static Execute(): Takes parameter String and returns void, the string is mysql command for the database to execute (example: insert, delete, etc…).
* Public static GetAdmins(): Takes no parameters and returns ArrayList<Admin>, it returns an arraylist containing all admins in the database.
* Public static GetOrganizers(): Takes no parameters and returns ArrayList<Organizer>, it returns an arraylist containing all organizers in the database.
* Public static GetAttendees(): Takes no parameters and returns ArrayList<Attendee>, it returns an arraylist containing all attendees in the database.
* Public static GetEvents(): Takes no parameters and returns ArrayList<Event>, it returns an arraylist containing all events in the database.
* Public static GetRooms(): Takes no parameters and returns ArrayList<Room>, it returns an arraylist containing all rooms in the database.
* Public static GetCategories(): Takes no parameters and returns ArrayList<Category>, it returns an arraylist containing all categories in the database.
* **DataType (Enum)**

It contains many different states that correspond to a string, which indicated the type of data you try to insert or get from the mysql database. This class is not mandatory but it helps in minimizing risk of error from making a typo in one of the mysql commands.

### **Data Fields:**

* Private final String \_TYPE: Holds the String corresponding to the name of the data type in the database.

### **Functions:**

* Public constructor DataType(): Initializes the \_TYPE String.
* Public toString(): Returns the \_TYPE String.
* **User (Abstract)**

### **Data Fields:**

* Protected String username: Holds the username to log in with
* Protected String email: Holds the email to log in with
* Protected String password: holds the password to log in with
* Protected MyDate dob: stands for date of birth
* Protected Gender gender: holds gender.

### **Functions:**

* Protected constructor User(): Takes parameters of username, email, password, date of birth, and gender and stores user locally.
* Public static Login(): takes parameters username (or) email, and password and checks against the database if the details are linked to a user and returns the User that has successfully logged in.
* Public static GetUsers(): returns all users in an ArrayList<Users>
* Public GetUsername(): returns username in string.
* **Gender (Enum)**

It contains the 2 genders and a third special one (MALE, FEMALE, ENGINEER). It also links each gender string containing the gender in string format, for ease of access.

### **Data Fields:**

* Private final String GENDER: Holds the gender in string format.

### **Functions:**

* Constructor Gender(): This initializes the gender String to the corresponding string format.
* Public toString(): Returns the gender String.
* **Admin (extends User)**

### **Data Fields:**

* Private String role: Holds the role of the admin (example: Manager, Data Engineer).
* Private Hours workingHours: Holds the admin’s working hours in object of class Hours.

### **Functions:**

* Public constructor Admin(): Takes parameters username, email, password, date of birth, gender, role, and working hours. This creates an object of the Admin locally only, but does not save anything in the database.
* Public static RegisterAdmin(): Takes parameters String username, String email, String password, MyDate date of birth, Gender gender, String role, and Hours working hours. Returns void. This inserts a new user of type admin in the mysql database if the entered details are valid.
* Public static FindAdmin(): Takes parameters ArrayList<Admin> and String username and returns Admin, It searches the arraylist for the Admin with the username given.
* Public CreateRoom(): Takes parameters Hours available hours, String location, int capacity. Returns void. This function inserts a new room into the database if the given details are valid.
* Public DeleteRoom(): Takes parameter Room, returns void. This deletes the given room from the database.
* Public AddCategory(): Takes parameter String category name, returns void. This calls the AddCategory() in the ategory class.
* Public RemoveCategory(): Takes parameter String category name, returns void. This calls the RemoveCategory() in the Category class.
* Public toString(): Returns String. This returns a string containing all the admin’s information.
* **Organizer (extends User)**

### **Data Fields:**

* private final Wallet WALLET: Holds the organizer’s wallet.
* private Map<Integer, Integer> ticketsSold: Holds a map of all tickets sold (the event ID and the number of tickets sold).

### **Functions:**

* public Organizer(): Takes parameter of username, email, password, date of birth, gender, wallet and tickets sold and creates an object of the Organizer locally after checking its validity but does not save anything in the database.
* public static RegisterOrganizer(): Takes parameter of username, email, password, date of birth, gender and balance. This inserts a new user of type Organizer in the mysql database if the entered details are valid.
* public static FindOrganizer(): Takes parameters of an ArrayList of organizers and a username and returns the organizer of the same username.
* public CreateEvent(): Takes parameters of Organizer, Event Title, event ID, maximum number of attendees, event price, Room ID, category, event date and event time range and store the event in the database after checking the information validity.
* public EditEventPrice(): Takes parameter of an event and a price and modifies the event’s price after checking the validation of the data.
* public EditEventRoom(): Takes parameter of an event and a room ID and modifies the event’s room after checking the validation of the data.
* public EditEventCategory(): Takes parameter of an event and a category and modifies the event’s category after checking the validation of the data.
* public EditEventDate(): Takes parameter of an event and a date and modifies the event’s date after checking the validation of the data.
* public CancelEvent(): Takes an event as parameter and removes this event from the database and refund all the tickets bought to the attendees (only if the event’s status is Upcoming).
* public GetWallet(): returns a reference for the wallet.
* public GetTicketsSold(): returns the map of all tickets sold (the event ID and the number of tickets sold).
* Public toString(): returns a formatted string of all the information of the event.
* **Attendee (extends User)**

### **Data Fields:**

* Private ArrayList<Category> Interests: Holds the interests of the attendee (like gaming, ceremonies, weddings, etc)
* Private Map<Integer, Integer> tickets: holds the attendees tickets in a map, with Room ID as the key, and the number of tickets as the value corresponding to the key.

### **Functions:**

* Public constructor Attendee(): Takes parameters username, email, password, date of birth, gender, interests, wallet, and tickets. This creates an object of the Attendee locally only, but does not save anything in the database.
* Public static strConvert(): returns StringBuilder of the users interests in a certain format [“interest1”,”interest2”,”interestN”] , useful for database purposes.
* Public static RegisterAttendee(): Takes parameters String username, String email, String password, MyDate date of birth, Gender gender, ArrayList<Category> interests, Wallet wallet, and Map<Integer, Integer> tickets. Returns void. This inserts a new user of type attendee in the mysql database if the entered details are valid.
* Public static FindAtendee(): Takes parameters ArrayList<Attendee> and String username and returns Attendee, It searches the arraylist for the Attendee with the username given.
* Public PurchaseEvent(): takes parameters of the event of which to buy tickets for, and the wanted number of tickets, if the tickets are available and event is suitable for ticket selling, the tickets are bought and stored in the attendee’s Map of tickets and Wallet is changed accordingly.
* Public RefundEvent(): takes parameters of the event of which to refund tickets from, and the wanted number of tickets, if the tickets are available and event is suitable for ticket refunding, the tickets are refunded and removed from the attendee’s Map of tickets and Wallet is changed accordingly.
* Public GetWallet(): returns attendee’s Wallet wallet.
* Public GetInterests (): returns attendee’s ArrayList<Category> interests.
* Public GetTickets (): returns attendee’s Map<Integer,Integer> tickets.
* Public toString(): Returns String. This returns a string containing all the attendee’s information.
* **Room**

### **Data Fields:**

* Private int id: the room’s id
* Private Event currentEvent: holds the current event held in the room
* Private Hours availableHours: the room’s available hours on a weekly basis (Saturday through Friday)
* Private Hours reservedHours: the room’s reserved hours aka hours an event will take place.
* Private final String LOCATION: room’s location
* Private final int MAX\_CAPACITY: room’s maximum capacity

### **Functions:**

* Public constructor Room(): Takes parameters of id, current event, available hours, reserved hours, room location, and maximum capacity and stores room locally.
* Public static FindRoom(): takes parameters of an arraylist of rooms and the room id we are searching for and returns the Room if found, null if not found.
* Public ReserveEvent(): takes the event to reserve as parameter and checks if the room is free and can hold this event (as in if the time is suitable and if the capacity can hold the event) and if all checks are passed, room is reserved for this event.
* Public AddAvailableHours(): takes the day in which we want to add hours and the TimeRange we want to add. If day doesn’t exist, it adds the day along with the hours.
* Public RemoveAvailableHours(): takes parameters the day and timerange we want to remove, if the day and timerange exists, the time is removed, if doesn’t exist, nothing happens to the avaiableHours
* Public SetAvailableHours(): takes Hours as parameter, and sets it as the avaible hours.
* Public GetID(): returns Room ID
* Public GetCurrentEvent: returns currentEvent
* Public GetAvailableHours: returns availableHours
* Public GetReservedHours(): returns reservedHours
* Public GetMaxCacapcity(): returns MAX\_CAPACITY
* **Event**

### **Data Fields:**

* Private final Organizer ORGANIZER: Holds the organizer of the event.
* Private final String EVENT\_TITLE: Holds the title of the event.
* Private final int ID: Holds the ID of the event.
* Private final int ROOM\_ID: Holds the room ID of the room in which the event takes place.
* Private Category category: Holds the category of the event.
* private int maxNumOfAttendees: Holds the maximum number of attendees that can join the event.
* Private double price: Holds the price of the event.
* Private MyDate date: Holds the date of the event (example: 14/12/2006).
* Private TimeRange timeRange: Holds the time range of the event (example: 18:00-22:00).
* Private final Status STATUS: Holds the status of the event (example: upcoming, ongoing, over).

### **Functions:**

* Public Event(): Takes parameters Organizer, Event Title, event ID, maximum number of attendees, event price, Room ID, category, event date and event time range. This creates an object of the Event locally but does not save anything in the database.
* public static FindEvent(): Takes parameters of an ArrayList of events and the event ID and returns the event of the same ID.
* Public EditPrice(): takes a parameter of the new price and edits it in the object and database.
* Public ChangeRoom(): takes a parameter of the new room and edits it in the object and database after checking its validity.
* Public EditCategory(): takes a parameter of the new Category and edits it in the object and database after checking its validity.
* Public EditDate(): takes a parameter of the new date and edits it in the object and database after checking its validity.
* Public EditTimeRange(): takes a parameter of the new time range and edits it in the object and database after checking its validity.
* Public EditMaxNumOfAttendees(): takes a parameter of the new number and edits it in the object and database after checking its validity.
* Public GetStatus(): returns the Status of the event.
* Public GetPrice(): returns the ticket price.
* Public GetID(): returns the event ID.
* Public GetOrganizer(): returns the Organizer of the event.
* Public GetTimeRange(): returns the time range of the event.
* Public GetDate(): returns the date of the event.
* Public GetRoomID(): returns the event’s room ID.
* Public GetEventTitle(): returns the event title.
* Public GetCategory(): returns the Category of the event.
* Public GetMaxNumOfAttendees(): returns the maximum number of attendees.
* Public toString(): returns a formatted string of all the information of the event.
* **Status (Enum)**

It contains three states that the event could be, which are the “Upcoming” events, the ”Ongoing” events and the “Over” events. They are used to minimize the chances of logical errors (example: buying a ticket for over events).

* **Category**

### **Data Fields:**

* Private String category: Holds the name of the category in string format.

### **Functions:**

* Public Constructor Category(): This creates an object of Category locally and gives it the “Default” category.
* Public Constructor Category(): Takes parameter String and creates an object of Category locally with this string as the category name if it is a name that exists in the Category database.
* Static AddCategory(): Takes parameter String and returns void. It inserts a new category into the category database, if it doesn’t exist already.
* Static RemoveCategory(): Takes parameter String and returns void. It deletes the given category from the database, if it exists.
* Public toString(): Returns the category String.
* **Wallet**

### **Data Fields:**

* private double balance: Holds the wallet balance.
* private final int WALLET\_NUMBER: Holds the wallet number.

### **Functions:**

* Public Wallet(): takes parameters of the balance and wallet number and creates an object of the Wallet locally after checking information validity and does not save anything in the database.
* Public static CreateWallet(): takes parameter of the username and balance and insert the wallet into the database and assign the balance to the username.
* public static FindEvent(): Takes parameters of an ArrayList of Wallets and a wallet number and returns the Wallet of the wallet number.
* Public EditBalance(): takes a parameter of an amount and edits the balance of the wallet (either by increase or decrease) and in case of decreasing the balance, it checks if the amount is valid.
* Public GetBalance(): returns the wallet balance.
* Public toString(): returns a formatted string of all the information of the wallet.
* **TimeRange**

### **Data Fields:**

* Double start: Holds the starting hour of the time range (example: 14:15 -> 14.25).
* Double end: Holds the ending hour of the time range.

### **Functions:**

* Public Constructor TimeRange(): Take two double parameters (the start and end), and creates a local object of the class with the given doubles.
* Public Constructor TimeRange(): Takes two String parameters. It makes use of the Convert() function to convert the string time range to doubles, and creates a local object with those doubles.
* Public GetTotalTime(): Takes no parameters and returns double. It calculates the total time taken by the time range by subtracting the start from the end.
* Public Static Convert(): Takes parameter of type double and returns String. Converts a time from double to String (example: 6.25 -> 06:15).
* Public Static Convert(): Takes parameter of type String and returns double. Converts a time from String to double.
* Public Contains(): Takes parameter TimeRange and returns a boolean. It calculates if the given time range is all inside the time range of the instance object that calls the function.
* Public Remove(): Takes parameter TimeRange and returns an ArrayList<TimeRange>. It removes the given time range from the instance object the calls the function and returns an array list containing the time ranges that remain.
* Public Static GetOverlap(): Takes two parameters TimeRange, and returns a TimeRange. This calculates the overlap between any two time ranges and returns that overlap in another time range. It returns null if there is no overlap.
* Public InRange(): Takes parameter double and returns boolean. It checks if the given double (symbolizing an hour) is inside the time range of the instance object that calls this function.
* Public Static EncryptWorkingHours(): Takes parameter Map<String, ArrayList<TimeRange>> and returns a String. This function encrypts the map (symbolizing available hours or working hours) to a String format that can be stored in the mysql database (Check the comments in the function for example).
* Public Static DecryptWorkingHours(): Takes parameter String and returns Map<String, ArrayList<TimeRange>>. This function reverses the encryption done by the EncryptWorkingHours() function.
* Public Static EncryptTimeRange(): Takes parameter Map<String, ArrayList<TimeRange>> and returns a String. This function encrypts the map (symbolizing reserved hours, which is different from the above functions because it requires any date not days of the week) to a String format that can be stored in the mysql database.
* Public Static DecryptTimeRange(): Takes parameter String and returns Map<String, ArrayList<TimeRange>>. This function reverses the encryption done by the EncryptTimeRange() function.
* Public GetStart(): Returns the start double.
* Public GetEnd(): Returns the end double.
* Public toString(): Returns the time range in String format (example: “08:00-19:30”).
* **Hours**

### **Data Fields:**

* Map<String, ArrayList<TimeRange>> map: a linked hashMap holding a schedule of days as Strings and their TimeRanges.

### **Functions:**

* Public constructer Hours(): takes map as parameter and stores locally
* Public Contains(): takes a day as string and a timerange and checks whether this timerange is contained in the day taken as parameter, returns the result of the check as Boolean value 1 for is contained, 0 if not contained.
* Public Overlaps(): takes string day and a timeRange and checks if there is any overlap, Boolean return, if overlap exists, returns 1, if overlap doesn’t exist, returns 0.
* Public RemoveTime(): takes parameters of the day we want to remove a time from and the TimeRange we want to remove and removes the time from the map.
* Public AddTime(): takes parameters of the day we want to add and the timeRange to add, and adds it to the map.
* Public CheckArrange(): Checks the map for any formatting issues and fixes the issues (for instance, a duplicate time, overlapping time)
* Public toString(): returns the map.toString()
* **Day (Enum)**

It contains seven different states (SATURDAY -> FRIDAY) that correspond to a day of the week. It also links each day of the week to a string containing the name of the day in string format, for ease of access.

### **Data Fields:**

* Private String day: Holds the name of the day in string format.
* Public static Map<String, Integer> days: Map with the keys being the day’s name, and the values being integers from 0 to 6 corresponding to Saturday to Friday.

### **Functions:**

* Constructor Day(): This initializes the day String to the corresponding string format of the day.
* Public static Translate(): Takes parameter int which is the number of the day from 0 to 6, and returns the Day corresponding to it.
* Public toString(): Returns the day String.
* **MyDate**

### **Data Fields:**

* private final int DAY: Holds the day.
* private final int MONTH: Holds the month.
* private final int YEAR: Holds the year.

### **Functions:**

* public MyDate(): takes parameters day, month and year as integer values and creates an object of the MyDate locally after checking information validity and does not save anything in the database.
* public MyDate(): takes parameters day, month and year as a string and creates an object of the MyDate locally after checking information validity and does not save anything in the database.
* private static IsLeapYear(): takes the year and returns true if it is a leap year. Used in the validations in the constructors.
* public int GetDay(): returns the day.
* public int GetMonth(): returns the month.
* public int GetYear(): returns the year.
* public static GetDayOfTheWeek(): takes parameter of a date and returns the day of the week of this date (example: Sunday, Monday etc).
* public toString(): returns a formatted string of the date.
* public equals(): takes parameter of an object and returns true of it is equal to the date object calling this method.

# **GitHub Link**

<https://github.com/ManCityP/OOP_EMS>