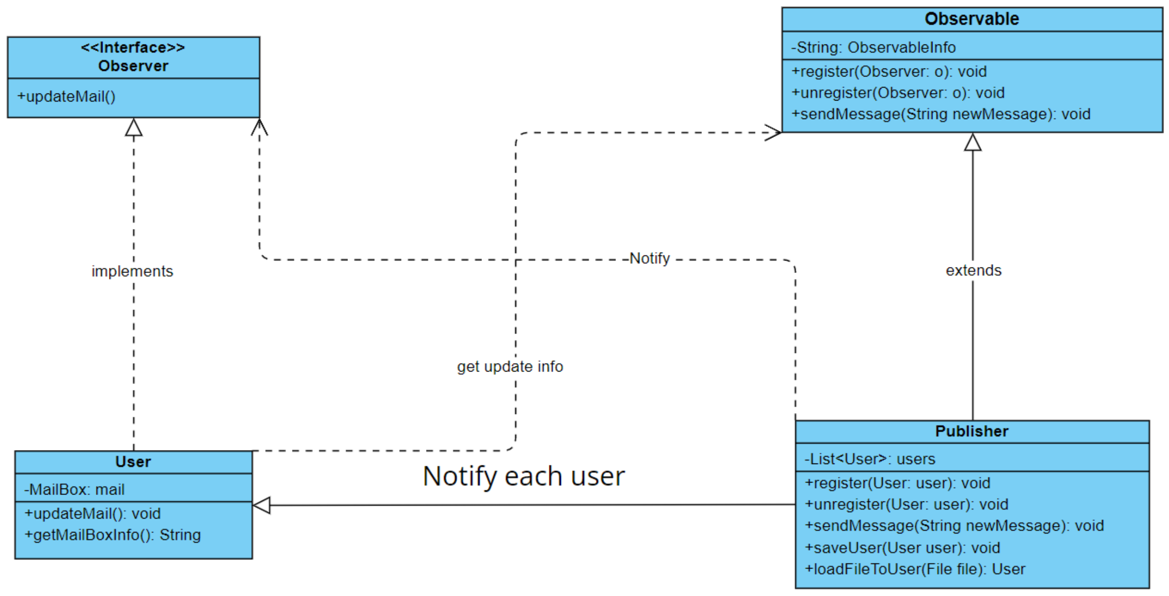
Design Patterns

Ziqi Zheng, Jianxin Liu, MingRen Liu, Sam Zhang

Content:

Observer Pattern

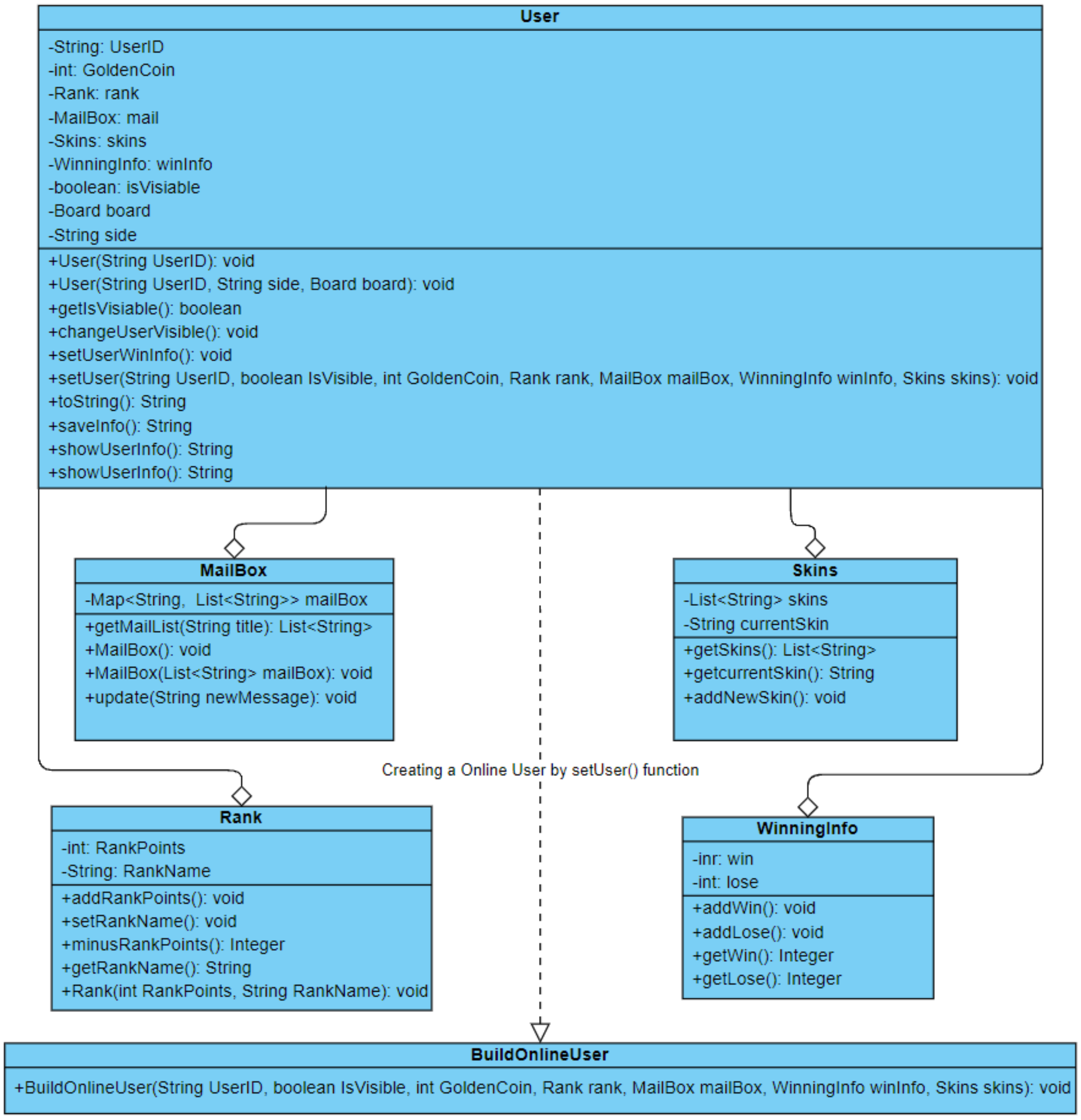
We will create a announcement part for developers publish update information. The User class should implements Observer interface for updating news to their mail box. Developer extends the Observable class that allow Developer register new user and remove the user who wants to delete his account. And once developers update a news, all users can be notified by notify function of Developer.



<<Interface>>
Observer
+updateMail()
Observable
-String: ObservableInfo
+register(Observer: o): void
+unregister(Observer: o): void
+sendMessage(String newMessage): void
implements
Notify
extends
get update info
User
-MailBox: mail
+updateMail(): void
+getMailBoxInfo(): String
Notify each user
Publisher
-List<User>: users
+register(User: user): void
+unregister(User: user): void
+sendMessage(String newMessage): void
+saveUser(User user): void
+loadFileToUser(File file): User

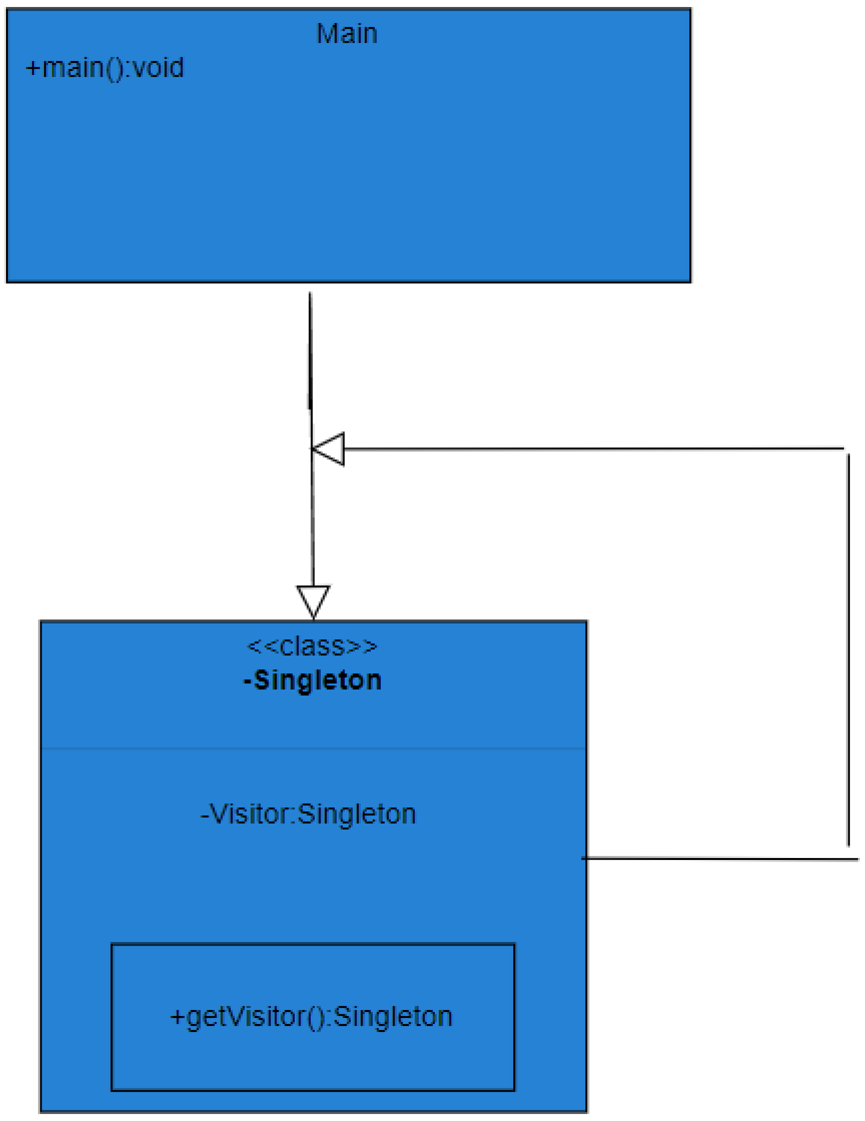
Builder Pattern

The builder pattern is used for creating User instance. A User class should include UserID, GoldenCoin, rank, mail, skins, winInfo, isVisible, board, side attributes. The UserID is a unique ID for each user and it is a String type. The rank is a instance of class Rank and class Rank includes current user's rank points and rank name. Such as rank points(1600) represents "GOLD". The MailBox is a class that use Map to save mail information. The key of a Map is the title of a piece of information which is a String type. The value of a Map is used for storing list of messages. The Skins is a class for storing user owned skins information and which skin is being used by user currently. WinningInfo is a class that shows how many games that user already won and lost. The board and side can show the side and current situation of board for user. Besides, there is a class called BuildOnlineUser which can create a instance of a user who is playing Human VS Human mode currently.


User
-String: UserID
-int: GoldenCoin
-Rank: rank
-MailBox: mail
-Skins: skins
-WinningInfo: winInfo
-boolean: isVisiable
-Board board
-String side
+User(String UserID): void
+User(String UserID, String side, Board board): void
+getIsVisiable(): boolean
+changeUserVisible(): void
+setUserWinInfo(): void
+setUser(String UserID, boolean IsVisible, int GoldenCoin, Rank rank, MailBox mailBox, WinningInfo winInfo, Skins skins): void
+toString(): String
+saveInfo(): String
+showUserInfo(): String
+showUserInfo(): String
MailBox
-Map<String, List<String>> mailBox
+getMailList(String title): List<String>
+MailBox(): void
+MailBox(List<String> mailBox): void
+update(String newMessage): void
Skins
-List<String> skins
-String currentSkin
+getSkins(): List<String>
+getcurrentSkin(): String
+addNewSkin(): void
Creating a Online User by setUser() function
Rank
-int: RankPoints
-String: RankName
+addRankPoints(): void
+setRankName(): void
+minusRankPoints(): Integer
+getRankName(): String
+Rank(int RankPoints, String RankName): void
WinningInfo
-inr: win
-int: lose
+addWin(): void
+addLose(): void
+getWin(): Integer
+getLose(): Integer
BuildOnlineUser
+BuildOnlineUser(String UserID, boolean IsVisible, int GoldenCoin, Rank rank, MailBox mailBox, WinningInfo winInfo, Skins skins): void

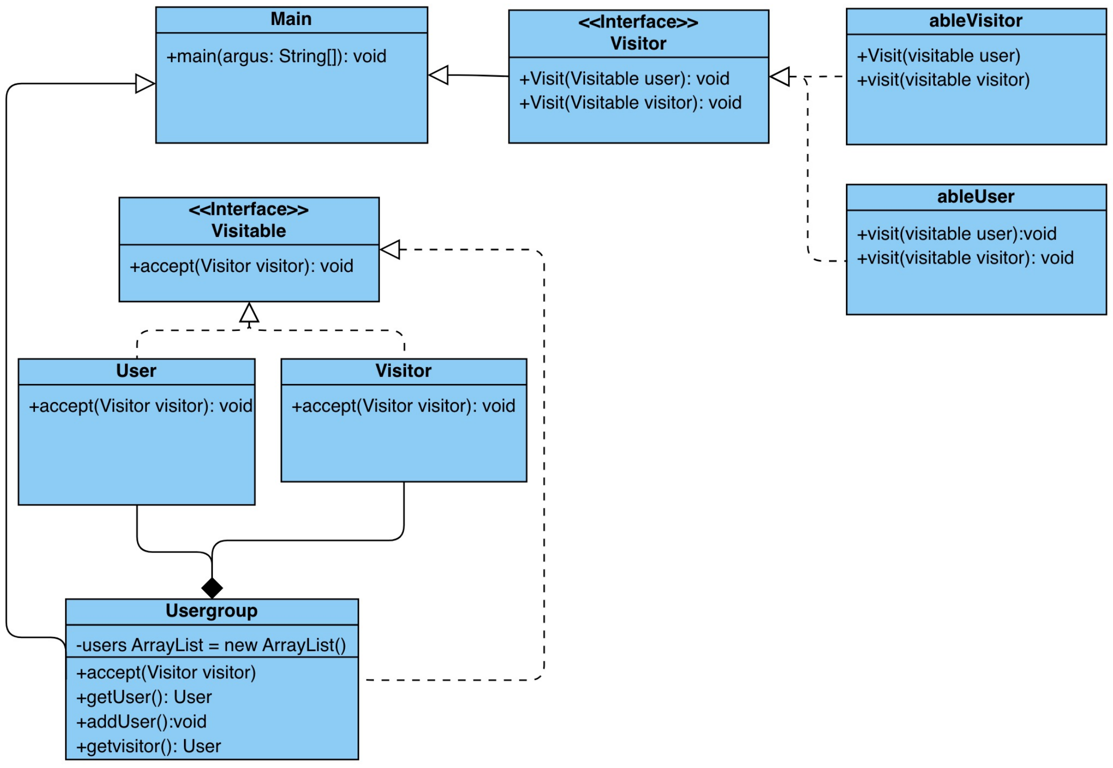
Singleton Pattern

For those user who are not willing to register account, they will be treated as visitor. Visitors can only experience the game, the result of the game cannot be counted in the ranking, and they cannot get gold coins and points. The purpose of singleton is to ensure that in a process,a certain class has and only one instance. Visitor's coin and point will always be none,because we won't save and update their information. So we don’t need to consider visitors when we need to create users frequently, Therefore, system resources can be saved and system performance can be improved.



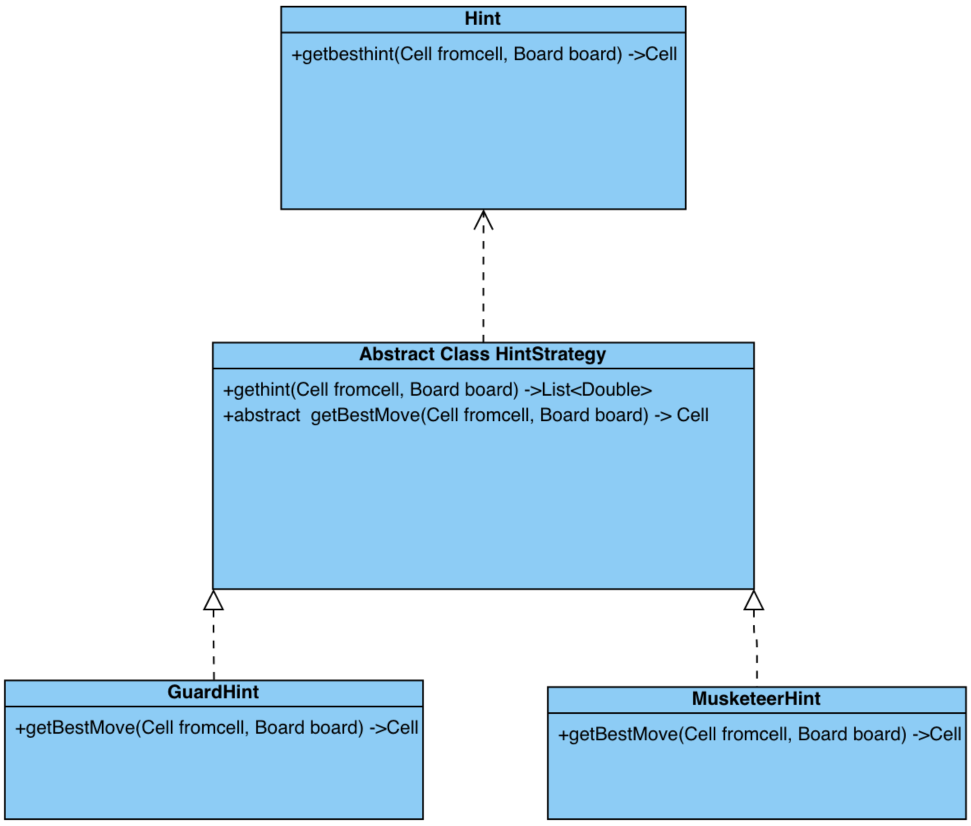
Visitor Pattern

In the Visitor Pattern, we use a visitor class that changes the execution algorithm of the element class. In this way, the execution algorithm of the element can change as the visitor changes. if the user type(user or visitor) can access some game data for example shops and person information UI which user can but visitor cannot. The getUser() and getVisitor() is from Class User and Class visitor. Then Store the User into an arraylist for further process.


Main
+main(argus: String[]): void
<<Interface>> Visitor
+Visit(Visitable user): void
+Visit(Visitable visitor): void
ableVisitor
+Visit(visitable user)
+visit(visitable visitor)
ableUser
+visit(visitable user):void
+visit(visitable visitor): void
<<Interface>> Visitable
+accept(Visitor visitor): void
User
+accept(Visitor visitor): void
Visitor
+accept(Visitor visitor): void
Usergroup
-users ArrayList = new ArrayList()
+accept(Visitor visitor)
+getUser(): User
+addUser():void
+getvisitor(): User

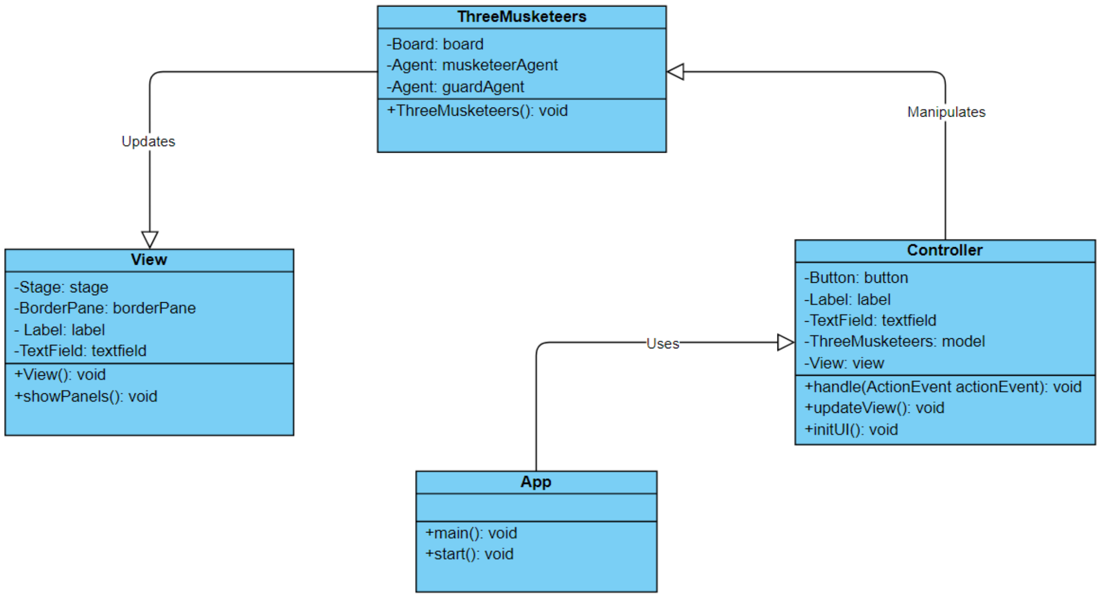
Strategy Pattern

In strategy Pattern,we define a series of algorithms, encapsulate them one by one, the behavior of a class or its algorithm can be changed while program is running. In the hint method, there are two type output for the Cell value. One is based on the Musketeer another is Guard. The Musketeer need return the Max value of BoardEvaluation, The Guard need return the Min value of BoardEvaluation. So the Using the Strategy Pattern can easily change while using the Hint method.


Hint
+getbesthint(Cell fromcell, Board board) ->Cell
Abstract Class HintStrategy
+gethint(Cell fromcell, Board board) ->List<Double>
+abstract getBestMove(Cell fromcell, Board board) -> Cell
GuardHint
+getBestMove(Cell fromcell, Board board) ->Cell
MusketeerHint
+getBestMove(Cell fromcell, Board board) ->Cell

MVC Pattern

This is the Model Control View Pattern, one of the most crucial components of the game. Using this pattern, we will create a GUI and link all the other features of said GUI. The main application uses the controller class where it will get data from the user. Once the user gives the data, the controller will use that data and manipulates the Threemusketeers model accordingly. Which then the view class will change the GUI to reflect the changes.


ThreeMusketeers
-Board: board
-Agent: musketeerAgent
-Agent: guardAgent
+ThreeMusketeers(): void
Updates
Manipulates
View
-Stage: stage
-BorderPane: borderPane
- Label: label
-TextField: textfield
+View(): void
+showPanels(): void
Controller
-Button: button
-Label: label
-TextField: textfield
-ThreeMusketeers: model
-View: view
+handle(ActionEvent actionEvent): void
+updateView(): void
+initUI(): void
Uses
App
+main(): void
+start(): void