Social Magnet

IS442-Object Oriented Programming

G1-T7

Andrew Liew

Samuel Chia

Gerard Tan

Zexel Lew

Content page

Introduction

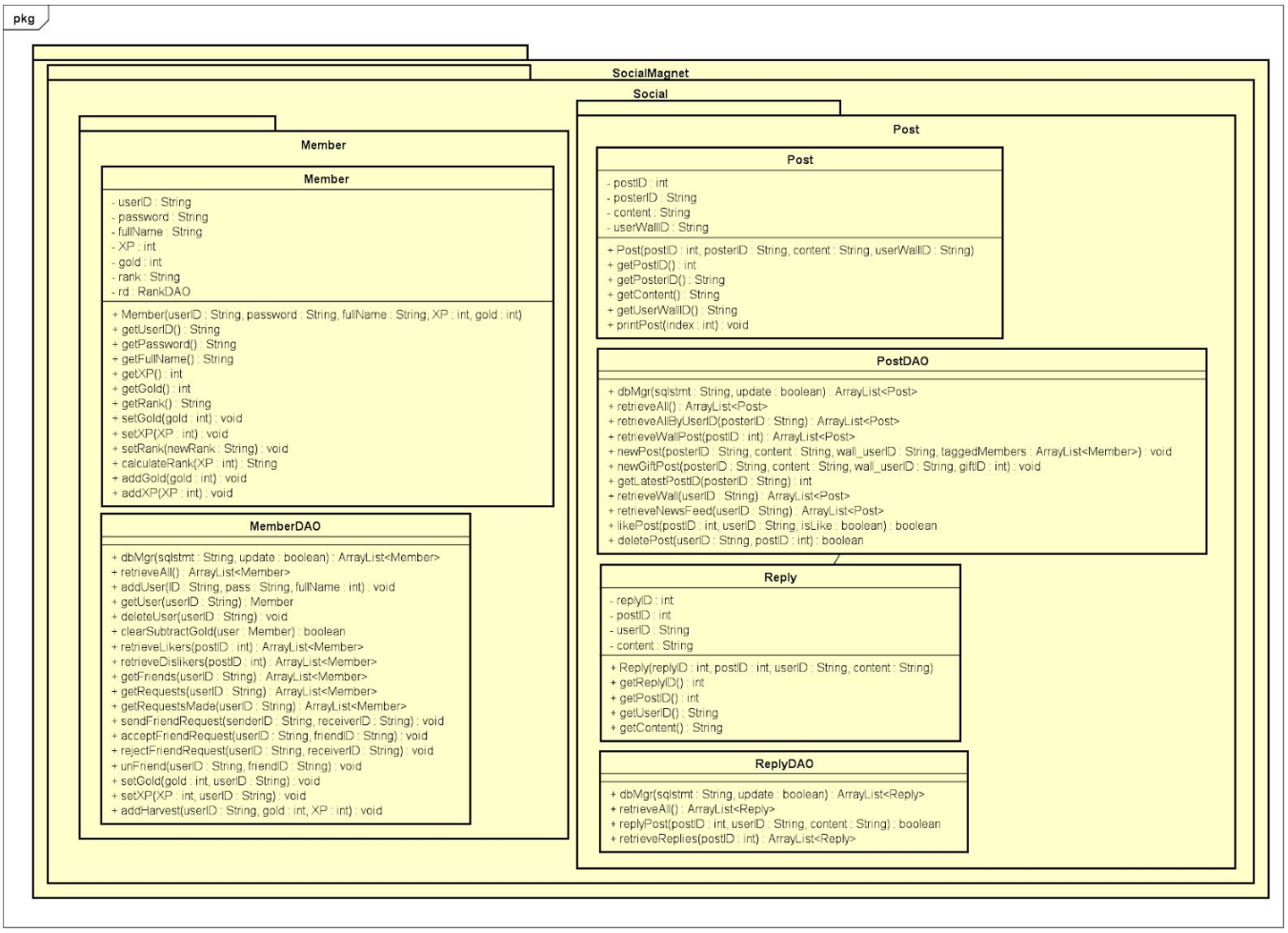
After some thought on how we should start off and tackle the project, our approach to this project was to first come up with the database structure, and design the Entity-Relationship (ER) model. We first decided on what entities were needed, and what attributes were required for each entity. Next was to link them up nicely, showing the relationships between each entity. With the ER model, our team was able to better visualize how each table in the database are linked to each other

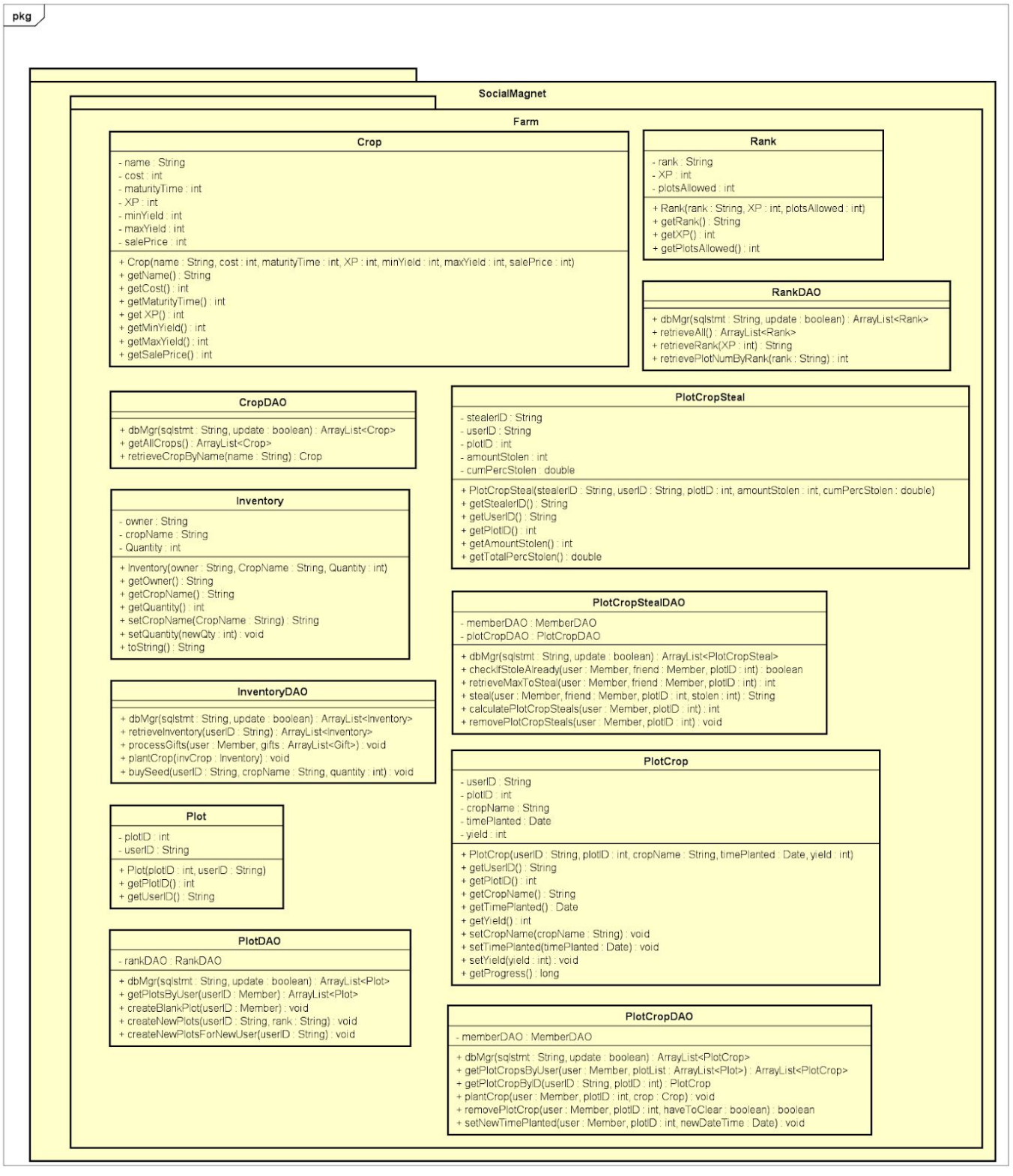
The next step we took was to create the database. We used MySQL and its GUI, phpMyAdmin to construct the structure and insert data into the tables. With the database in place, we were then able to start on the programming for the project. The flow of which component we tackled first is:

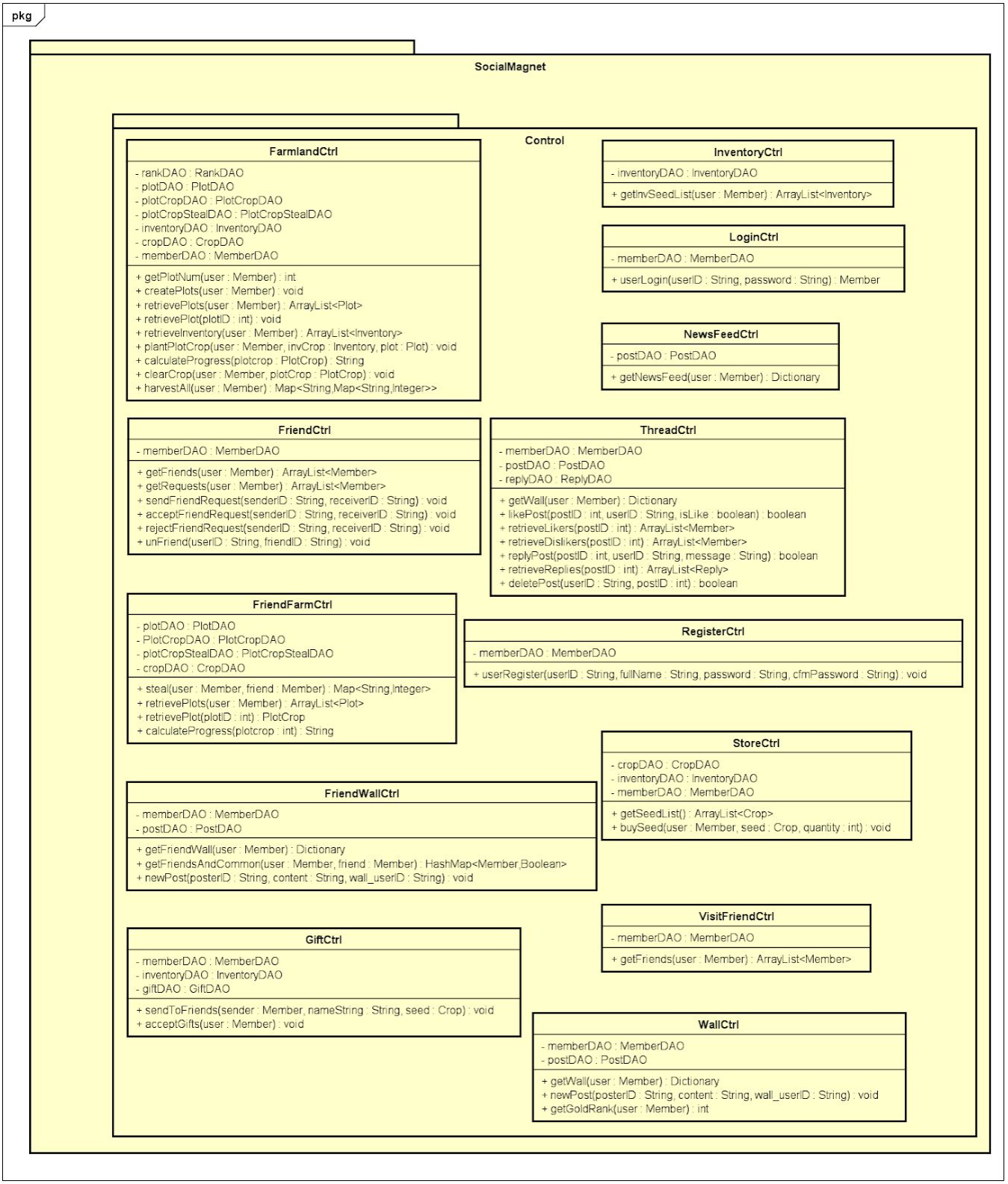
All the Menus → Registration → Login → Post (writing of post on wall) → Friends → Likes/Dislikes → Replies → Post Tagging → Newsfeed | Farmland rank (Journeyman, etc.) → Plots → Inventory → Store → Planting crops → Clearing Crops → Harvesting crops → Visiting friends → Steal → Gift

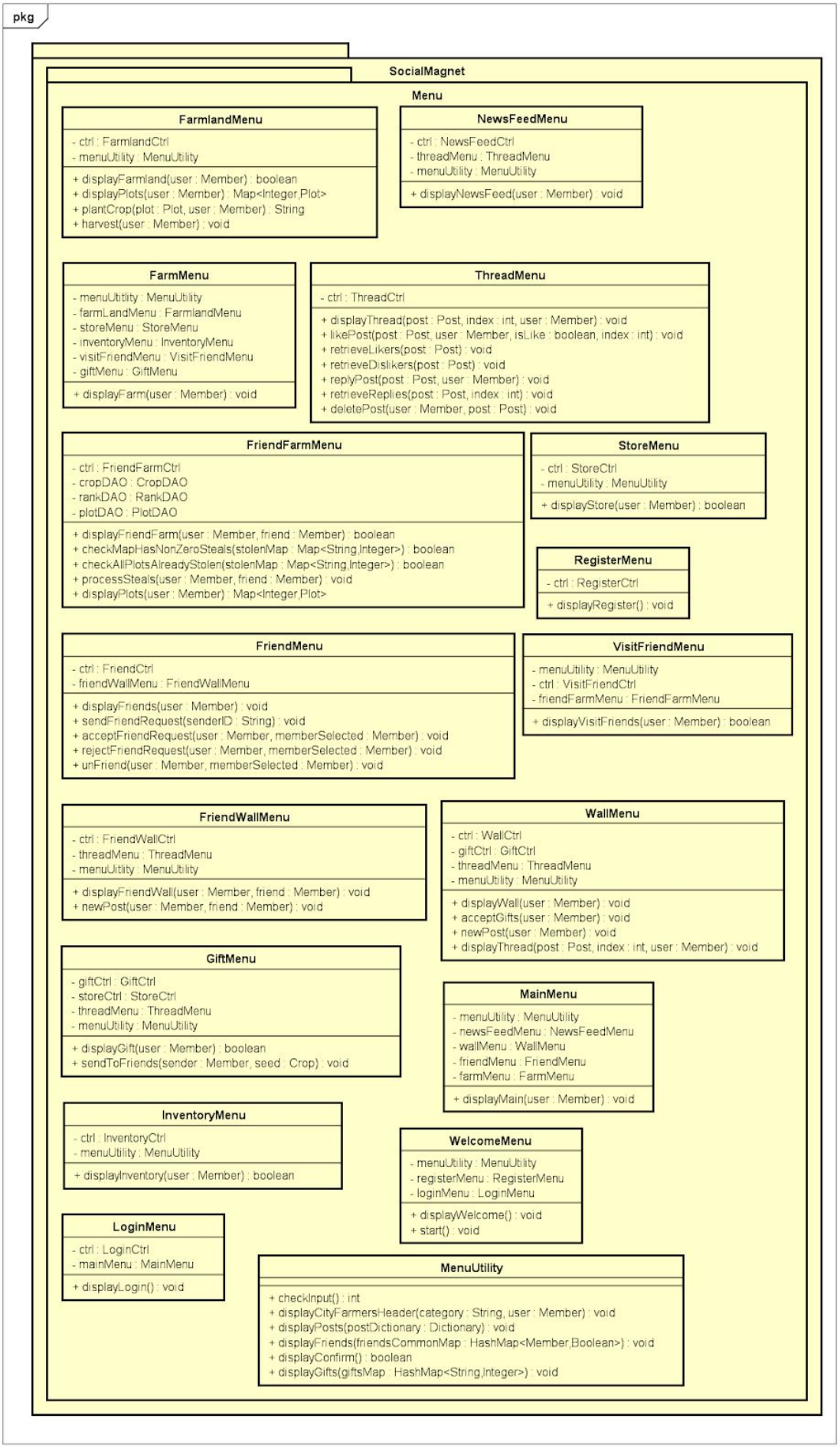
Social Magnet Class Diagram

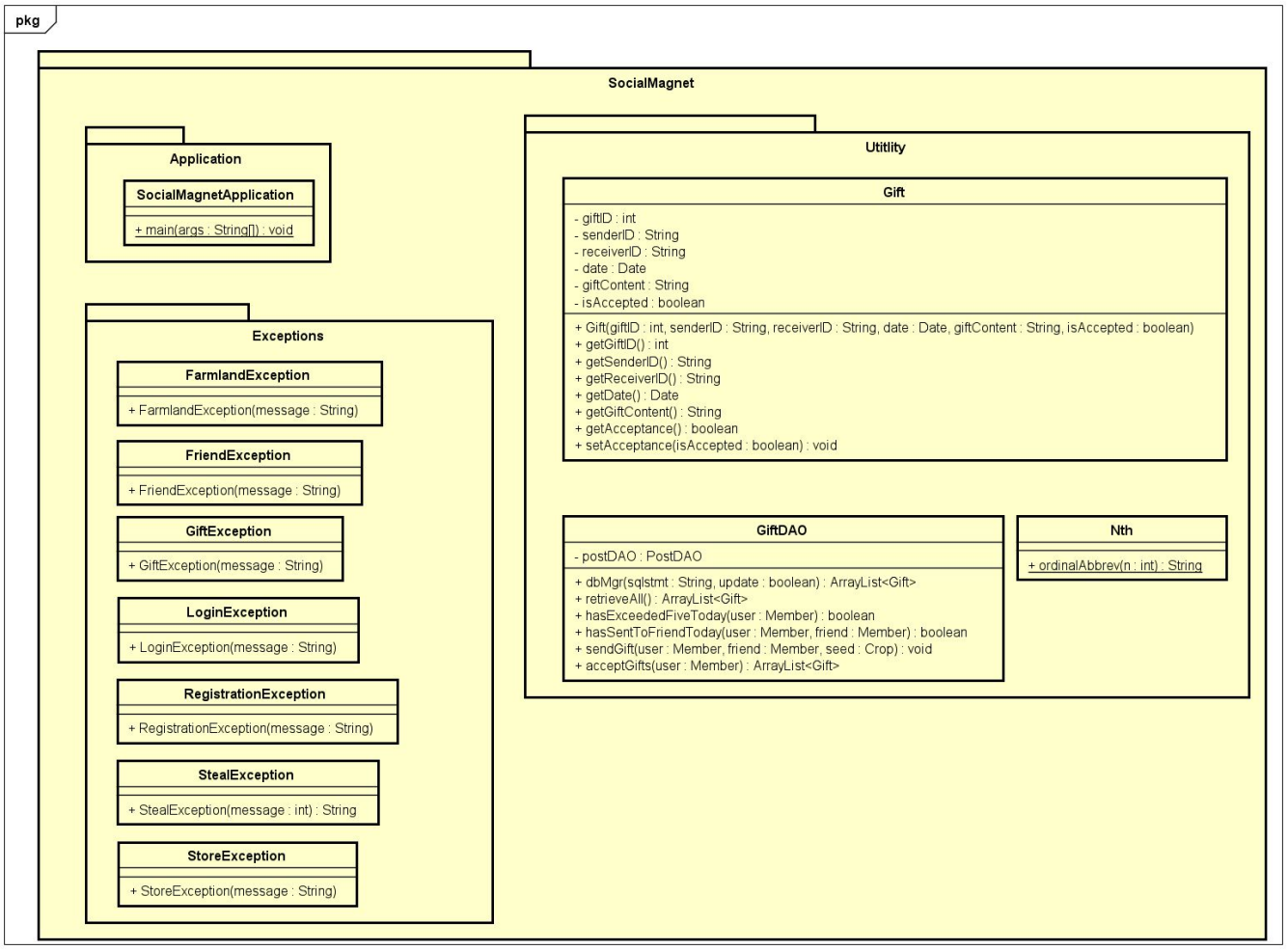
Due to the number of classes, we have split the class diagram to multiple parts. There is no inheritance. Associations were not depicted in the class diagram. This was done to maintain readability of the diagram. Associations can be seen in the class's attribute sections.




pkg
SocialMagnet
Farm
Crop
- name : String
- cost : int
- maturityTime : int
- XP : int
- minYield : int
- maxYield : int
- salePrice : int
+ Crop(name : String, cost : int, maturityTime : int, XP : int, minYield : int, maxYield : int, salePrice : int)
+ getName() : String
+ getCost() : int
+ getMaturityTime() : int
+ get XP() : int
+ getMinYield() : int
+ getMaxYield() : int
+ getSalePrice() : int
Rank
- rank : String
- XP : int
- plotsAllowed : int
+ Rank(rank : String, XP : int, plotsAllowed : int)
+ getRank() : String
+ getXP() : int
+ getPlotsAllowed() : int
RankDAO
+ dbMgr(sqlstmt : String, update : boolean) : ArrayList<Rank>
+ retrieveAll() : ArrayList<Rank>
+ retrieveRank(XP : int) : String
+ retrievePlotNumByRank(rank : String) : int
CropDAO
+ dbMgr(sqlstmt : String, update : boolean) : ArrayList<Crop>
+ getAllCrops() : ArrayList<Crop>
+ retrieveCropByName(name : String) : Crop
PlotCropSteal
- stealerID : String
- userID : String
- plotID : int
- amountStolen : int
- cumPercStolen : double
+ PlotCropSteal(stealerID : String, userID : String, plotID : int, amountStolen : int, cumPercStolen : double)
+ getStealerID() : String
+ getUserID() : String
+ getPlotID() : int
+ getAmountStolen() : int
+ getTotalPercStolen() : double
Inventory
- owner : String
- cropName : String
- Quantity : int
+ Inventory(owner : String, CropName : String, Quantity : int)
+ getOwner() : String
+ getCropName() : String
+ getQuantity() : int
+ setCropName(CropName : String) : String
+ setQuantity(newQty : int) : void
+ toString() : String
PlotCropStealDAO
- memberDAO : MemberDAO
- plotCropDAO : PlotCropDAO
+ dbMgr(sqlstmt : String, update : boolean) : ArrayList<PlotCropSteal>
+ checkIfStoleAlready(user : Member, friend : Member, plotID : int) : boolean
+ retrieveMaxToSteal(user : Member, friend : Member, plotID : int) : int
+ steal(user : Member, friend : Member, plotID : int, stolen : int) : String
+ calculatePlotCropSteals(user : Member, plotID : int) : int
+ removePlotCropSteals(user : Member, plotID : int) : void
InventoryDAO
+ dbMgr(sqlstmt : String, update : boolean) : ArrayList<Inventory>
+ retrieveInventory(userID : String) : ArrayList<Inventory>
+ processGifts(user : Member, gifts : ArrayList<Gift>) : void
+ plantCrop(invCrop : Inventory) : void
+ buySeed(userID : String, cropName : String, quantity : int) : void
PlotCrop
- userID : String
- plotID : int
- cropName : String
- timePlanted : Date
- yield : int
+ PlotCrop(userID : String, plotID : int, cropName : String, timePlanted : Date, yield : int)
+ getUserID() : String
+ getPlotID() : int
+ getCropName() : String
+ getTimePlanted() : Date
+ getYield() : int
+ setCropName(cropName : String) : void
+ setTimePlanted(timePlanted : Date) : void
+ setYield(yield : int) : void
+ getProgress() : long
Plot
- plotID : int
- userID : String
+ Plot(plotID : int, userID : String)
+ getPlotID() : int
+ getUserID() : String
PlotDAO
- rankDAO : RankDAO
+ dbMgr(sqlstmt : String, update : boolean) : ArrayList<Plot>
+ getPlotsByUser(userID : Member) : ArrayList<Plot>
+ createBlankPlot(userID : Member) : void
+ createNewPlots(userID : String, rank : String) : void
+ createNewPlotsForNewUser(userID : String) : void
PlotCropDAO
- memberDAO : MemberDAO
+ dbMgr(sqlstmt : String, update : boolean) : ArrayList<PlotCrop>
+ getPlotCropsByUser(user : Member, plotList : ArrayList<Plot>) : ArrayList<PlotCrop>
+ getPlotCropByID(userID : String, plotID : int) : PlotCrop
+ plantCrop(user : Member, plotID : int, crop : Crop) : void
+ removePlotCrop(user : Member, plotID : int, haveToClear : boolean) : boolean
+ setNewTimePlanted(user : Member, plotID : int, newDateTime : Date) : void


pkg
SocialMagnet
Control
FarmlandCtrl
- rankDAO : RankDAO
- plotDAO : PlotDAO
- plotCropDAO : PlotCropDAO
- plotCropStealDAO : PlotCropStealDAO
- inventoryDAO : InventoryDAO
- cropDAO : CropDAO
- memberDAO : MemberDAO
+ getPlotNum(user : Member) : int
+ createPlots(user : Member) : void
+ retrievePlots(user : Member) : ArrayList<Plot>
+ retrievePlot(plotID : int) : void
+ retrieveInventory(user : Member) : ArrayList<Inventory>
+ plantPlotCrop(user : Member, invCrop : Inventory, plot : Plot) : void
+ calculateProgress(plotcrop : PlotCrop) : String
+ clearCrop(user : Member, plotCrop : PlotCrop) : void
+ harvestAll(user : Member) : Map<String,Map<String,Integer>>
InventoryCtrl
- inventoryDAO : InventoryDAO
+ getInvSeedList(user : Member) : ArrayList<Inventory>
LoginCtrl
- memberDAO : MemberDAO
+ userLogin(userID : String, password : String) : Member
NewsFeedCtrl
- postDAO : PostDAO
+ getNewsFeed(user : Member) : Dictionary
FriendCtrl
- memberDAO : MemberDAO
+ getFriends(user : Member) : ArrayList<Member>
+ getRequests(user : Member) : ArrayList<Member>
+ sendFriendRequest(senderID : String, receiverID : String) : void
+ acceptFriendRequest(senderID : String, receiverID : String) : void
+ rejectFriendRequest(senderID : String, receiverID : String) : void
+ unFriend(userID : String, friendID : String) : void
ThreadCtrl
- memberDAO : MemberDAO
- postDAO : PostDAO
- replyDAO : ReplyDAO
+ getWall(user : Member) : Dictionary
+ likePost(postID : int, userID : String, isLike : boolean) : boolean
+ retrieveLikers(postID : int) : ArrayList<Member>
+ retrieveDislikers(postID : int) : ArrayList<Member>
+ replyPost(postID : int, userID : String, message : String) : boolean
+ retrieveReplies(postID : int) : ArrayList<Reply>
+ deletePost(userID : String, postID : int) : boolean
FriendFarmCtrl
- plotDAO : PlotDAO
- PlotCropDAO : PlotCropDAO
- plotCropStealDAO : PlotCropStealDAO
- cropDAO : CropDAO
+ steal(user : Member, friend : Member) : Map<String,Integer>
+ retrievePlots(user : Member) : ArrayList<Plot>
+ retrievePlot(plotID : int) : PlotCrop
+ calculateProgress(plotcrop : int) : String
RegisterCtrl
- memberDAO : MemberDAO
+ userRegister(userID : String, fullName : String, password : String, cfmPassword : String) : void
StoreCtrl
- cropDAO : CropDAO
- inventoryDAO : InventoryDAO
- memberDAO : MemberDAO
+ getSeedList() : ArrayList<Crop>
+ buySeed(user : Member, seed : Crop, quantity : int) : void
FriendWallCtrl
- memberDAO : MemberDAO
- postDAO : PostDAO
+ getFriendWall(user : Member) : Dictionary
+ getFriendsAndCommon(user : Member, friend : Member) : HashMap<Member,Boolean>
+ newPost(posterID : String, content : String, wall_userID : String) : void
VisitFriendCtrl
- memberDAO : MemberDAO
+ getFriends(user : Member) : ArrayList<Member>
GiftCtrl
- memberDAO : MemberDAO
- inventoryDAO : InventoryDAO
- giftDAO : GiftDAO
+ sendToFriends(sender : Member, nameString : String, seed : Crop) : void
+ acceptGifts(user : Member) : void
WallCtrl
- memberDAO : MemberDAO
- postDAO : PostDAO
+ getWall(user : Member) : Dictionary
+ newPost(posterID : String, content : String, wall_userID : String) : void
+ getGoldRank(user : Member) : int


pkg
SocialMagnet
Menu
FarmlandMenu
- ctrl : FarmlandCtrl
- menuUtility : MenuUtility
+ displayFarmland(user : Member) : boolean
+ displayPlots(user : Member) : Map<Integer,Plot>
+ plantCrop(plot : Plot, user : Member) : String
+ harvest(user : Member) : void
NewsFeedMenu
- ctrl : NewsFeedCtrl
- threadMenu : ThreadMenu
- menuUtility : MenuUtility
+ displayNewsFeed(user : Member) : void
FarmMenu
- menuUtility : MenuUtility
- farmLandMenu : FarmlandMenu
- storeMenu : StoreMenu
- inventoryMenu : InventoryMenu
- visitFriendMenu : VisitFriendMenu
- giftMenu : GiftMenu
+ displayFarm(user : Member) : void
ThreadMenu
- ctrl : ThreadCtrl
+ displayThread(post : Post, index : int, user : Member) : void
+ likePost(post : Post, user : Member, isLike : boolean, index : int) : void
+ retrieveLikers(post : Post) : void
+ retrieveDislikers(post : Post) : void
+ replyPost(post : Post, user : Member) : void
+ retrieveReplies(post : Post, index : int) : void
+ deletePost(user : Member, post : Post) : void
FriendFarmMenu
- ctrl : FriendFarmCtrl
- cropDAO : CropDAO
- rankDAO : RankDAO
- plotDAO : PlotDAO
+ displayFriendFarm(user : Member, friend : Member) : boolean
+ checkMapHasNonZeroSteals(stolenMap : Map<String,Integer>) : boolean
+ checkAllPlotsAlreadyStolen(stolenMap : Map<String,Integer>) : boolean
+ processSteals(user : Member, friend : Member) : void
+ displayPlots(user : Member) : Map<Integer,Plot>
StoreMenu
- ctrl : StoreCtrl
- menuUtility : MenuUtility
+ displayStore(user : Member) : boolean
RegisterMenu
- ctrl : RegisterCtrl
+ displayRegister() : void
FriendMenu
- ctrl : FriendCtrl
- friendWallMenu : FriendWallMenu
+ displayFriends(user : Member) : void
+ sendFriendRequest(senderID : String) : void
+ acceptFriendRequest(user : Member, memberSelected : Member) : void
+ rejectFriendRequest(user : Member, memberSelected : Member) : void
+ unFriend(user : Member, memberSelected : Member) : void
VisitFriendMenu
- menuUtility : MenuUtility
- ctrl : VisitFriendCtrl
- friendFarmMenu : FriendFarmMenu
+ displayVisitFriends(user : Member) : boolean
FriendWallMenu
- ctrl : FriendWallCtrl
- threadMenu : ThreadMenu
- menuUtility : MenuUtility
+ displayFriendWall(user : Member, friend : Member) : void
+ newPost(user : Member, friend : Member) : void
WallMenu
- ctrl : WallCtrl
- giftCtrl : GiftCtrl
- threadMenu : ThreadMenu
- menuUtility : MenuUtility
+ displayWall(user : Member) : void
+ acceptGifts(user : Member) : void
+ newPost(user : Member) : void
+ displayThread(post : Post, index : int, user : Member) : void
GiftMenu
- giftCtrl : GiftCtrl
- storeCtrl : StoreCtrl
- threadMenu : ThreadMenu
- menuUtility : MenuUtility
+ displayGift(user : Member) : boolean
+ sendToFriends(sender : Member, seed : Crop) : void
MainMenu
- menuUtility : MenuUtility
- newsFeedMenu : NewsFeedMenu
- wallMenu : WallMenu
- friendMenu : FriendMenu
- farmMenu : FarmMenu
+ displayMain(user : Member) : void
InventoryMenu
- ctrl : InventoryCtrl
- menuUtility : MenuUtility
+ displayInventory(user : Member) : boolean
WelcomeMenu
- menuUtility : MenuUtility
- registerMenu : RegisterMenu
- loginMenu : LoginMenu
+ displayWelcome() : void
+ start() : void
LoginMenu
- ctrl : LoginCtrl
- mainMenu : MainMenu
+ displayLogin() : void
MenuUtility
+ checkInput() : int
+ displayCityFarmersHeader(category : String, user : Member) : void
+ displayPosts(postDictionary : Dictionary) : void
+ displayFriends(friendsCommonMap : HashMap<Member,Boolean>) : void
+ displayConfirm() : boolean
+ displayGifts(giftsMap : HashMap<String,Integer>) : void


pkg
SocialMagnet
Application
SocialMagnetApplication
+ main(args : String[]) : void
Exceptions
FarmlandException
+ FarmlandException(message : String)
FriendException
+ FriendException(message : String)
GiftException
+ GiftException(message : String)
LoginException
+ LoginException(message : String)
RegistrationException
+ RegistrationException(message : String)
StealException
+ StealException(message : int) : String
StoreException
+ StoreException(message : String)
Utitlity
Gift
- giftID : int
- senderID : String
- receiverID : String
- date : Date
- giftContent : String
- isAccepted : boolean
+ Gift(giftID : int, senderID : String, receiverID : String, date : Date, giftContent : String, isAccepted : boolean)
+ getGiftID() : int
+ getSenderID() : String
+ getReceiverID() : String
+ getDate() : Date
+ getGiftContent() : String
+ getAcceptance() : boolean
+ setAcceptance(isAccepted : boolean) : void
GiftDAO
- postDAO : PostDAO
+ dbMgr(sqlstmt : String, update : boolean) : ArrayList<Gift>
+ retrieveAll() : ArrayList<Gift>
+ hasExceededFiveToday(user : Member) : boolean
+ hasSentToFriendToday(user : Member, friend : Member) : boolean
+ sendGift(user : Member, friend : Member, seed : Crop) : void
+ acceptGifts(user : Member) : ArrayList<Gift>
Nth
+ ordinalAbbrev(n : int) : String

Design Outline

Our team has decided on the final file structure and implementation after taking into account a few other structure considerations. Under src, there are two folders, main and test. In main there are a total of 7 sub folders in java/SocialMagnet:

1) **Application** - Contains only the SocialMagnetApplication.java file which runs the whole application.
2) **Control** - Contains all the control (ctrl) files which are java files that acts as a bridge between the User Interface (Printing into console), and the Data Access Object (DAO) files. Methods in the ctrl files calls the DAO methods and perform checks and validations. Exceptions are thrown in the ctrl files which are caught by the menu.
3) **Exceptions** - Stores the custom made Exceptions which are created to handle the different scenarios in the application.
4) **Farm** - Contains all object java files and its DAO files used in the farm game.
5) **Menu** - Contains each and every Menu used in the application. Menu java files also has methods that calls the methods in the ctrl files. Try-catch blocks are also seen in the Menu files where it handles the exception and displays the corresponding error message.
6) **Social** - This folder is split into two other sub-folders, Member and Post. The Member folder has the Member java file and its DAO file. Similarly, Post folder has Post java file and its DAO file. Additionally, it also stores Reply and ReplyDAO
7) **Utility** - Contains java files which belongs in both Farm and Social side. Gift is used in both Social and Farm. It will be displayed on a user's wall whenever a gift is sent, and a gift is sent through the Farm game where it will make changes to the inventory in the user's farm. Nth.java retrieves the ranking of the user through the farm game and displays it on the user's wall, as such it is in Utility instead of Social or Farm.

Design Considerations

The overall usage of Menus, Ctrls, DAOs & Exceptions were used in reference to In-Class Exercises done previously in the module - ICE 8: Sequence Diagrams. We felt that this method made it easy to relate the user's flow of actions with our code. Below are principles that we considered but may have not implemented:

1. **Single Responsibility Principle**
   Our implementation of Social Magnet implemented the Single Responsibility Principle. Most functions are broken down to do one thing. For example, when a user logs in to Social Magnet, the MemberDAO class assumes the responsibility of retrieving the user information from the Db only. From there the LoginControl class assumes responsibility of processing the user input information. If there is an error, LoginException throws an error. The menu classes then assume the responsibility of displaying the processed information.
2. **Open/Closed Principle**
   We failed to fully implement the open/closed principle. In each Menu, users were asked to input a letter of response. We used a switch-case in each menu to handle their inputs. This was done as the input requirements on each menu were already clearly defined. Although following this principle would make it easier to extend the different inputs in each page, our team decided not to and went with a switch-case in each menu.
3. **Dependency Inversion Principle**
   By using Ctrls & DAOs, we removed the dependency from high level modules to low level modules. In each menu, functions call methods and retrieves a response. However, the details of the response are abstract. As long as the right input goes in, the right output comes out.
4. **Don't repeat yourself (DRY)**
   To some extent, we made our functions atomic, especially in our DAOs. This ensures that there are no 2 repeating methods that retrieve the exact same thing. For example, in our MemberDAO, we have the getUser() function that is reusable. Many different sources call this getUser function - RegisterCtrl, LoginCtrl, etc. There were no duplicate functions of retrieving a user from the Db.

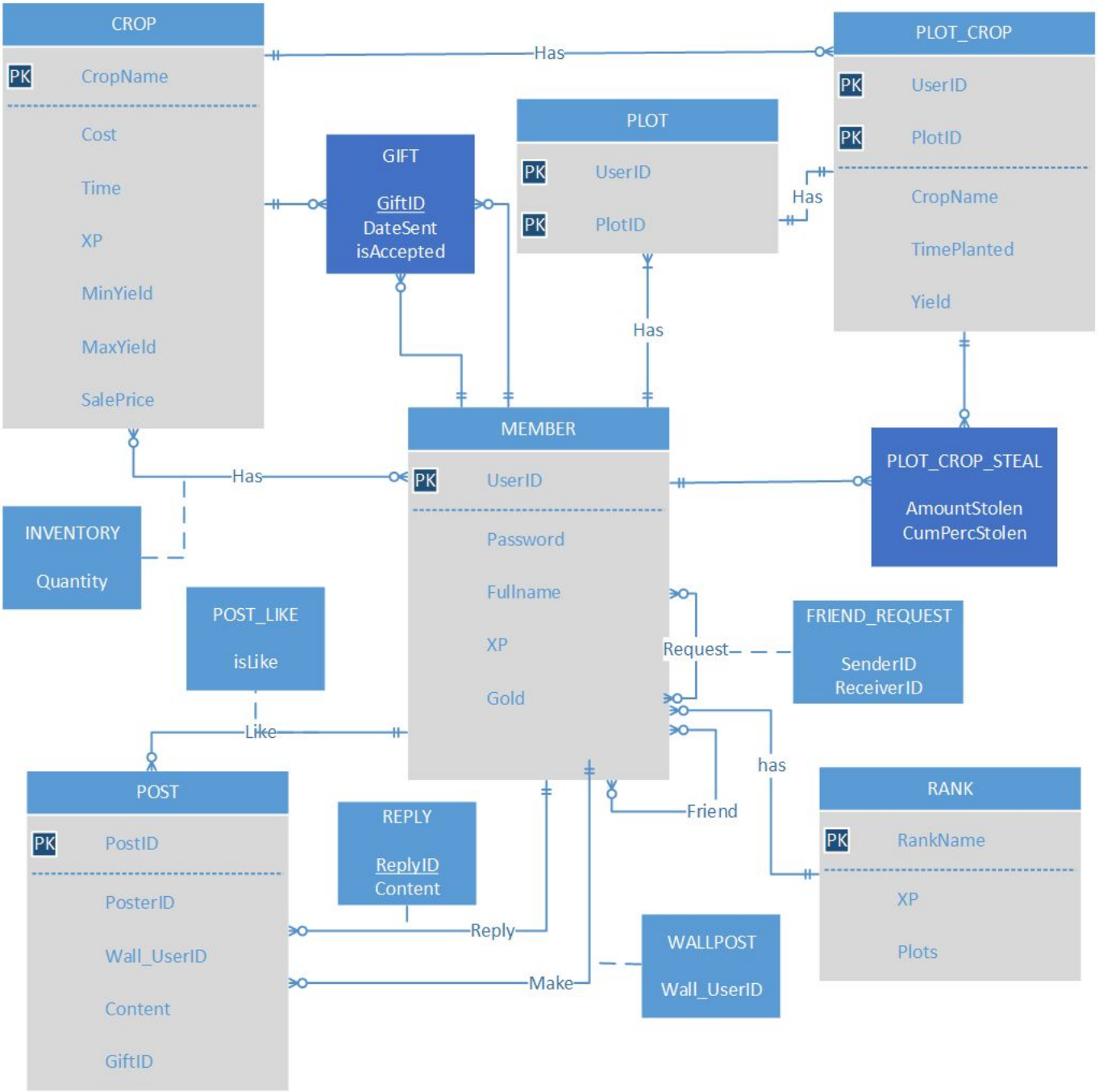
Since we did not implement any interfaces or inheritance, Liskov's substitution principle and the Interface segregation principle were not considered.

Improvements for the Future

Our team felt that although we completed the project, there were many ways which we could have gone through it better.

- Abstraction was a concept that was new to us in OOP. We would have liked to incorporate more abstraction - interfaces - that would encapsulate other software design principles. For e.g - Post,Reply & Like implementing an Interaction Interface.
- JUnit was also something new and unfamiliar. We should have dove into it earlier.

ER Model


CROP
PK CropName
Cost
Time
XP
MinYield
MaxYield
SalePrice
Has
PLOT_CROP
PK UserID
PK PlotID
CropName
TimePlanted
Yield
GIFT
GiftID
DateSent
isAccepted
PLOT
PK UserID
PK PlotID
Has
Has
MEMBER
PK UserID
Password
Fullname
XP
Gold
Has
PLOT_CROP_STEAL
AmountStolen
CumPercStolen
INVENTORY
Quantity
POST_LIKE
isLike
Like
Request
FRIEND_REQUEST
SenderID
ReceiverID
has
Friend
POST
PK PostID
PosterID
Wall_UserID
Content
GiftID
REPLY
ReplyID
Content
Reply
Make
WALLPOST
Wall_UserID
RANK
PK RankName
XP
Plots

ER Model - Relation Table Explanation

The above model depicts how our database is designed. **Crop** and **Rank** are tables created based on the 2 csv files given to us.

Member - Stores the information required for each user, as well as their XP and Gold for the Farmland game. As seen from the diagram above, Member is connected to almost everything as any actions taken on the application is done to be done by the member.

Friend - Friend will be converted into a table in the database from the unary relationship in Member. It will stores the two UserIDs of the members who are friends

Friend_Request - Stores the friend request made by each user. Stores two UserIDs, the one sending the friend request and the one receiving the friend request.

Post - Stores information about each Post, its contents, the ID of the member who made the post, and the userID of who's wall the post will be on. It also has a GiftID where if it is null, it would mean that it is a normal post, and will be displayed on the newsfeed. If GiftID has an ID attached, it will not be displayed on the newsfeed.

Member has 3 relationships with Post, the Member can make, reply or like a post. Each of these relationships has its own attributes as well.
i) For make, the user creates a Post and when the post is created, it will store in another table, **Wallpost**, that stores the UserID of the Members that will have the newly created posted on their walls. As one post can appear on multiple walls due to tagging, it will be necessary to have this table in the database.
ii) The reply relationship will be converted into a **Reply** table in the database. A post can have many replies by different members and a member can make any number of replies to each post. As such, the Reply table stores the ReplyID and the content of the reply for each post and by which member.
iii) The like relationship will also be converted into a database table, **Like.** Like stores which member liked/disliked which post. isLike is a boolean variable that when is false would mean that the user dislikes, and true would mean that the user likes the post.

Inventory - Each member will have its own Inventory that stores the amount of crops they have. It takes note of the quantity of each crop they have in their possession.

Gift - Gift is an associative entity in the ER model that stores the UserIDs of two members, the one sending the gift and the one receiving the gift, as well as the crop that is being sent as a gift.

Plot - Each member will have the number of plots assigned to them based on their rank in Farmland. The UserID and the PlotID acts as a primary key for this table.

Plot_Crop - This table stores the crop that is being planted on each plot, the time it was planted and the amount of yield the user will get. The yield will be randomly generated based on the min and max yield and stored into the database once the crop has been planted. The yield will be invisible to the member, but it is required for us to be able to keep track of the 20% that can be stolen once the crops are ready for harvest.

Plot_Crop_Steal - This associative entity has 2 relationships. One towards the Member and one towards the Plot_Crop. The relationships to Member will be converted into StealerID which is the UserID of the one stealing and it will store the plot_crop that the user is stealing from.