# Git Token: ghp\_QzMtBu6QUrVo7a4kKtBeEBChtP905O10zUco

**Project 1**

**Library Management System**

Library management system will be responsible for managing a Library and its associated entities appropriately. Here, there are 3 major entities: books, writers & users.

**Each of these entities will have 2 attributes in common:**

* Instance level attribute "ID"
* Static variable named "counter", which will be initialized with 0. In each object creation, the "ID" attribute will be set as the current value of this variable. And after the object creation, this static variable needs to be incremented by 1.

**In the Library, we can add a book with the following attributes:**

1. Title
2. ISBN
3. Authors
4. category
5. Release year
6. Awards received

We should also be able to keep track of the inventory of books: while adding a book in the stock, we can provide the initial stock quantity, later on, we can also add more copies of the same book. Every book should be linked to its authors. We should be able to get details of the authors of a specific book.

**We can also add a writer in the system, with the following details:**

1. First Name
2. Last Name
3. Awards received
4. Active since

We should also be able to retrieve all the books a writer has authored or co-authored. Also, on what topics, a specific writer has written books.

**The system will have the provision of registering users with the following attributes:**

* First name
* Last name
* Contact
  + Email ID
  + Phone
* Address
  + City
  + Locality
  + State
  + Pin Code

Once the user completes registration, the system should also capture the time(Format: dd/MM/yyyy hh:mm:ss AM/PM), when the user has registered. A User should be able to have a maximum of 3 books at any point of time.

The Library management application should be able to track the book's inventory and user's book allotment history. The following operations should be supported in the same class:

1. Register an user
2. Add a writer
3. Add a new book
4. Add copies of an existing book
5. Allocate a copy of a book to an user
6. Return a copy of the book
7. Get books of an author
8. Get books of a specific category.

All these operations should have the required validations in place, like: a user can't have more than three books at a time. Book, writer and user identifier should be valid.

**Project 2**

**HR Management application**

The HR Application will keep track of employees' access and leaves. The complete application revolves around employees only. **Once an employee joins the company, the following attributes will be captured:**

* First name
* Last name
* Role
* Reports To
* Leave balance
  + Sick leaves
  + Paid Leaves
* Contact
  + Email ID
  + Phone
* Address
  + City
  + Locality
  + State
  + Pin Code

**Every employee also have these 2 attributes:**

* Instance level attribute "ID"
* Static variable named "counter", which will be initialized with 0. In each object creation, the "ID" attribute will be set as the current value of this variable. And after the object creation, this static variable needs to be incremented by 1.

The Employee with ID 0 denotes the CEO of the company, everybody except him/her reports someone else.

While onboarding an employee, there should be a proper check, such that he/she reports to a valid employee. As per company policy, every employee is entitled to 12 days of sick leave and 10 days of paid leaves in a year. While onboarding leave balances should be populated appropriately.

**There should be a different class called HR Application, which will be the interface for all customer-facing operations:**

1. Onboard an employee
2. Change manager of an employee
3. An employee should be able to apply for leave. An employee can apply for leave, only if there's sufficient balance for the corresponding leave type. Until the manager approves the leave, the leaves will be in an "Applied" state.
4. For each manager, there should be an option to see the pending leave application
5. Each manager can approve/reject a leave application. The leave state should be updated accordingly.
6. We should be able to get details of an employee with a given email ID or phone number. In case, the email or phone number is not valid, it should show appropriate failure messages.
7. For each manager, we should be able to list down the employee details, who are reporting to him/her.

There should be separate classes for managing Leaves. Leaves will be either of type Sick leave or Paid leave. Each leave should have a start date, end date & current status.

**Proect 3**

**Messaging App**

In this assignment, we will be building a messaging app, where users can send messages to other users personally or in a group.

**The users can register with the following attributes:**

* First name
* Last name
* Contact
  + Email ID
  + Phone
* Link to Profile picture
* Status Message

Once the user completes registration, the system should also capture the time (Format: dd/MM/yyyy hh:mm:ss AM/PM), when the user has registered. **Every user will also have these 2 attributes:**

* Instance level attribute "ID"
* Static variable named "counter", which will be initialized with 1. In each object creation, the "ID" attribute will be set as the current value of this variable. And after the object creation, this static variable needs to be incremented by 1.

The system should also keep track of one-to-one(private) and group messages for a user with proper messages. **Both private and group messages will have a set of common attributes and a few attributes specific to the type of the message. The common attributes will be:**

* Message-ID(All messages should have counter and ID similar to that of user ID & counter)
* Message Text
* Sender user ID
* Sending timestamp

The value of message ID should remain unique across private and group chat messages.

**Note: You can achieve this by using a common parent class of both the types of messages and implementing the ID & counter population logic in the Superclass constructor.**  
  
**The following attributes will belong to private messages only:**

* Receiver user ID
* Received timestamp
* Message status[Unread or Read]

**The following attributes will belong to only group chat messages:**

* Group ID
* Received timestamps [A map of user ID and their respective received timestamps]
* The message read statuses [A map of User ID and the respective messages]

**A Group will contain the following information:**

* Group ID
* Group name
* Group Icon link
* List of admin user IDs
* List of non-admin member user IDs
* Creation timestamp

Another class needs to be created for running the functionalities together, let's call it MessagingApp.**The messaging app should support the following operations:**

1. Register a user
2. Send private messages from one person to another
3. Receive all chats with their most recent message for a user with respective statuses
4. See specific private chat's messages
5. Create a group of users
6. Adding a user to a group
7. See a group messages for a user

For all operations, the timestamp and message statuses have to be populated automatically. When a message is sent, the sending timestamp should be captured as the sending timestamp and the message status will be set as "Unread" for all users in the group or private chat. **Once a user enters a private/group chat, only then the corresponding message's status will be set to "Read" for that specific user.**

# Project 4

# 2 Player Tic Tac Toe

A 2 player tic-tac-toe game should be played between 2 players present in the system. **The application will revolve around the following entities:**

1. Player
2. Game
3. Move

**A player can register with the following details:**

* First name
* Last name
* Contact
  + Email ID
  + Phone
* Link to Profile picture

Once the user completes registration, the system should also capture the time(Format: dd/MM/yyyy hh:mm:ss AM/PM), when the user has registered. **Every user will also have these 2 attributes:**

* Instance level attribute "ID"
* Static variable named "counter", which will be initialized with 1. In each object creation, the "ID" attribute will be set as the current value of this variable. And after the object creation, this static variable needs to be incremented by 1.

**Each Game consists of multiple valid moves and a few other game level attributes:**

* ID (ID & counter needs to be maintained similar to that of the user)
* Start Timestamp
* Current Status
* List of moves
* End timestamp
* Winner
* Current Turn Player ID

A game will not have any value for the winner or end timestamp, until it's concluded. A game will conclude, when a user has won or when all the positions have been filled up, just like a physical tic-tac-toe game. Once a game has concluded, no further moves will be accepted.

**Each move will consist of the following details:**

* Player ID
* Marker symbol
* Horizontal position(HP)
* Vertical position(VP)

Please check the following diagram for understanding the horizontal & vertical position in detail:

+-------+------+----+

| HP = 1 | HP = 2 | HP = 3 |

| VP = 1 | VP = 1 | VP = 1 |

+-------+------+----+

| HP = 1 | HP = 2 | HP = 3 |

| VP = 2 | VP = 2 | VP = 2 |

+-------+-------+----+

| HP = 1 | HP = 2 | HP = 3 |

| VP = 3 | VP = 3 | VP = 3 |

+-------+-------+----+

Every move needs to be validated before accepting. A move will be considered invalid if the provided inputs for the horizontal or vertical positions are invalid or the corresponding place is already filled in the current game. After every move, the status & winner(if any) of the game needs to be calculated.

**A global Application class needs to be created for supporting the following operations:**

1. Register a user
2. Create a game between 2 existing players.
3. Let the 2 users play the game by entering their respective moves
4. Show the current status of the game
5. See the details of a user. It should print the user's profile attributes and the number of games the user has won and lost.

**Project 5**

**Expense reimbursement portal**

Expense reimbursement portal will be used for managing the invoices submitted for expense claims for a  company. Each vertical/department in a company can have their own budget for a specific type of expense.

There will be a set of different types of expenses defined in the system level: likes: relocation, travel, morale etc. Every expense request needs to be submitted against one of these fixed sets of expense types.

**For each department in a company, the following details need to be stored:**

* Department Name
* Total number of employees
* Total budget for different types of expenses for each employee
* Total amount spent for each type of expenses across all employees

**Once an employee joins the company, the following attributes will be captured:**

* First name
* Last name
* Role
* Reports To
* Department ID
* Contact
  + Email ID
  + Phone
* Address
  + City
  + Locality
  + State
  + Pin Code

**Every employee, Department, Expense reimbursement request also have these 2 attributes:**

* Instance level attribute "ID"
* Static variable named "counter", which will be initialized with 0. In each object creation, the "ID" attribute will be set as the current value of this variable. And after the object creation, this static variable needs to be incremented by 1.

The Employee with ID 0 denotes the CEO of the company, everybody except him/her reports someone else. While onboarding an employee, there should be a proper check, such that he/she reports to a valid employee or whether the employee belongs to a valid department.

**Every expense reimbursement request should contain the following details other than ID & counter:**

* Expense type
* Amount of expenditure
* Date for the expense
* Employee ID
* Approver ID
* Timestamp for request submission.
* Status

**No expense request can be submitted, in case:**

* If the date of expense is older than 1 month from today's date or
* The amount requested is more than the budget allocated for that expense

An expense request can be approved by the manager of the employee. **A global application class needs to be created for providing interface for the following operations:**

1. Add a department
2. Onboard an employee
3. Change manager of an employee
4. An employee should be able to submit an expense reimbursement request. Until the manager approves the request, the leaves will be in "Submitted" state.
5. For each manager, there should be option to see the pending requests for approval
6. Each manager can approve/reject a reimbursement request. The request status should be updated accordingly.
7. For each department, we should be able to see the current available budget for different expense types
8. For each department, we should be able to see the total expense made so far across all employees for each type of expense.

# Project 6

# Furniture Renting

Furniture renting application keeps track of furniture and rents them among customers. The application majorly maintains 2 entities: furniture and customer.

**A customer can register into the system with the following attributes:**

* First name
* Last name
* Contact
  + Email ID
  + Phone
* Address
  + Apartment
  + City
  + Locality
  + State
  + Pin Code

Once the user completes registration, the system should also capture the time(Format: dd/MM/yyyy hh:mm:ss AM/PM), when the user has registered.

**Furnitures can be added to the system with the following attributes:**

* Title
* Description
* Color
* Length
* Width
* Leight
* Monthly rent
* Type of furniture
* Suitable rooms
* Quantity in stock

**Both furniture and customers will also have these 2 attributes:**

* Instance level attribute "ID"
* Static variable named "counter", which will be initialized with 0. In each object creation, the "ID" attribute will be set as the current value of this variable. And after the object creation, this static variable needs to be incremented by 1.

**The application should also keep track of the rent invoices, containing the following details:**

* ID (ID needs to be generated from counter variable n the same way as of customer & furnitures)
* Furniture ID
* Customer ID
* Amount
* Status
* Start date
* End date
* Delivery Address
  + Apartment
  + City
  + Locality
  + State
  + Pin Code

**There should be a central class for maintaining the application, which provides the interfaces for the following operations:**

1. Register a customer
2. Add a furniture
3. Add more stocks for a specific furniture
4. Search for furniture based on the type of furniture. It should show only from the list of available ones, sorted by monthly rent in ascending order.
5. The customers should also search the furniture based on both color and type of furniture in similar manner.
6. Rent a furniture to a user. This should also generate the rent invoice for that customer.
7. Cancel a rent invoice. The user can cancel a rent invoice, only before the start date, free of cost. Once the start date has begun, the invoice can no longer be cancelled.
8. Return furniture post the end period for renting.

# Project 7

# Telephone directory(Truecaller)

The telephone directory application will maintain the list of users and their phone numbers. **Each user will have the following details:**

* Phone number
* First name
* Last name
* City
* State
* Picture
* Last seen timestamp

**The users will also have these 2 attributes:**

* Instance level attribute "ID"
* Static variable named "counter", which will be initialized with 0. In each object creation, the "ID" attribute will be set as the current value of this variable. And after the object creation, this static variable needs to be incremented by 1.

Once the user completes registration, the system should also capture the time (Format: dd/MM/yyyy hh:mm:ss AM/PM), when the user has registered.

Every user will have a subscription to the application. By default, all the users will be having a free subscription available. In the free subscription, a user can do only upto 10 searches a day. A user can buy the premium subscription of the application by paying the subscription cost. Once the premium subscription is availed, the user can do unlimited searches per day.

The cost of a premium subscription is fixed at the application level. There will be a global call named TeleCallerApp, which will store all the user's contacts. It will also store the search history of all users.

**The TeleCallerApp will also provide an interface for the following operations:**

1. Add a user's contact
2. Update a specific user's contact
3. Update a list of user's contact
4. Search for a user by phone number
5. Search a user's contact by his/her name
6. Search all users, where the name contains a keyword. E.g. Search for users, where the name of the user contains the word "elvi". it will return the users having "Kelvin" as their first name. If there is another user present with the first name "Elvis'', then that user should also be displayed. **Hint:** **Use the "contains" method available in the Java String class.**
7. Avail premium subscription for a user
8. Get the search history of a specific user.

All the search operations need to be case insensitive. **For operations numbered 5 & 6, the search needs to be done in both first & last name.** Each operation needs to have proper validations. Like once a user has exhausted the search limit of free subscriptions, appropriate error messages should be shown.

# Project 8

# Stock Broker

The Stock Broker application keeps track of stocks of multiple companies and notifies the various brokers about the change in prices of the stocks they have subscribed to.

**Every stock should be holding the following attributes:**

* Stock Symbol
* Current price
* launch timestamp
* Company name

Each stock should be uniquely identifiable by its stock Symbol.

**Brokers can register in the system with the following attributes:**

* First name
* Last name
* Contact
  + Email
  + Phone
* City
* Locality

**The broker will also have these 2 attributes:**

* Instance level attribute "ID"
* Static variable named "counter", which will be initialized with 0. In each object creation, the "ID" attribute will be set as the current value of this variable. And after the object creation, this static variable needs to be incremented by 1.

Once the user completes registration, the system should also capture the time(Format: dd/MM/yyyy hh:mm:ss AM/PM), when the user has registered. **There will also be an application class for providing interfaces for all of the following operations:**

1. Register a broker
2. Add a stock
3. Update a specific stock's price. This should notify all the brokers who have subscribed to that specific stock
4. Each broker can subscribe to various stocks
5. Each broker should also be able to unsubscribe from a specific stock
6. Each broker should be able to buy a stock
7. Each broker should also be able to sell a stock
8. For each broker, we should be able to provide how much profit/loss that broker has made for each of the stocks.

**Bonus:** Try to apply the **Observer design pattern**. You can learn about the **Observer design pattern** from [here](https://www.baeldung.com/java-observer-pattern) and then try to implement it. You can also use the online resources available to understand the concept.