# Design Blackjack and a Deck of Cards

Let's design a game of Blackjack.

Blackjack is the most widely played casino game in the world. It falls under the category of comparing-card games and is usually played between several players and a dealer. Each player, in turn, competes against the dealer, but players do not play against each other. In Blackjack, all players and the dealer try to build a hand that totals 21 points without going over. The hand closest to 21 wins.


## System Requirements

Blackjack is played with one or more standard 52-card decks. The standard deck has 13 ranks in 4 suits.

**Background**

- To start with, the players and the dealer are dealt separate hands. Each hand has two cards in it.
- The dealer has one card exposed (the up card) and one card concealed (the hole card), leaving the player with incomplete information about the state of the game.
- The player's objective is to make a hand that has more points than the dealer, but less than or equal to 21 points.
- The player is responsible for placing bets when they are offered, and taking additional cards to complete their hand.
- The dealer will draw additional cards according to a simple rule: when the dealer's hand is 16 or less, they will draw cards (called a hit), when it is 17 or more, they will not draw additional cards (or stand pat).

**Points calculation**

Blackjack has different point values for each of the cards:

- The number cards (2-10) have the expected point values.

- The face cards (Jack, Queen, and King) all have a value of 10 points.
- The Ace can count as one point or eleven points. Because of this, an Ace and a 10 or face card totals 21. This two-card winner is called “blackjack”.
- When the points include an ace counting as 11, the total is called soft-total; when the ace counts as 1, the total is called hard-total. For example, A+5 can be considered a soft 16 or a hard 6.

**Gameplay**

1. The player places an initial bet.
2. The player and dealer are each dealt a pair of cards.
3. Both of the player’s cards are face up, the dealer has one card up and one card down.
4. If the dealer’s card is an ace, the player is offered insurance.

Initially, the player has a number of choices:

- If the two cards are the same rank, the player can elect to split into two hands.
- The player can double their bet and take just one more card.
- The more typical scenario is for the player to take additional cards (a hit ) until either their hand totals more than 21 (they bust ), or their hand totals exactly 21, or they elect to stand.

If the player’s hand is over 21, their bet is resolved immediately as a loss. If the player’s hand is 21 or less, it will be compared to the dealer’s hand for resolution.

**Dealer has an Ace.** If the dealer’s up card is an ace, the player is offered an insurance bet. This is an additional proposition that pays 2:1 if the dealer’s hand is exactly 21. If this insurance bet wins, it will, in effect, cancel the loss of the initial bet. After offering insurance to the player, the dealer will check their hole card and resolve the insurance bets. If the hole card is a 10-point card, the dealer has blackjack, the card is revealed, and insurance bets are paid. If the hole card is not a 10-point card, the insurance bets are lost, but the card is not revealed.

**Split Hands.** When dealt two cards of the same rank, the player can split the cards to create two hands. This requires an additional bet on the new hand. The dealer will deal an additional card to each new hand, and the hands are played independently. Generally, the typical scenario described above applies to each of these hands.

**Bets**

- Ante: This is the initial bet and is mandatory to play.
- Insurance: This bet is offered only when the dealer shows an ace. The amount must be half the ante.
- Split: This can be thought of as a bet that is offered only when the player’s hand has two cards of equal rank. The amount of the bet must match the original ante.
- Double: This can be thought of as a bet that is offered instead of taking an ordinary hit. The amount of the bet must match the original ante.

## Use case diagram

We have two main Actors in our system:

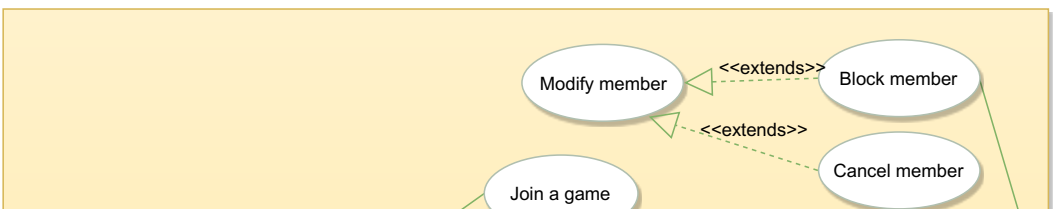
- **Dealer:** Mainly responsible for dealing cards and game resolution.
- **Player:** Places the initial bets, accepts or declines additional bets - including insurance, and splits

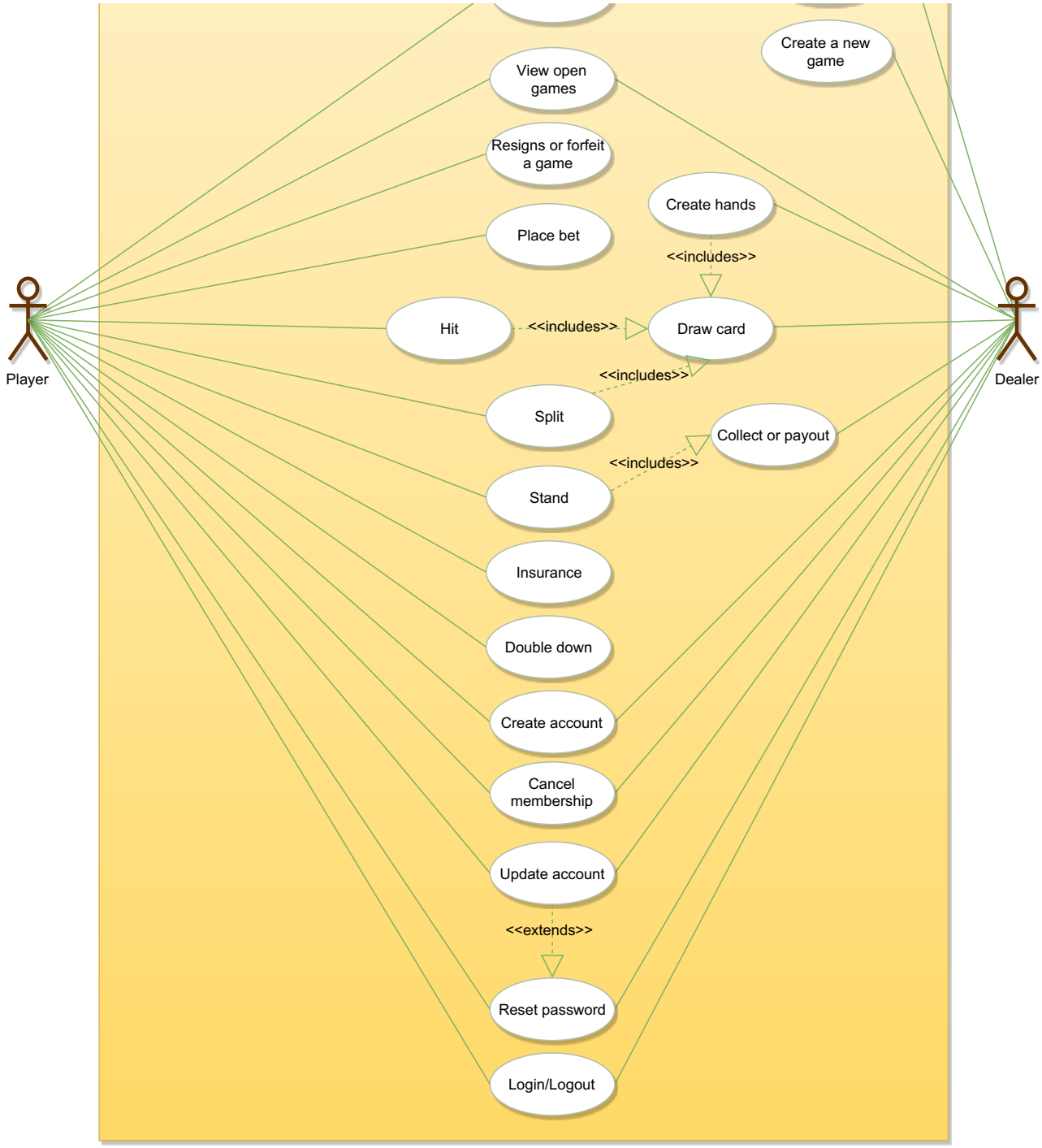
hands. Accepts or rejects the offered resolution, including even money. Chooses among hit, double and stand pat options.

**Typical Blackjack Game Use cases**

Here are the top use cases of the Blackjack game:

- **Create Hands:** Initially both the player and the dealer are given two cards each. The player has both cards visible whereas only one card of the dealer's hand is visible to the player.
- **Place Bet:** To start the game, the player has to place a bet.
- **Player plays the hand:** If the hand is under 21 points, the player has three options:
    - Hit: The hand gets an additional card and this process repeats.
    - Double Down: The player creates an additional bet, and the hand gets one more card and play is done.
    - Stands Pat: If the hand is 21 points or over, or the player chooses to stand pat, the game is over.
    - Resolve Bust. If a hand is over 21, it is resolved as a loser.
- **Dealer plays the hand:** The dealer keeps getting a new card if the total point value of the hand is 16 or less, and stops dealing cards at the point value of 17 or more.
    - Dealer Bust: If the dealer's hand is over 21, the player's wins the game. Player Hands with two cards totaling 21 ( "blackjack" ) are paid 3:2, all other hands are paid 1:1.
- **Insurance:** If the dealer's up card is an Ace, then the player is offered insurance:
    - Offer Even Money: If the player's hand totals to a soft 21, a blackjack; the player is offered an even money resolution. If the player accepts, the entire game is resolved at this point. The ante is paid at even money; there is no insurance bet.
    - Offer Insurance: The player is offered insurance, which they can accept by creating a bet. For players with blackjack, this is the second offer after even money is declined. If the player declines, there are no further insurance considerations.
    - Examine Hole Card: The dealer's hole card is examined. If it has a 10-point value, the insurance bet is resolved as a winner, and the game is over. Otherwise, the insurance is resolved as a loser, the hole card is not revealed, and play continues.
- **Split:** If the player's hand has both cards of equal rank, the player is offered a split. The player accepts by creating an additional Bet. The original hand is removed; The two original cards are split and then the dealer deals two extra cards to create two new Hands. There will not be any further splitting.
- **Game Resolution:** The Player's Hand is compared against the Dealer's Hand, and the hand with the higher point value wins. In the case of a tie, the bet is returned. When the player wins, a winning hand with two cards totaling 21 ("blackjack") is paid 3:2, any other winning hand is paid 1:1.





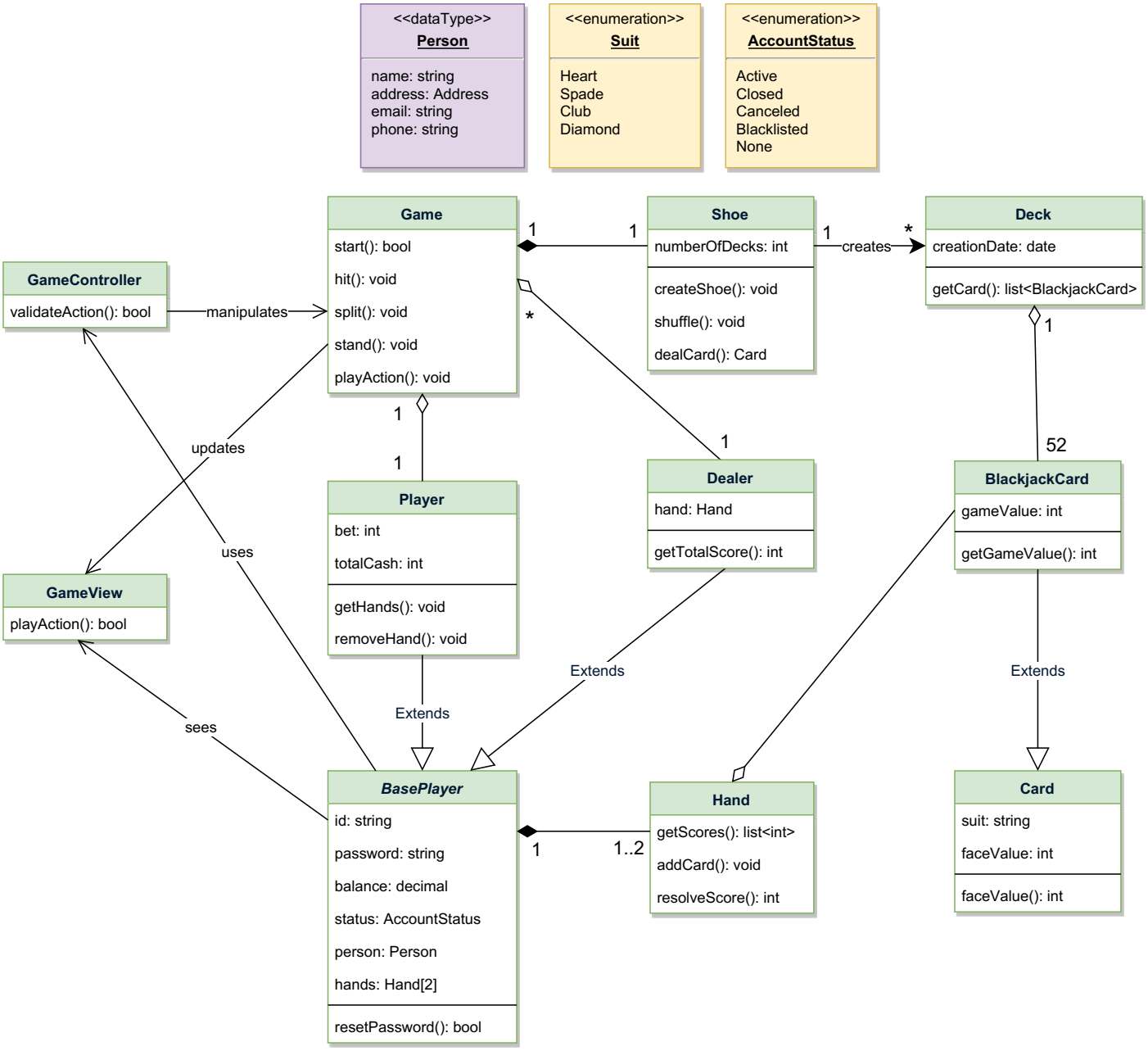

Use case diagram

## Class diagram

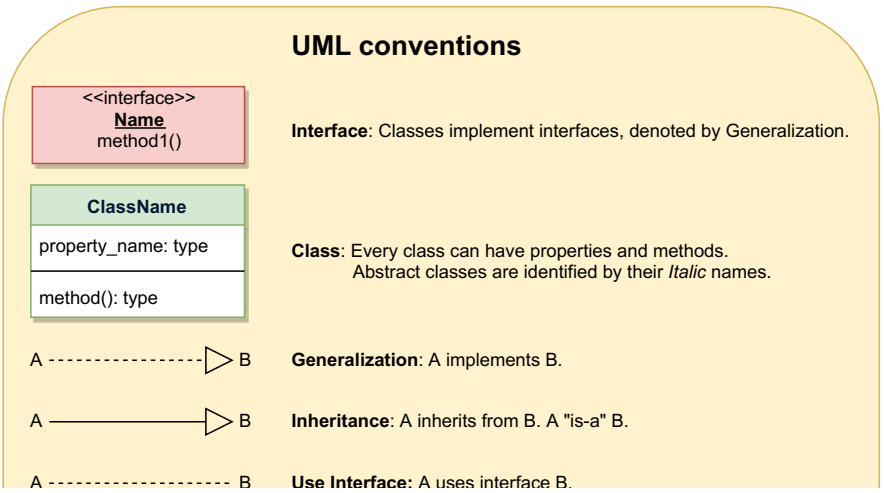
Here are the main classes of our Blackjack game:

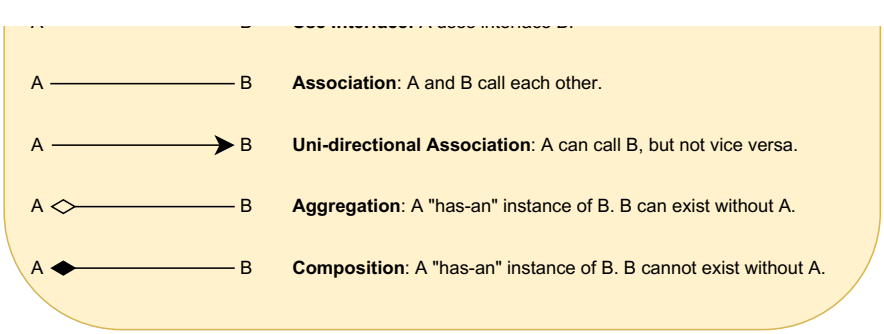
- **Card:** A standard playing card has a suit and point value from 1 to 11.
- **BlackjackCard:** In blackjack, cards have different face values. For example, jack, queen, and king, all have a face value of 10. An ace can be counted as either 1 or 11.
- **Deck:** A standard playing card deck has 52 cards and 4 suits.
- **Shoe:** Contains a set of decks. In casinos, a dealer's shoe is a gaming device to hold multiple decks of playing cards.
- **Hand:** A collection of cards with one or two point values: a hard value (when an ace counts as 1) and a soft value (when an ace counts as 11).
- **Player:** Places the initial bets, updates the stake with amounts won and lost, Accepts or declines

- **Player:** Places the initial bets, updates the stake with amounts won and lost. Accepts or declines offered additional bets - including insurance, and split hands. Accepts or declines offered resolution, including even money. Chooses between hit, double and stand options.
- **Game:** This class encapsulates the basic sequence of play. It runs the game, offers bets to players, deals the cards from the shoe to hands, updates the state of the game, collects losing bets, pays winning bets, splits hands, and responds to player choices of a hit, double or stand.



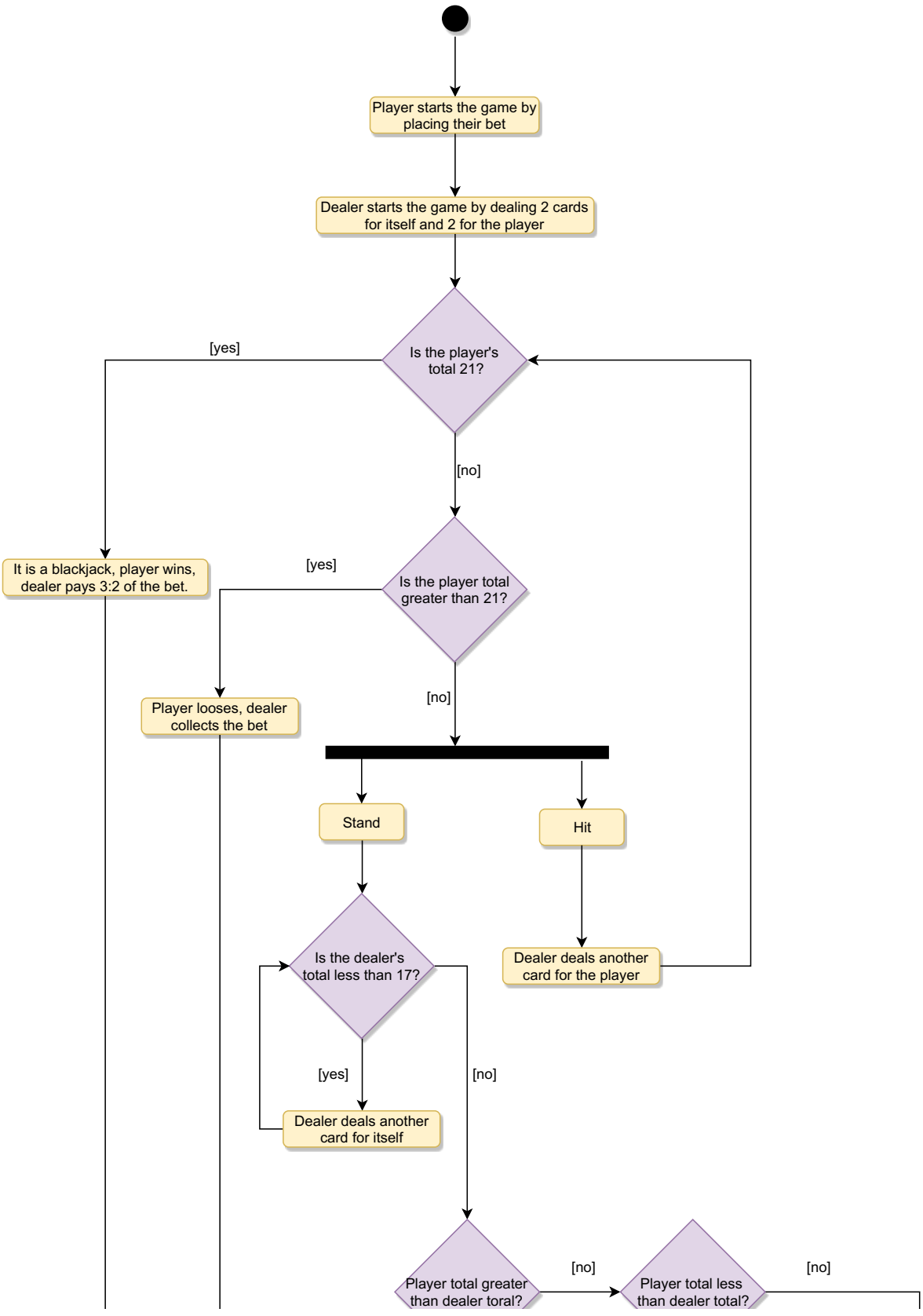

Class diagram

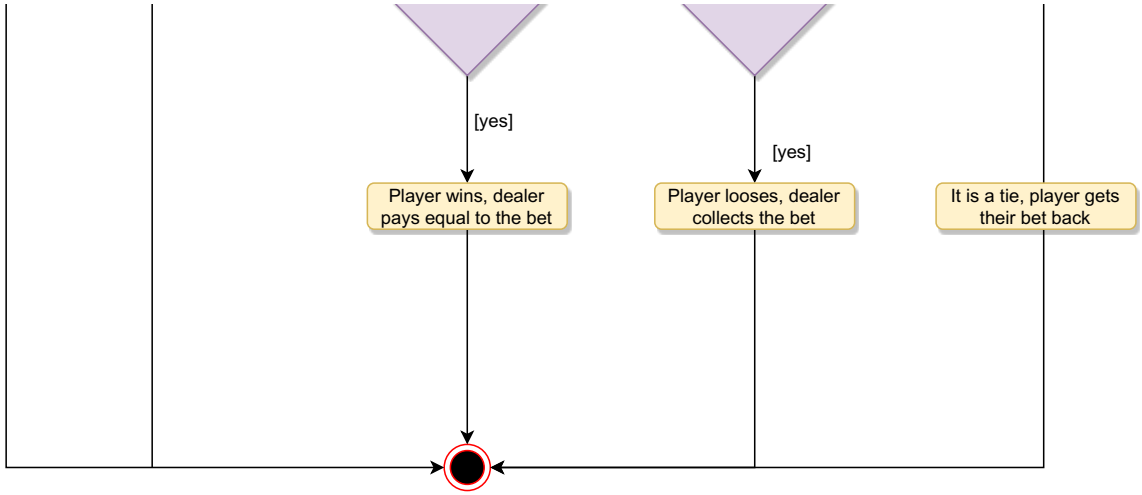





## Activity diagrams

**Blackjack hit or stand:** Here are the set of steps to play blackjack with hit or stand:






## Code

**Enums:** Here are the required enums:

```
1
2
3
```

**Card:** The following class encapsulates a playing card:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
```

**BlackjackCard:** BlackjackCard extends from Card class to represent a blackjack card:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

```
16
```

**Deck and Shoe:** Shoe contains cards from multiple decks:

```
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
```

**Hand:** Hand class encapsulates a blackjack hand which can contain multiple cards:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
```

**Player:** Player class extends from BasePlayer:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
```

**Game:** This class encapsulates a blackjack game:

```
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
```

Mark as Completed

← Back

Design an Airline Management System

Next →

Design a Hotel Management System

Send feedback

6 Recommendations

Stuck? Get help on DISCUSS

# Design a Hotel Management System

Let's design a hotel management system.

A Hotel Management System is a software built to handle all online hotel activities easily and safely. This System will give the hotel management power and flexibility to manage the entire system from a single online portal. The system allows the manager to keep track of all the available rooms in the system as well as to book rooms and generate bills.


## System Requirements

We'll focus on the following set of requirements while designing the Hotel Management System:

1. The system should support the booking of different room types like standard, deluxe, family suite, etc.
2. Guests should be able to search the room inventory and book any available room.
3. The system should be able to retrieve information, such as who booked a particular room, or what rooms were booked by a specific customer.
4. The system should allow customers to cancel their booking - and provide them with a full refund if the cancelation occurs before 24 hours of the check-in date.
5. The system should be able to send notifications whenever the booking is nearing the check-in or check-out date.
6. The system should maintain a room housekeeping log to keep track of all housekeeping tasks.
7. Any customer should be able to add room services and food items.
8. Customers can ask for different amenities.
9. The customers should be able to pay their bills through credit card, check or cash.

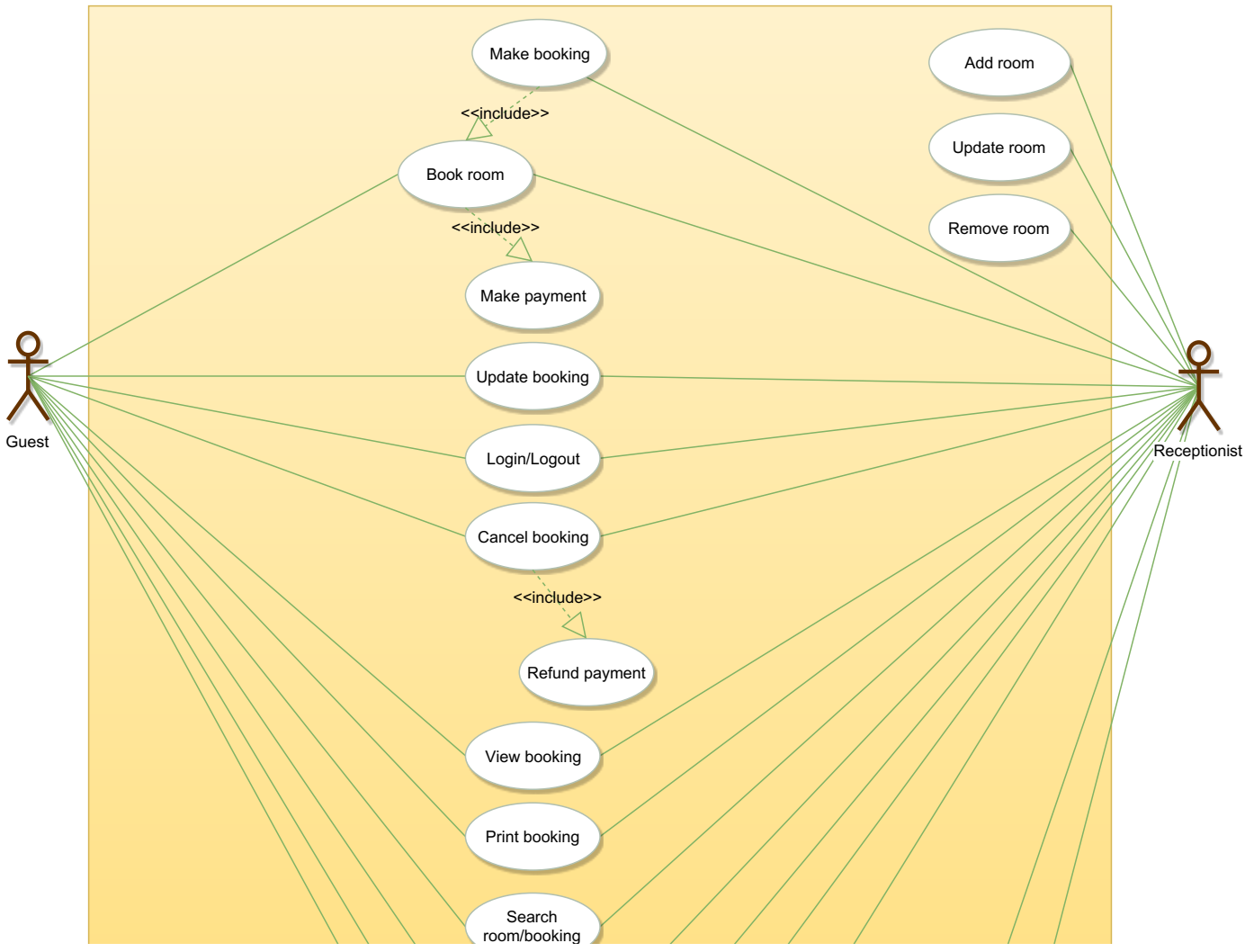
## Use case diagram

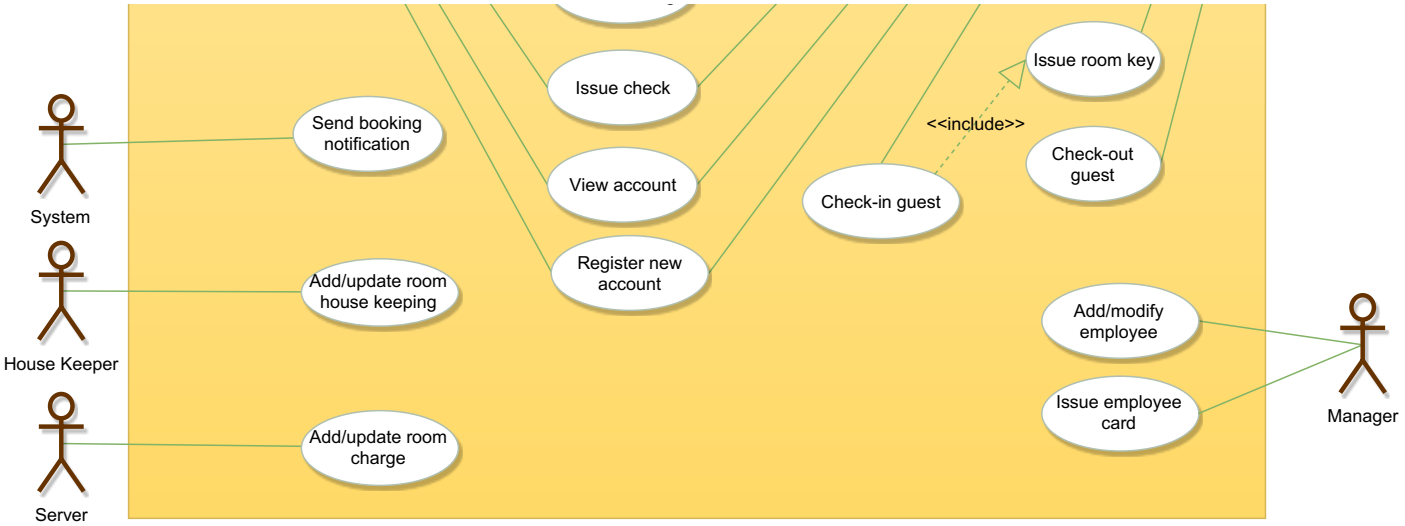
Here are the main Actors in our system:

- **Guest:** All guests can search the available rooms, as well as make a booking.
- **Receptionist:** Mainly responsible for adding and modifying rooms, creating room bookings, check-in, and check-out customers.
- **System:** Mainly responsible for sending notifications for room booking, cancellation, etc.
- **Manager:** Mainly responsible for adding new workers.
- **Housekeeper:** To add/modify housekeeping record of rooms.
- **Server:** To add/modify room service record of rooms.

Here are the top use cases of the Hotel Management System:

- **Add/Remove/Edit room:** To add, remove, or modify a room in the system.
- **Search room:** To search for rooms by type and availability.
- **Register or cancel an account:** To add a new member or cancel the membership of an existing member.
- **Book room:** To book a room.
- **Check-in:** To let the guest check-in for their booking.
- **Check-out:** To track the end of the booking and the return of the room keys.
- **Add room charge:** To add a room service charge to the customer's bill.
- **Update housekeeping log:** To add or update the housekeeping entry of a room.



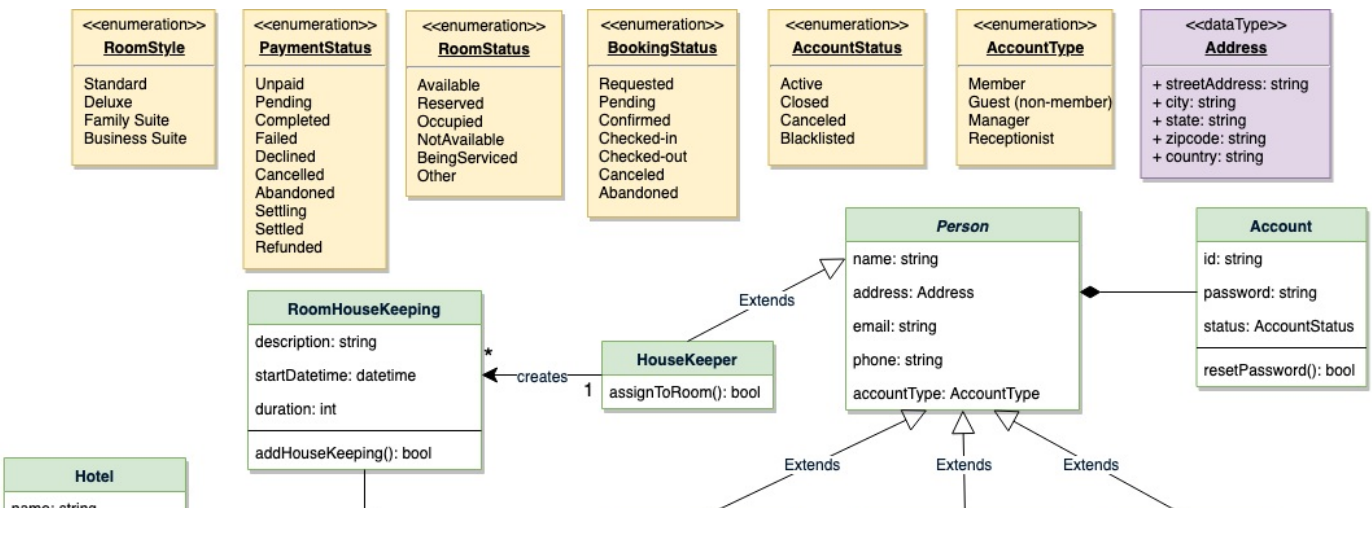


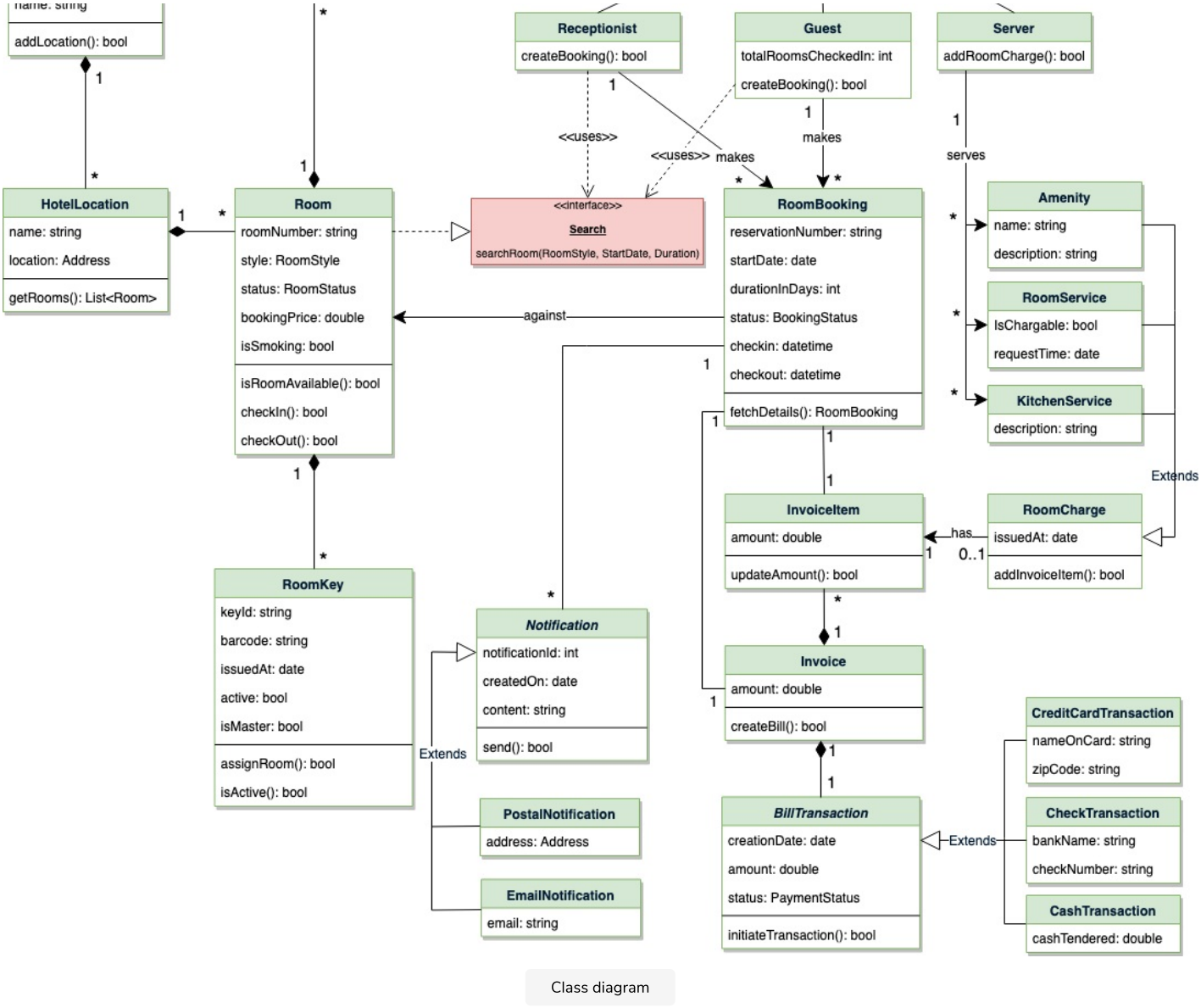

Use case diagram

## Class diagram

Here are the main classes of our Hotel Management System:

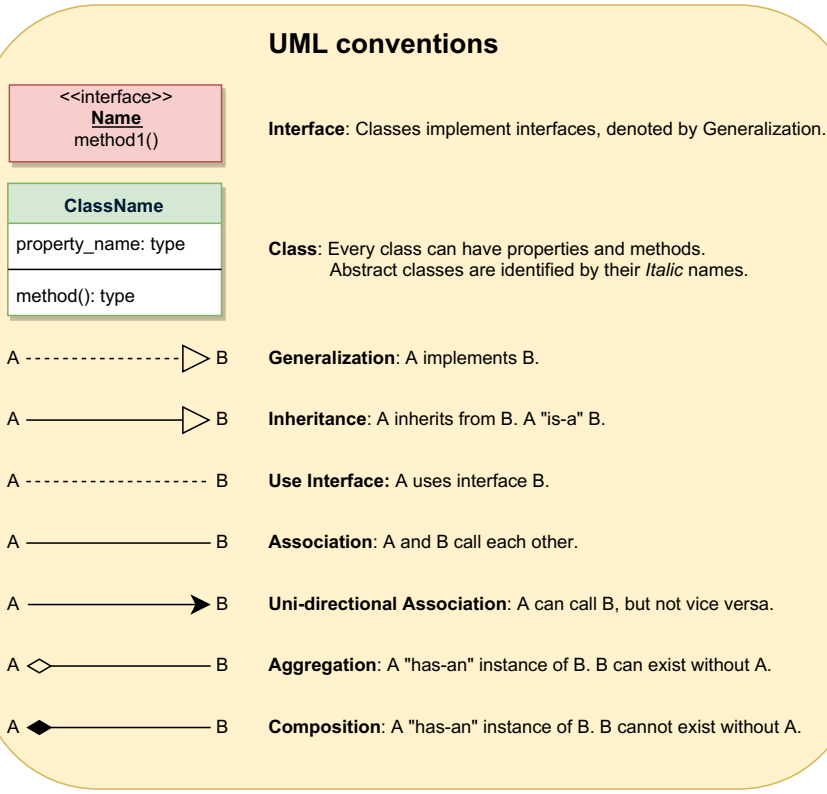
- **Hotel and HotelLocation:** Our system will support multiple locations of a hotel.
- **Room:** The basic building block of the system. Every room will be uniquely identified by the room number. Each Room will have attributes like Room Style, Booking Price, etc.
- **Account:** We will have different types of accounts in the system: one will be a guest to search and book rooms, another will be a receptionist. Housekeeping will keep track of the housekeeping records of a room, and a Server will handle room service.
- **RoomBooking:** This class will be responsible for managing bookings for a room.
- **Notification:** Will take care of sending notifications to guests.
- **RoomHouseKeeping:** To keep track of all housekeeping records for rooms.
- **RoomCharge:** Encapsulates the details about different types of room services that guests have requested.
- **Invoice:** Contains different invoice-items for every charge against the room.
- **RoomKey:** Each room can be assigned an electronic key card. Keys will have a barcode and will be uniquely identified by a key-ID.






Class diagram

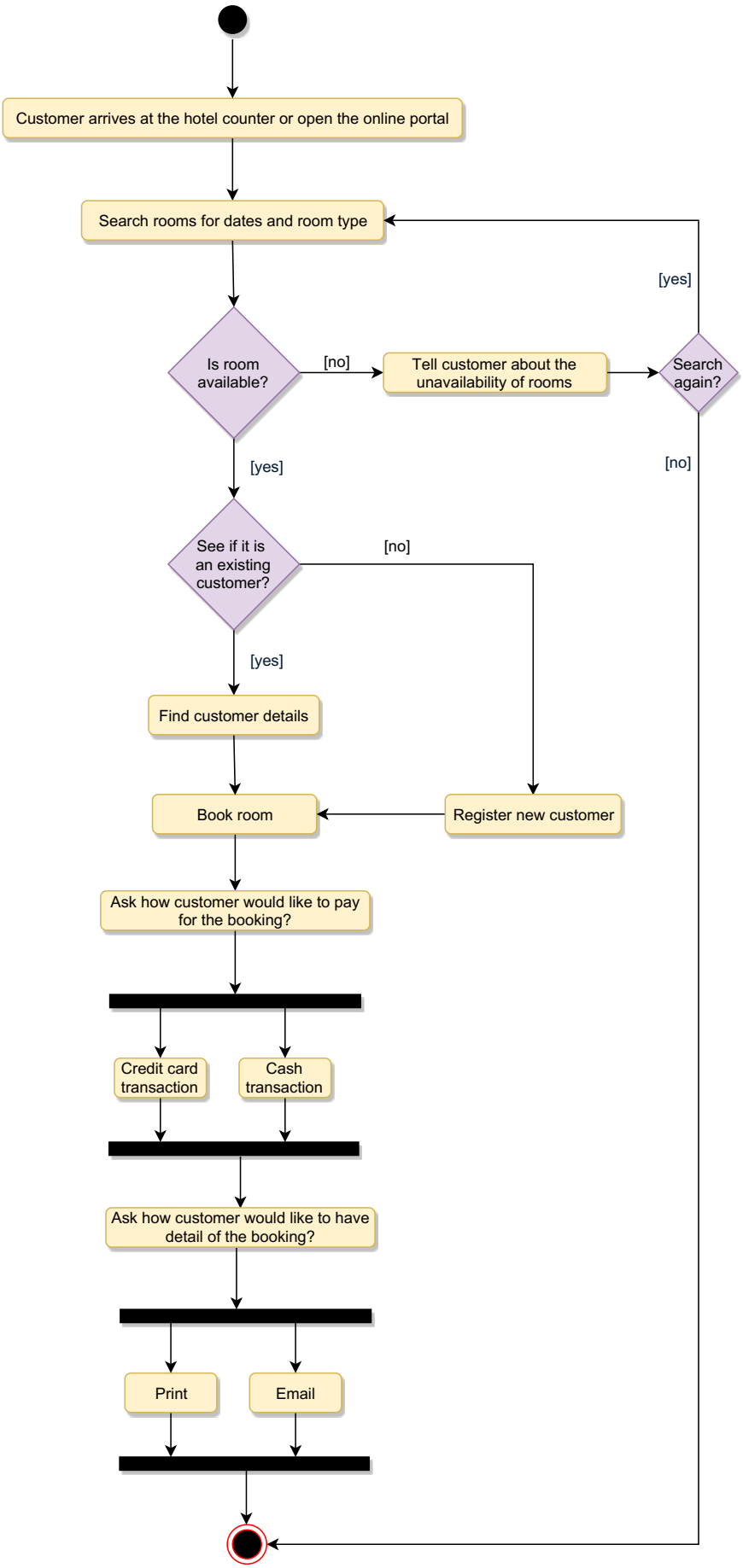
## UML conventions



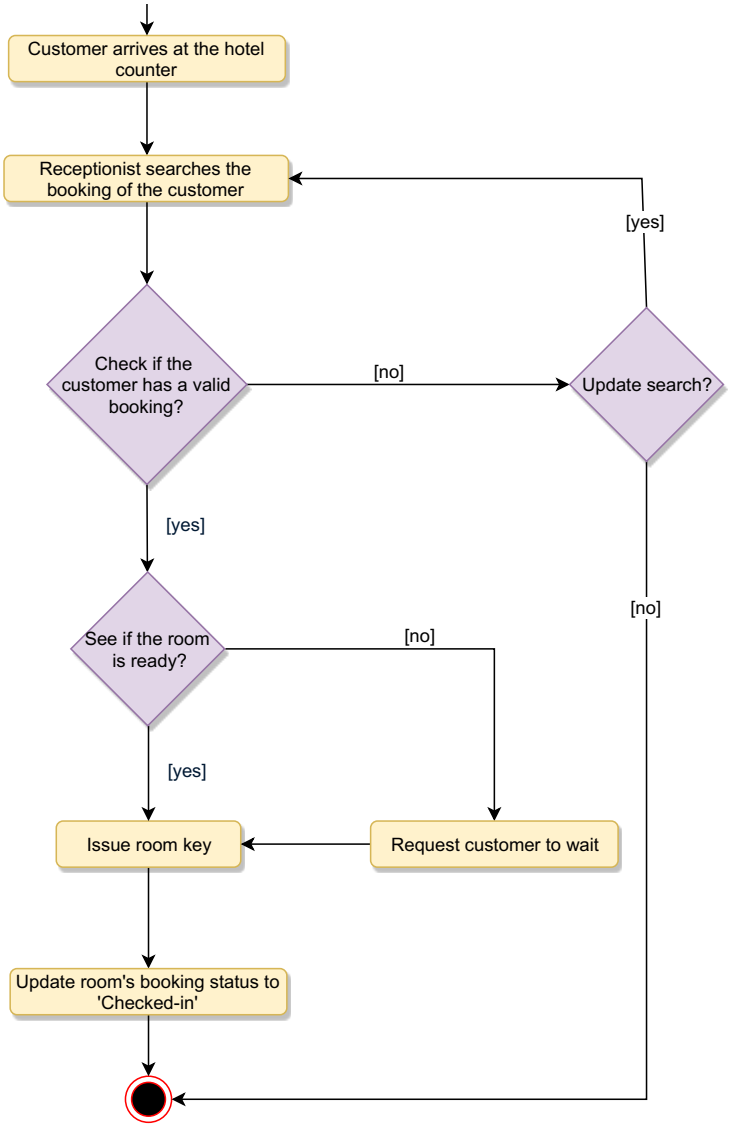

## Activity diagrams

**Make a room booking:** Any guest or receptionist can perform this activity. Here are the set of steps to

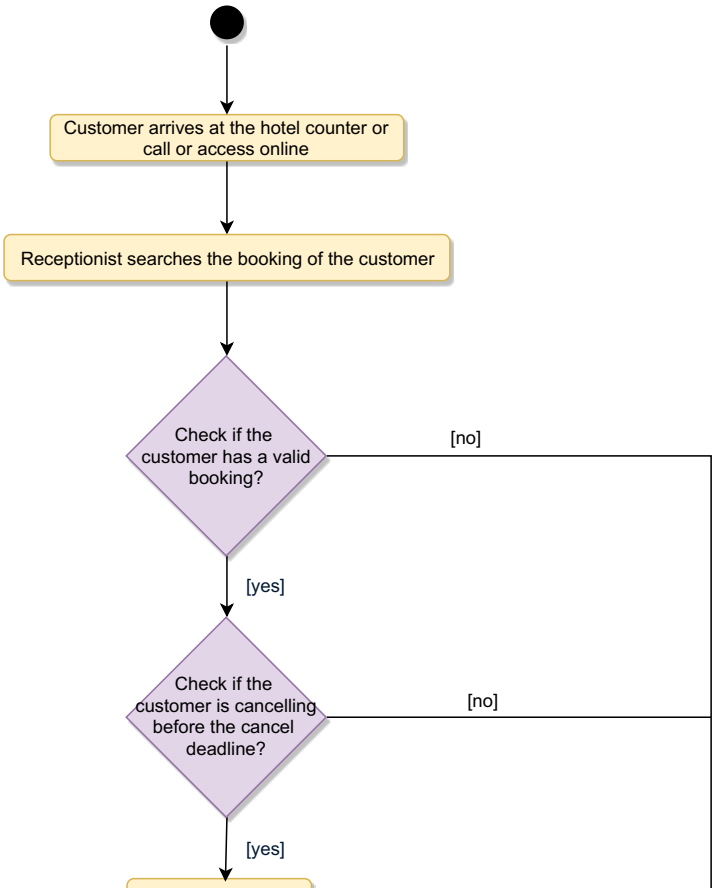
book a room:

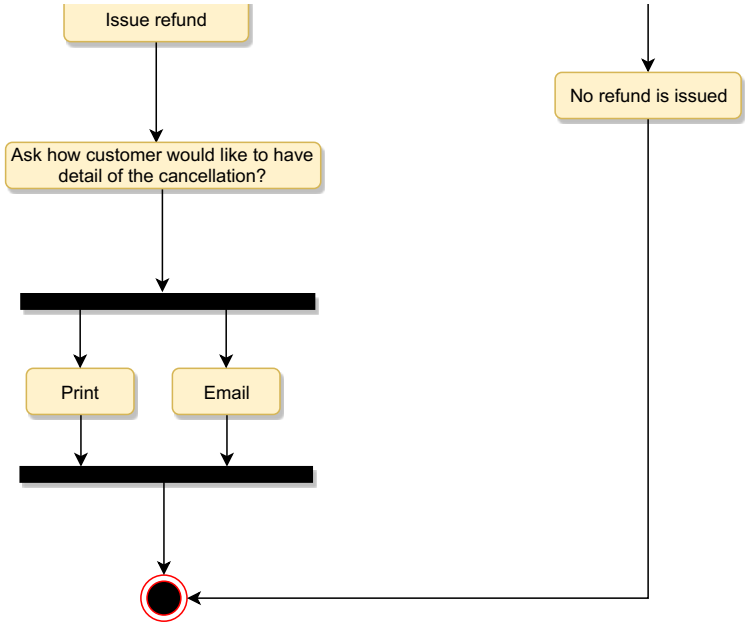



**Check in:** Guest will check in for their booking. The Receptionist can also perform this activity. Here are the steps:




**Cancel a booking:** Guest can cancel their booking. Receptionist can perform this activity. Here are the different steps of this activity:






## Code

Here is the high-level definition for the classes described above.

**Enums, data types, and constants:** Here are the required enums, data types, and constants:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
```

**Account, Person, Guest, Receptionist, and Server:** These classes represent the different people that interact with our system:

```
1
2
```

```
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
```

**Hotel and HotelLocation:** These classes represent the top-level classes of the system:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

**Room, RoomKey, and RoomHouseKeeping:** To encapsulate a room, room key, and housekeeping:

```
1
2
3
4
5
6
7
8
9
10
11
```

**RoomBooking and RoomCharge:** To encapsulate a booking and different charges against a booking:

```
34
35
```

Mark as Completed

← Back

Design Blackjack and a Deck of Cards

Next →

Design a Restaurant Management sys...

Send feedback 13 Recommendations

Stuck? Get help on DISCUSS

# Design a Restaurant Management system

Let's design a restaurant management system.

A Restaurant Management System is a software built to handle all restaurant activities in an easy and safe manner. This System will give the Restaurant management power and flexibility to manage the entire system from a single portal. The system allows the manager to keep track of available tables in the system as well as the reservation of tables and bill generation.


## System Requirements

We will focus on the following set of requirements while designing the Restaurant Management System:

1. The restaurant will have different branches.
2. Each restaurant branch will have a menu.
3. The menu will have different menu sections, containing different menu items.
4. The waiter should be able to create an order for a table and add meals for each seat.
5. Each meal can have multiple meal items. Each meal item corresponds to a menu item.
6. The system should be able to retrieve information about tables currently available to seat walk-in customers.
7. The system should support the reservation of tables.
8. The receptionist should be able to search for available tables by date/time and reserve a table.
9. The system should allow customers to cancel their reservation.

10. The system should be able to send notifications whenever the reservation time is approaching.
11. The customers should be able to pay their bills through credit card, check or cash.
12. Each restaurant branch can have multiple seating arrangements of tables.

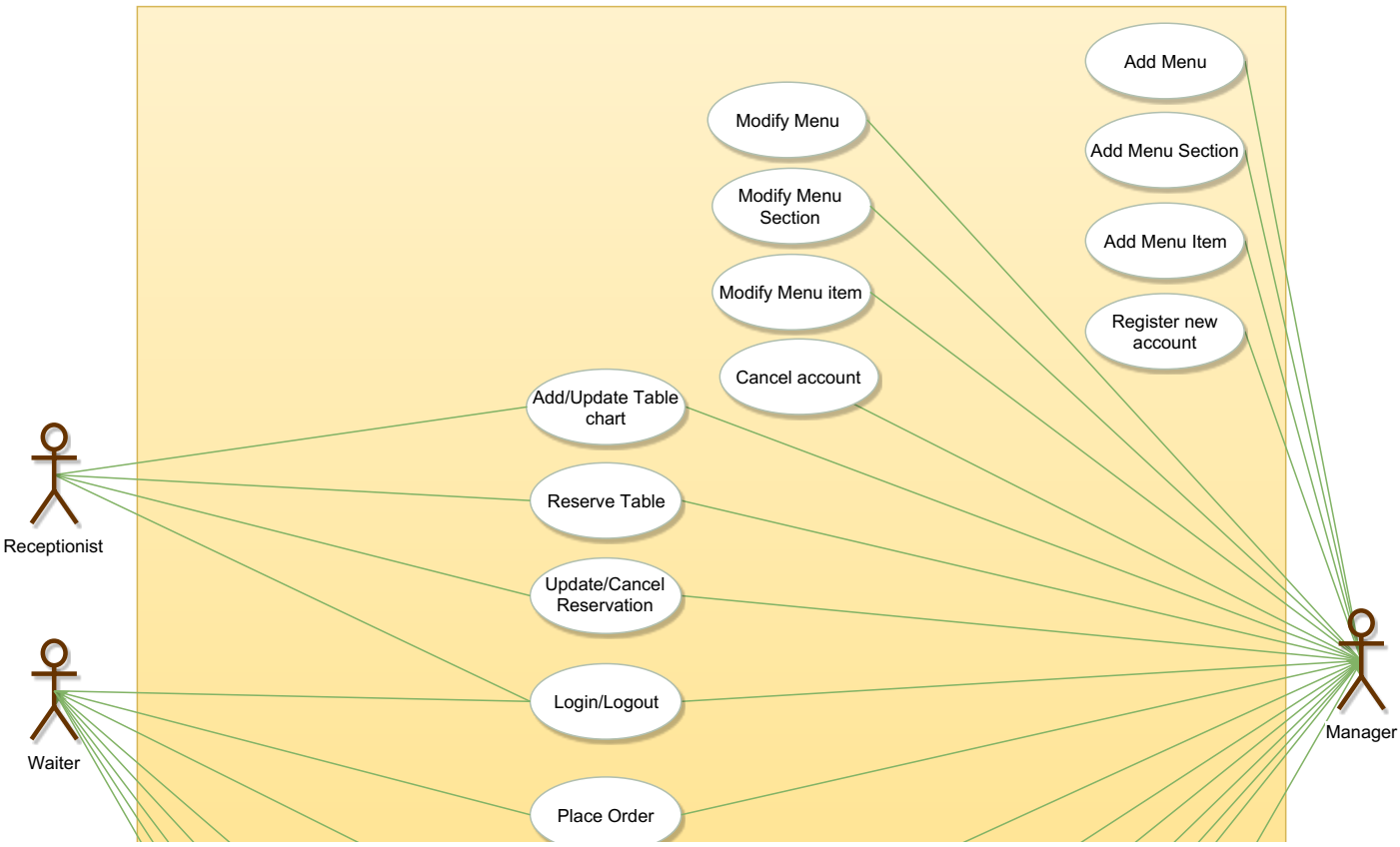
## Use case diagram

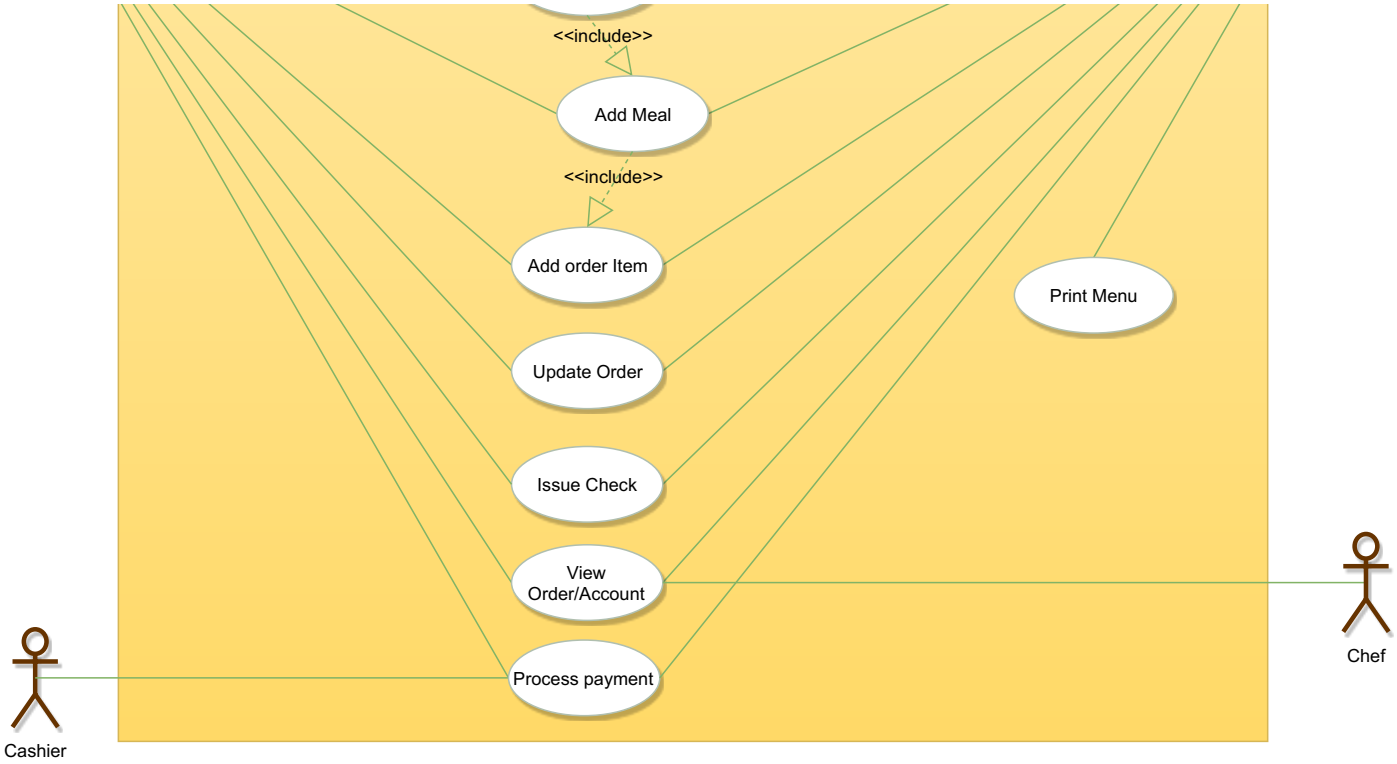
Here are the main Actors in our system:

- **Receptionist:** Mainly responsible for adding and modifying tables and their layout, and creating and canceling table reservations.
- **Waiter:** To take/modify orders.
- **Manager:** Mainly responsible for adding new workers and modifying the menu.
- **Chef:** To view and work on an order.
- **Cashier:** To generate checks and process payments.
- **System:** Mainly responsible for sending notifications about table reservations, cancellations, etc.

Here are the top use cases of the Restaurant Management System:

- **Add/Modify tables:** To add, remove, or modify a table in the system.
- **Search tables:** To search for available tables for reservation.
- **Place order:** Add a new order in the system for a table.
- **Update order:** Modify an already placed order, which can include adding/modifying meals or meal items.
- **Create a reservation:** To create a table reservation for a certain date/time for an available table.
- **Cancel reservation:** To cancel an existing reservation.
- **Check-in:** To let the guest check in for their reservation.
- **Make payment:** Pay the check for the food.



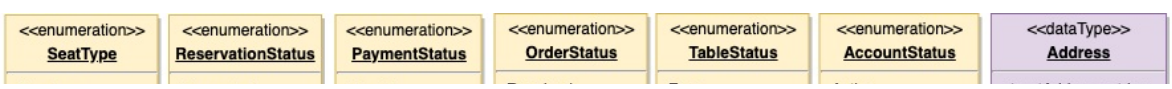


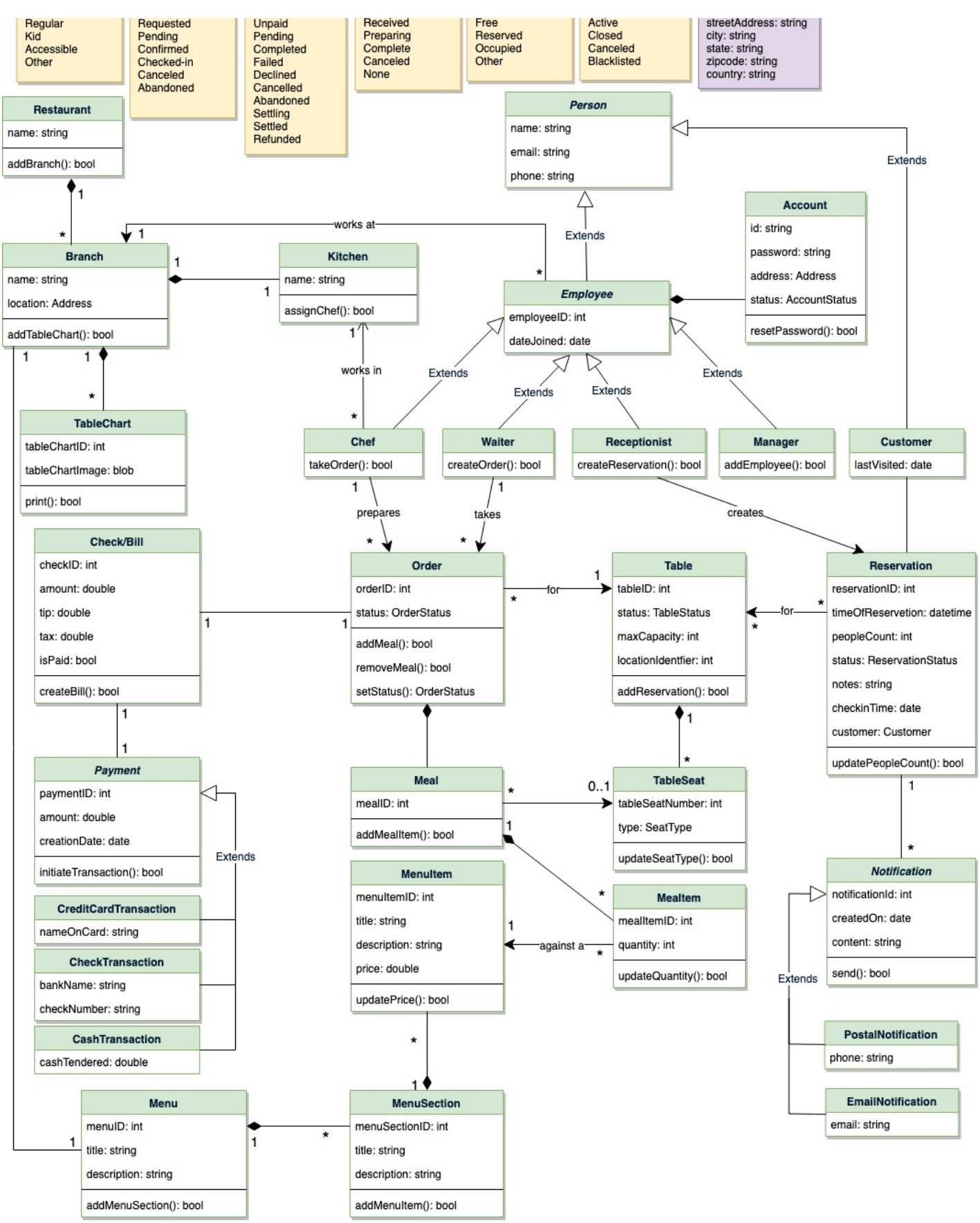

Use case diagram

## Class diagram

Here is the description of the different classes of our Restaurant Management System:

- **Restaurant:** This class represents a restaurant. Each restaurant has registered employees. The employees are part of the restaurant because if the restaurant becomes inactive, all its employees will automatically be deactivated.
- **Branch:** Any restaurants can have multiple branches. Each branch will have its own set of employees and menus.
- **Menu:** All branches will have their own menu.
- **MenuSection and MenuItem:** A menu has zero or more menu sections. Each menu section consists of zero or more menu items.
- **Table and TableSeat:** The basic building block of the system. Every table will have a unique identifier, maximum sitting capacity, etc. Each table will have multiple seats.
- **Order:** This class encapsulates the order placed by a customer.
- **Meal:** Each order will consist of separate meals for each table seat.
- **Meal Item:** Each Meal will consist of one or more meal items corresponding to a menu item.
- **Account:** We'll have different types of accounts in the system, one will be a receptionist to search and reserve tables and the other, the waiter will place orders in the system.
- **Notification:** Will take care of sending notifications to customers.
- **Bill:** Contains different bill-items for every meal item.






Class diagram

## UML conventions

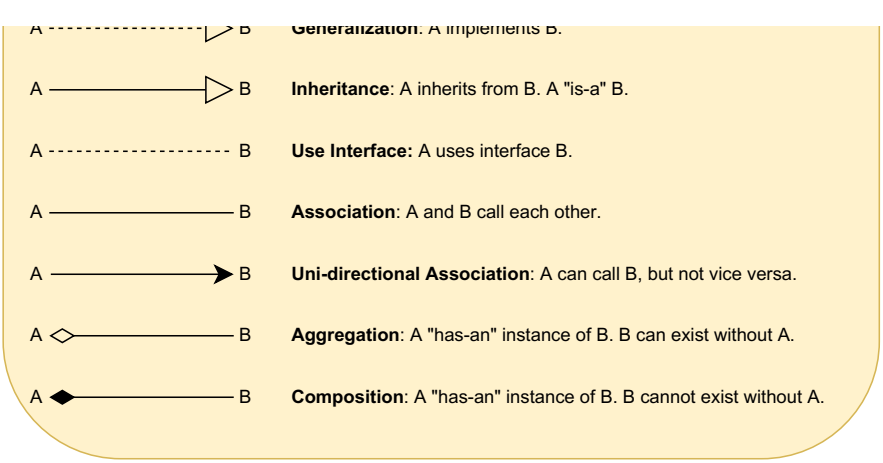
<<interface>>
**Name**
method1()

**Interface**: Classes implement interfaces, denoted by Generalization.

**ClassName**
property_name: type
method(): type

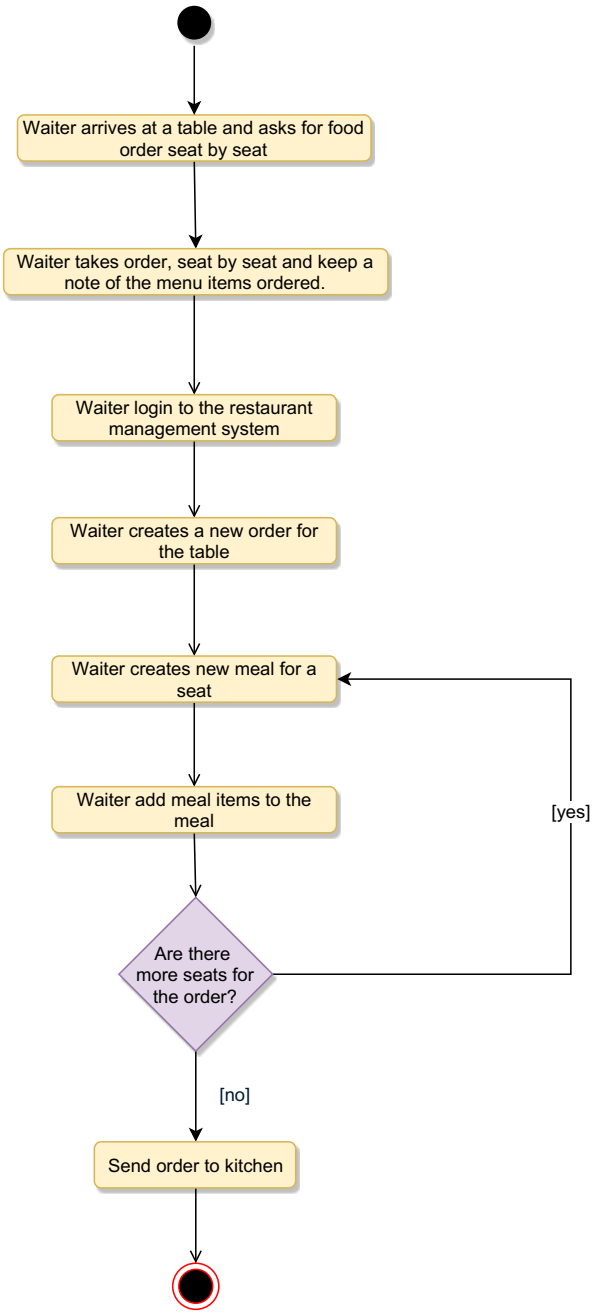
**Class**: Every class can have properties and methods.
Abstract classes are identified by their *Italic* names.

A ▷ B **Generalization**: A implements B

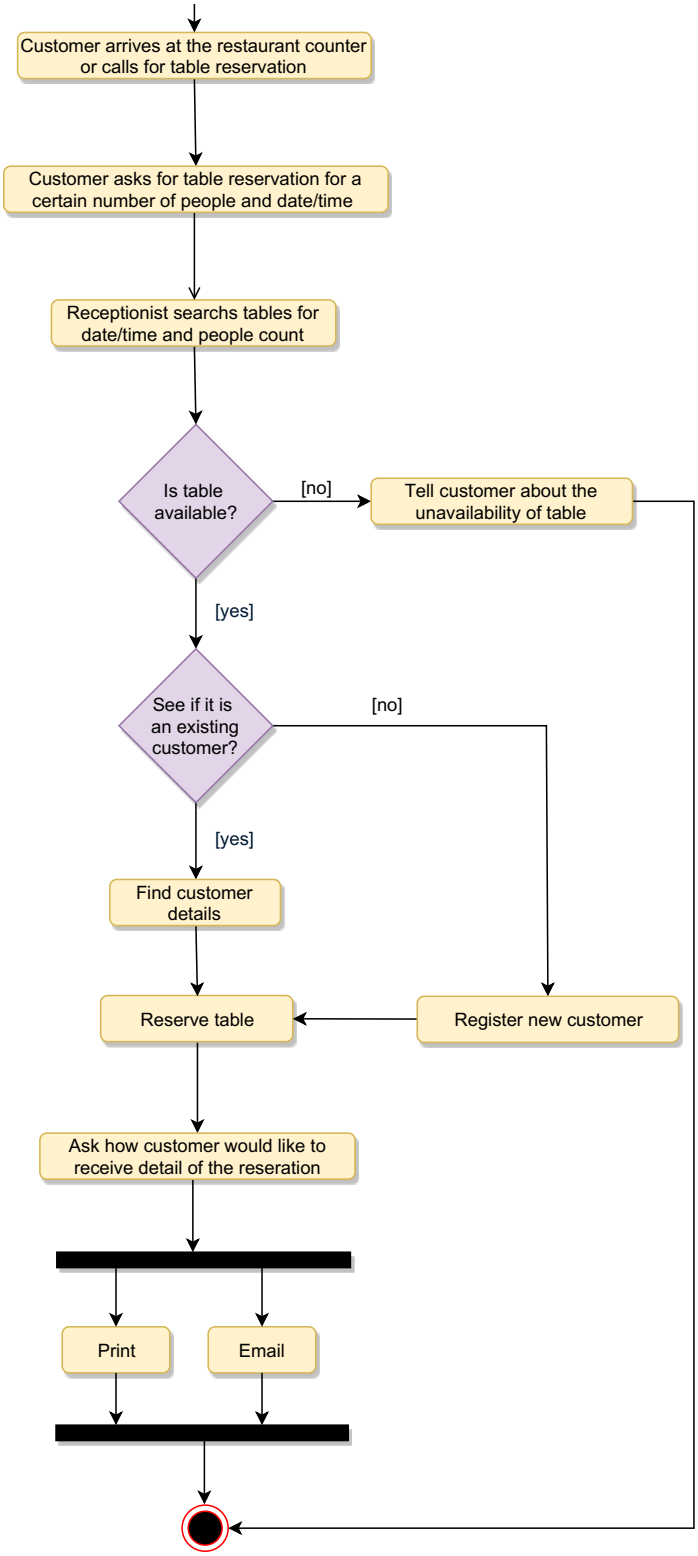



## Activity diagrams

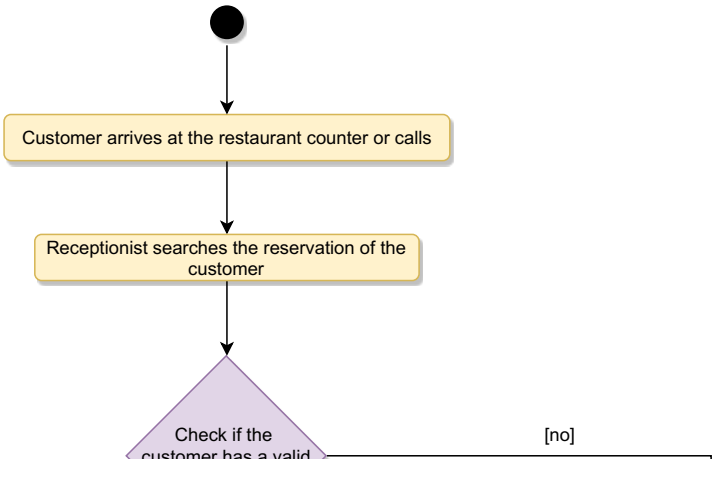
**Place order:** Any waiter can perform this activity. Here are the steps to place an order:

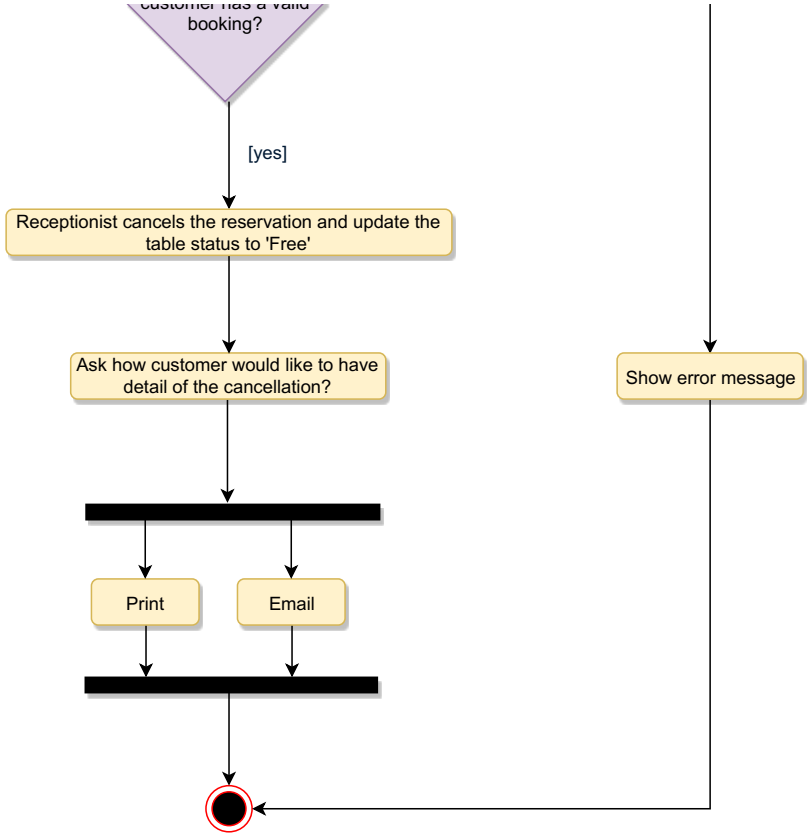



**Make a reservation:** Any receptionist can perform this activity. Here are the steps to make a reservation:




**Cancel a reservation:** Any receptionist can perform this activity. Here are the steps to cancel a reservation:






## Code

Here is the high-level definition for the classes described above.

**Enums, data types, and constants:** Here are the required enums, data types, and constants:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
```

**Account, Person, Employee, Receptionist, Manager, and Chef:** These classes represent the different

**Account, Person, Employee, Receptionist, Manager, and Chef:** These classes represent the different people that interact with our system:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
```

**Restaurant, Branch, Kitchen, TableChart:** These classes represent the top-level classes of the system:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```

```
23
24
25
26
27
28
```

**Table, TableSeat, and Reservation:** Each table can have multiple seats and customers can make reservations for tables:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
```

**Menu, MenuSection, and MenuItem:** Each restaurant branch will have its own menu, each menu will have multiple menu sections, which will contain menu items:

```
1
2
3
4
5
6
7
8
9
10
11
```

```
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
```

**Order, Meal, and MealItem:** Each order will have meals for table seats:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
```

Mark as Completed

← Back

Design a Hotel Management System

Next →

Design Chess

Send feedback 8 Recommendations

Stuck? Get help on DISCUSS

# Design Chess

Let's design a system to play online chess.

Chess is a two-player strategy board game played on a chessboard, which is a checkered gameboard with 64 squares arranged in an 8×8 grid. There are a few versions of game types that people play all over the world. In this design problem, we are going to focus on designing a two-player online chess game.

## System Requirements

We'll focus on the following set of requirements while designing the game of chess:

1. The system should support two online players to play a game of chess.
2. All rules of international chess will be followed.
3. Each player will be randomly assigned a side, black or white.
4. Both players will play their moves one after the other. The white side plays the first move.
5. Players can't cancel or roll back their moves.
6. The system should maintain a log of all moves by both players.
7. Each side will start with 8 pawns, 2 rooks, 2 bishops, 2 knights, 1 queen, and 1 king.
8. The game can finish either in a checkmate from one side, forfeit or stalemate (a draw), or resignation.

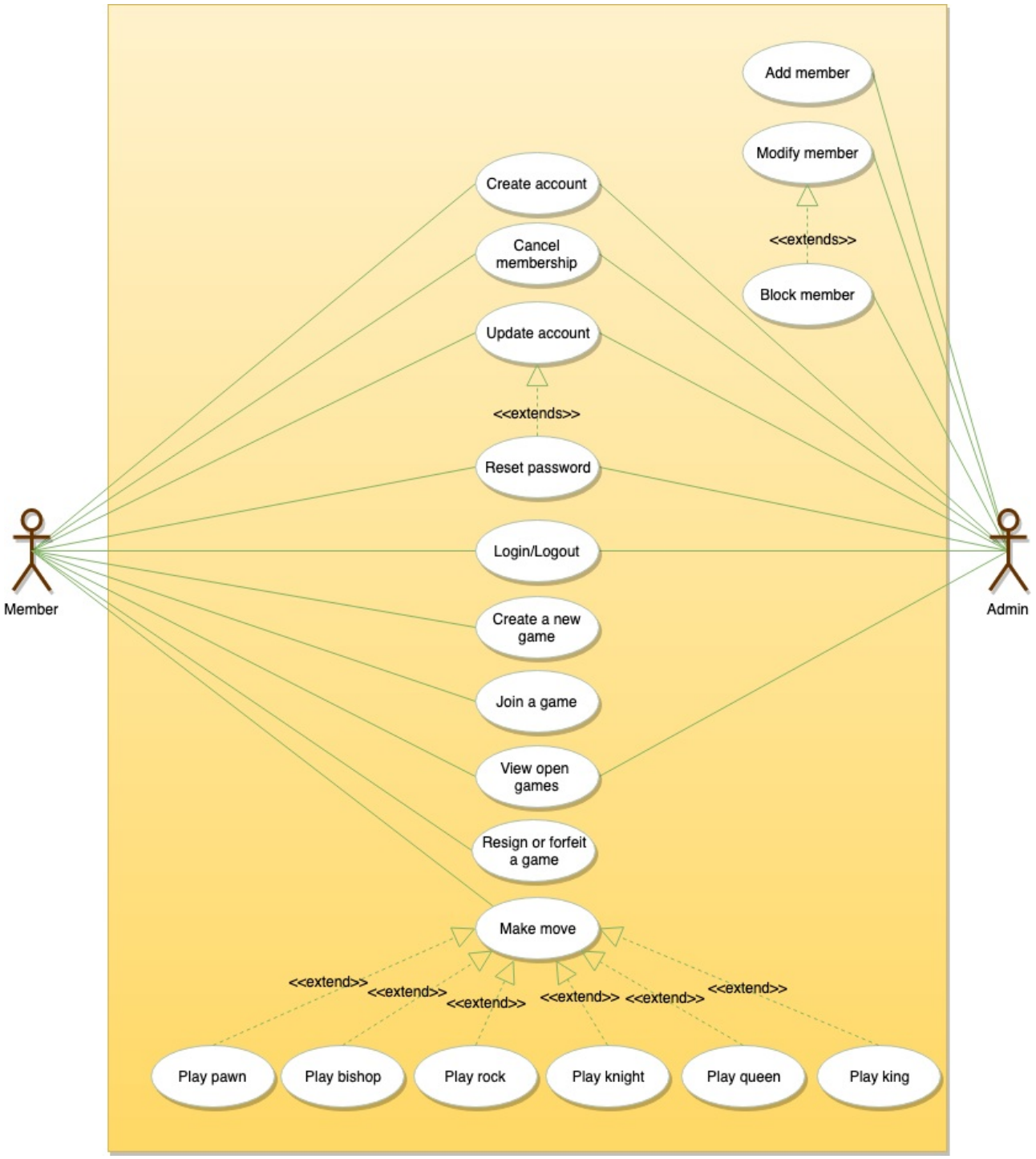
## Use case diagram

We have two actors in our system:

- **Player:** A registered account in the system, who will play the game. The player will play chess moves.
- **Admin:** To ban/modify players.

Here are the top use cases for chess:

- **Player moves a piece:** To make a valid move of any chess piece.
- **Resign or forfeit a game:** A player resigns from/forfeits the game.
- **Register new account/Cancel membership:** To add a new member or cancel an existing member.
- **Update game log:** To add a move to the game log.




Use case diagram

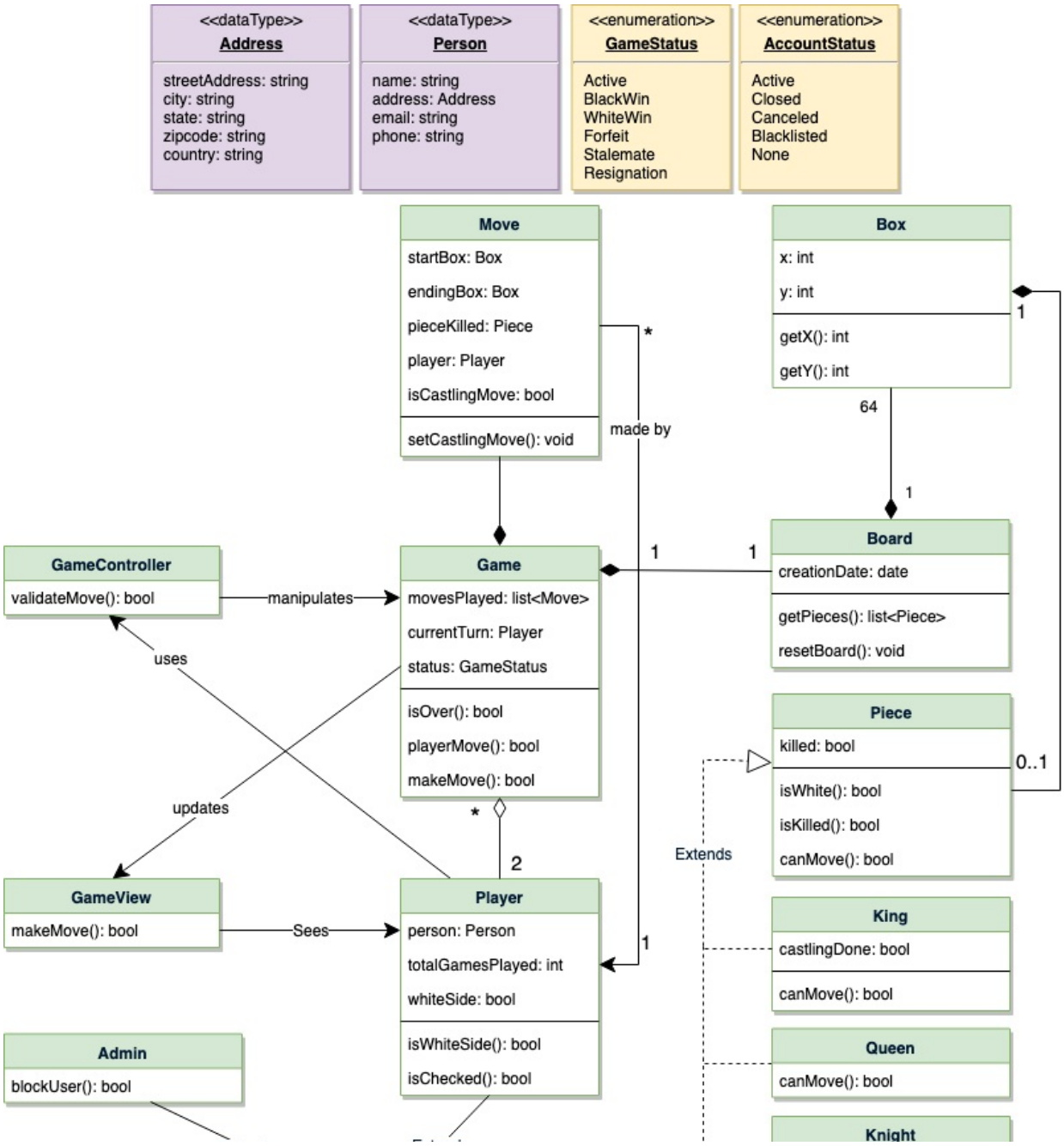
## Class diagram

Here are the main classes for chess:

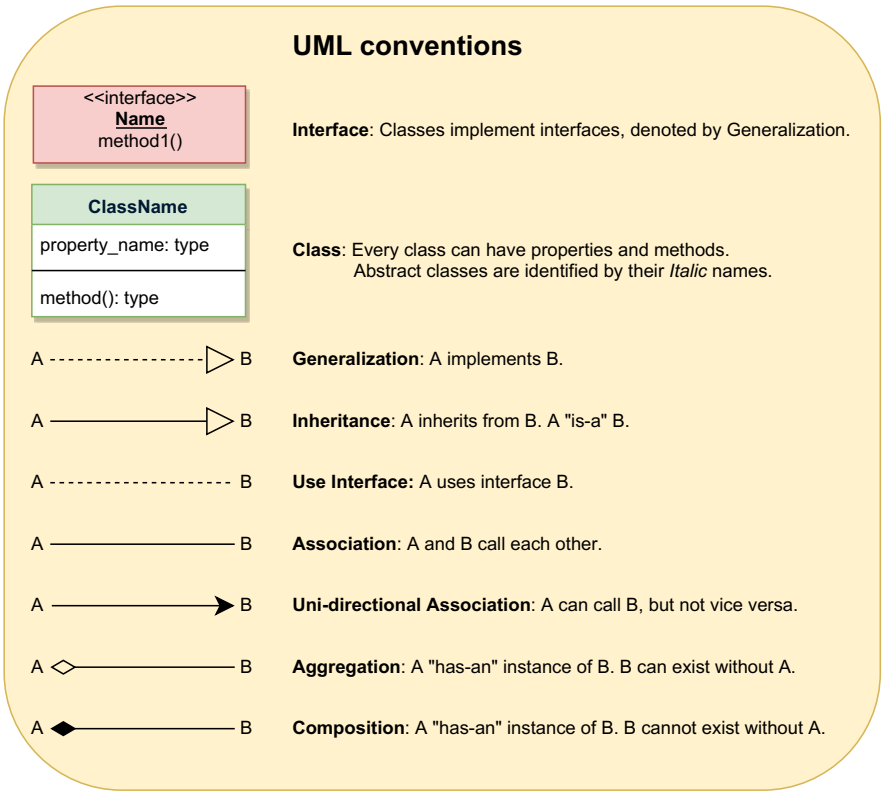
- **Player:** Player class represents one of the participants playing the game. It keeps track of which side (black or white) the player is playing.

- **Account:** We'll have two types of accounts in the system; one will be a player, and the other will be an

- **Account:** We'll have two types of accounts in the system: one will be a player, and the other will be an admin.
- **Game:** This class controls the flow of a game. It keeps track of all the game moves, which player has the current turn, and the final result of the game.
- **Box:** A box represents one block of the 8x8 grid and an optional piece.
- **Board:** Board is an 8x8 set of boxes containing all active chess pieces.
- **Piece:** The basic building block of the system, every piece will be placed on a box. This class contains the color the piece represents and the status of the piece (that is, if the piece is currently in play or not). This would be an abstract class and all game pieces will extend it.
- **Move:** Represents a game move, containing the starting and ending box. The Move class will also keep track of the player who made the move, if it is a castling move, or if the move resulted in the capture of a piece.
- **GameController:** Player class uses GameController to make moves.
- **GameView:** Game class updates the GameView to show changes to the players.

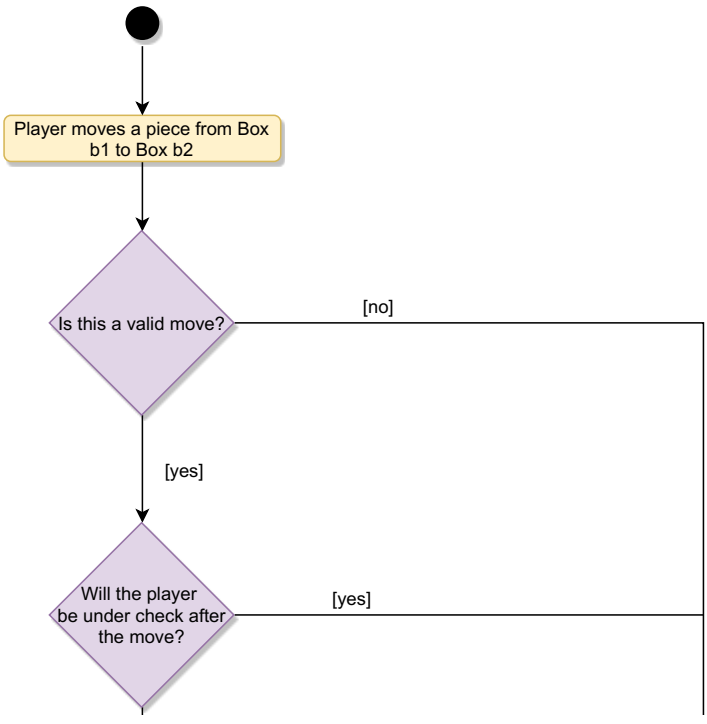


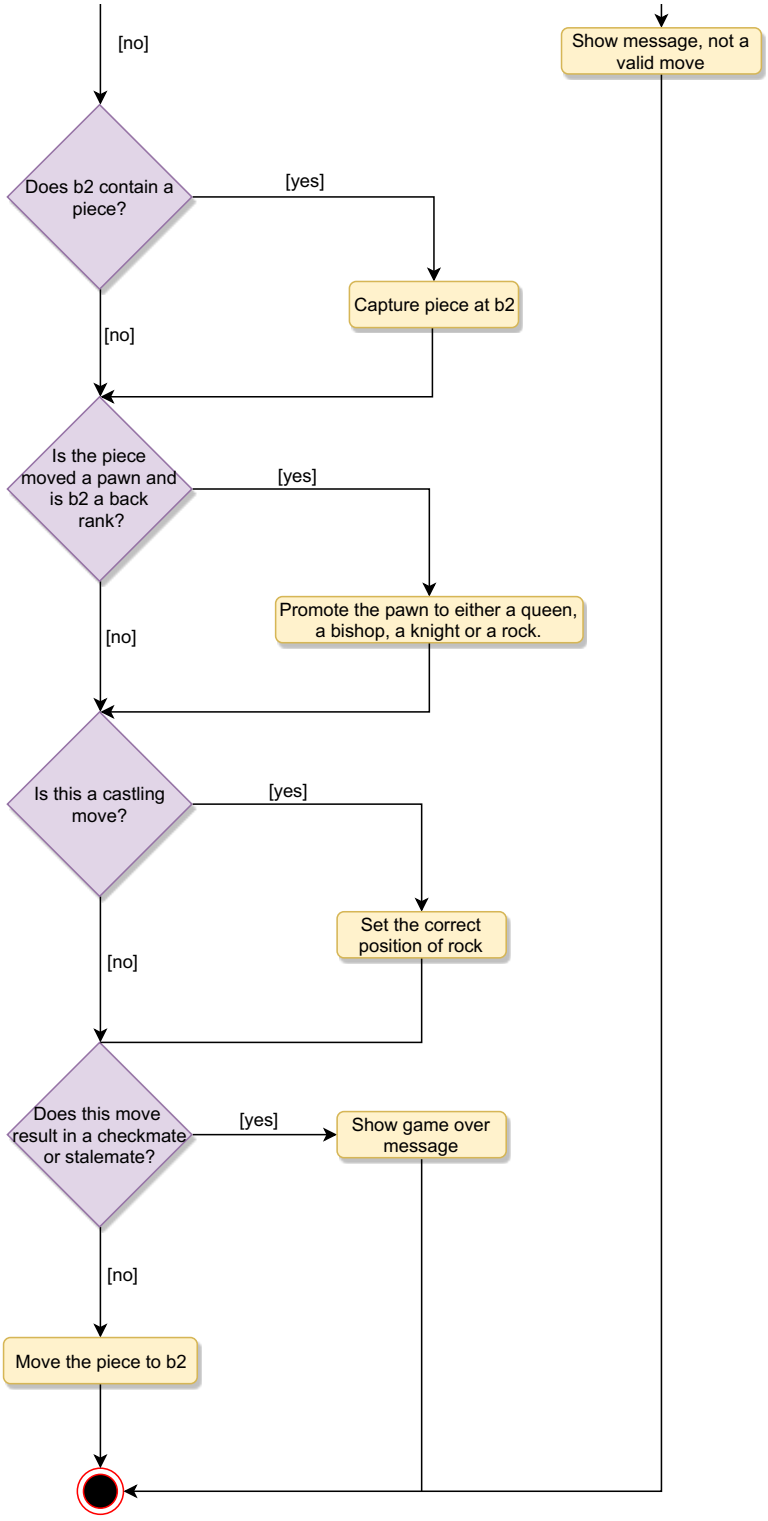

Class diagram




## Activity diagrams

**Make move:** Any Player can perform this activity. Here are the set of steps to make a move:






## Code

Here is the code for the top use cases.

**Enums, DataTypes, Constants:** Here are the required enums, data types, and constants:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
```

```
14
15
16
17
18
19
20
21
22
23
```

**Box:** To encapsulate a cell on the chess board:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
```

**Piece:** An abstract class to encapsulate common functionality of all chess pieces:

```
1
2
3
4
5
6
7
8
9
10
11
12
```

```
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
```

**King:** To encapsulate King as a chess piece:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
```

```
53
54
55
56
57
58
59
60
61
62
63
64
65
```

**Knight:** To encapsulate Knight as a chess piece:

```
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
16
17
18
19
20
```

**Board:** To encapsulate a chess board:

```
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

```
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
```

**Player:** To encapsulate a chess player:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

**Move:** To encapsulate a chess move:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

**Game:** To encapsulate a chess game:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
```

```
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Mark as Completed

← Back

Design a Restaurant Management sys...

Next →

Design an Online Stock Brokerage Sys...

Send feedback

12 Recommendations

Stuck? Get help on DISCUSS

# Design an Online Stock Brokerage System

Let's design an Online Stock Brokerage System.

An Online Stock Brokerage System facilitates its users the trade (i.e. buying and selling) of stocks online. It allows clients to keep track of and execute their transactions, and shows performance charts of the different stocks in their portfolios. It also provides security for their transactions and alerts them to pre-defined levels of changes in stocks, without the use of any middlemen.

The online stock brokerage system automates traditional stock trading using computers and the internet, making the transaction faster and cheaper. This system also gives speedier access to stock reports, current market trends, and real-time stock prices.

## System Requirements

We will focus on the following set of requirements while designing the online stock brokerage system:

1. Any user of our system should be able to buy and sell stocks.
2. Any user can have multiple watchlists containing multiple stock quotes.
3. Users should be able to place stock trade orders of the following types: 1) market, 2) limit, 3) stop loss and, 4) stop limit.
4. Users can have multiple ‘lots’ of a stock. This means that if a user has bought a stock multiple times, the system should be able to differentiate between different lots of the same stock.
5. The system should be able to generate reports for quarterly updates and yearly tax statements.
6. Users should be able to deposit and withdraw money either via check, wire or electronic bank transfer.
7. The system should be able to send notifications whenever trade orders are executed.

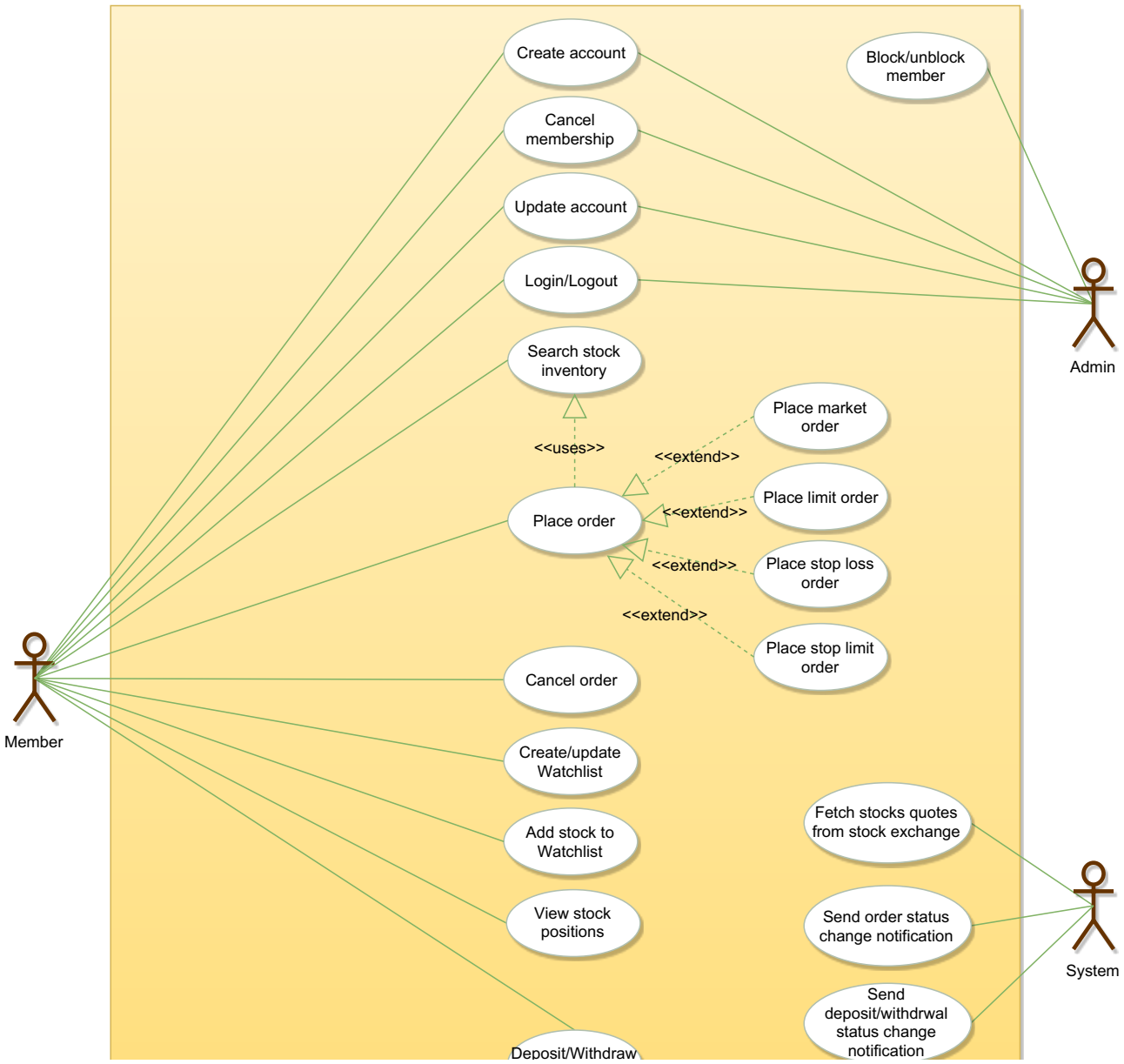
## Usecase diagram

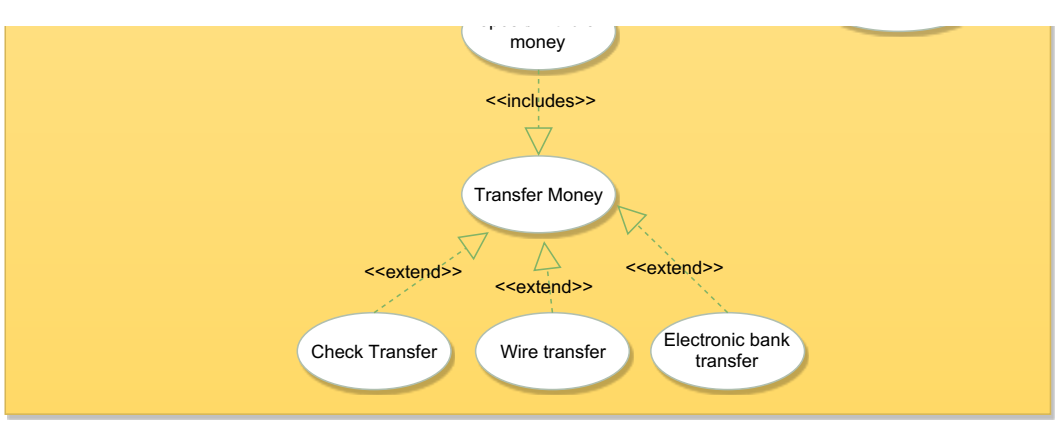
We have three main Actors in our system:

- **Admin:** Mainly responsible for administrative functions like blocking or unblocking members.
- **Member:** All members can search the stock inventory, as well as buy and sell stocks. Members can have multiple watchlists containing multiple stock quotes.
- **System:** Mainly responsible for sending notifications for stock orders and periodically fetching stock quotes from the stock exchange.

Here are the top use cases of the Stock Brokerage System:

- **Register new account/Cancel membership:** To add a new member or cancel the membership of an existing member.
- **Add/Remove/Edit watchlist:** To add, remove or modify a watchlist.
- **Search stock inventory:** To search for stocks by their symbols.
- **Place order:** To place a buy or sell order on the stock exchange.
- **Cancel order:** Cancel an already placed order.
- **Deposit/Withdraw money:** Members can deposit or withdraw money via check, wire or electronic bank transfer.





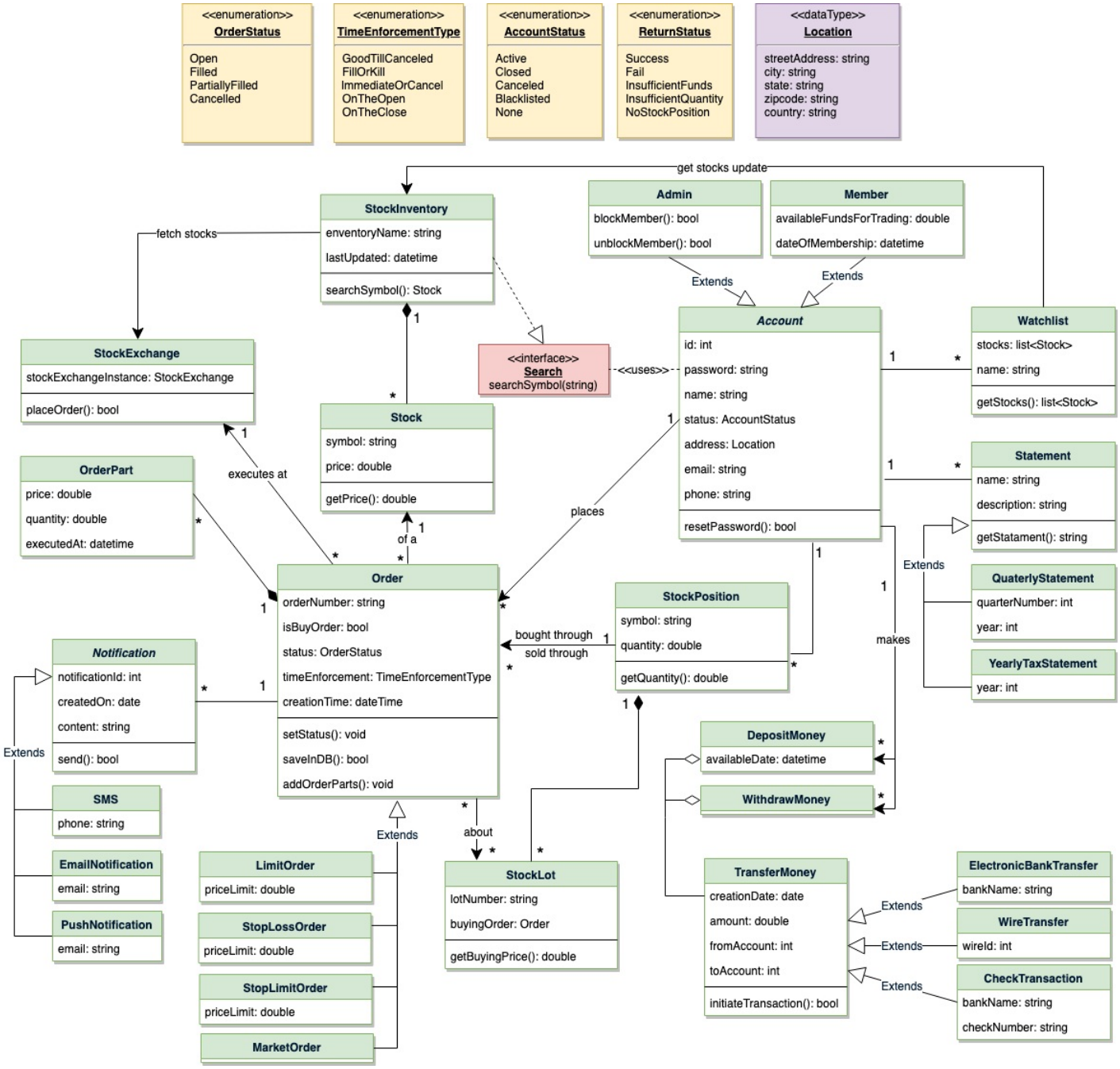

Use case diagram for Stock Brokerage System

## Class diagram

Here are the main classes of our Online Stock Brokerage System:

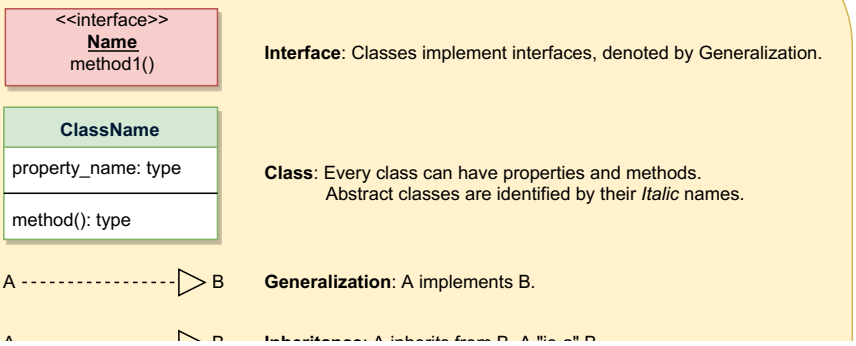
- **Account:** Consists of the member's name, address, e-mail, phone, total funds, funds that are available for trading, etc. We'll have two types of accounts in the system: one will be a general member, and the other will be an Admin. The Account class will also contain all the stocks the member is holding.
- **StockExchange:** The stockbroker system will fetch all stocks and their current prices from the stock exchange. StockExchange will be a singleton class encapsulating all interactions with the stock exchange. This class will also be used to place stock trading orders on the stock exchange.
- **Stock:** The basic building block of the system. Every stock will have a symbol, current trading price, etc.
- **StockInventory:** This class will fetch and maintain the latest stock prices from the StockExchange. All system components will read the most recent stock prices from this class.
- **Watchlist:** A watchlist will contain a list of stocks that the member wants to follow.
- **Order:** Members can place stock trading orders whenever they would like to sell or buy stock positions. The system would support multiple types of orders:
    - **Market Order:** Market order will enable users to buy or sell stocks immediately at the current market price.
    - **Limit Order:** Limit orders will allow a user to set a price at which they want to buy or sell a stock.
    - **Stop Loss Order:** An order to buy or sell once the stock reaches a certain price.
    - **Stop Limit Order:** The stop-limit order will be executed at a specified price, or better, after a given stop price has been reached. Once the stop price is reached, the stop-limit order becomes a limit order to buy or sell at the limit price or better.
- **OrderPart:** An order could be fulfilled in multiple parts. For example, a market order to buy 100 stocks could have one part containing 70 stocks at $10 and another part with 30 stocks at $10.05.
- **StockLot:** Any member can buy multiple lots of the same stock at different times. This class will represent these individual lots. For example, the user could have purchased 100 shares of AAPL yesterday and 50 more stocks of AAPL today. While selling, users will be able to select which lot they want to sell first.
- **StockPosition:** This class will contain all the stocks that the user holds

- **StockPosition:** This class will contain all the stocks that the user holds.
- **Statement:** All members will have reports for quarterly updates and yearly tax statements.
- **DepositMoney & WithdrawMoney:** Members will be able to move money through check, wire or electronic bank transfers.
- **Notification:** Will take care of sending notifications to members.




Class diagram

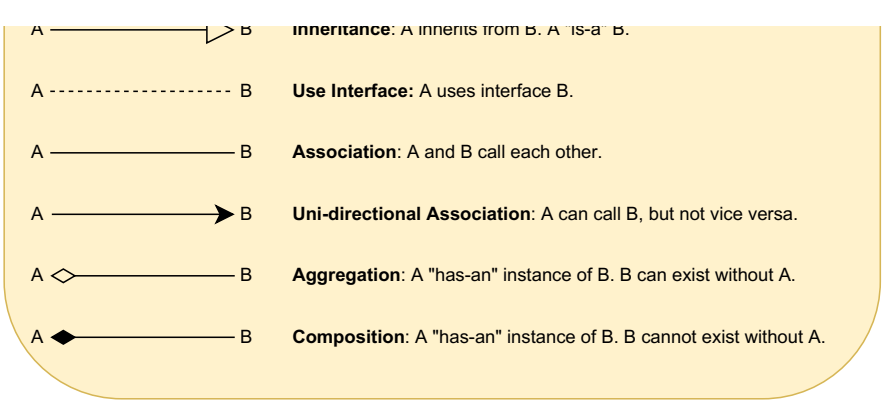
## UML conventions




**Interface**: Classes implement interfaces, denoted by Generalization.

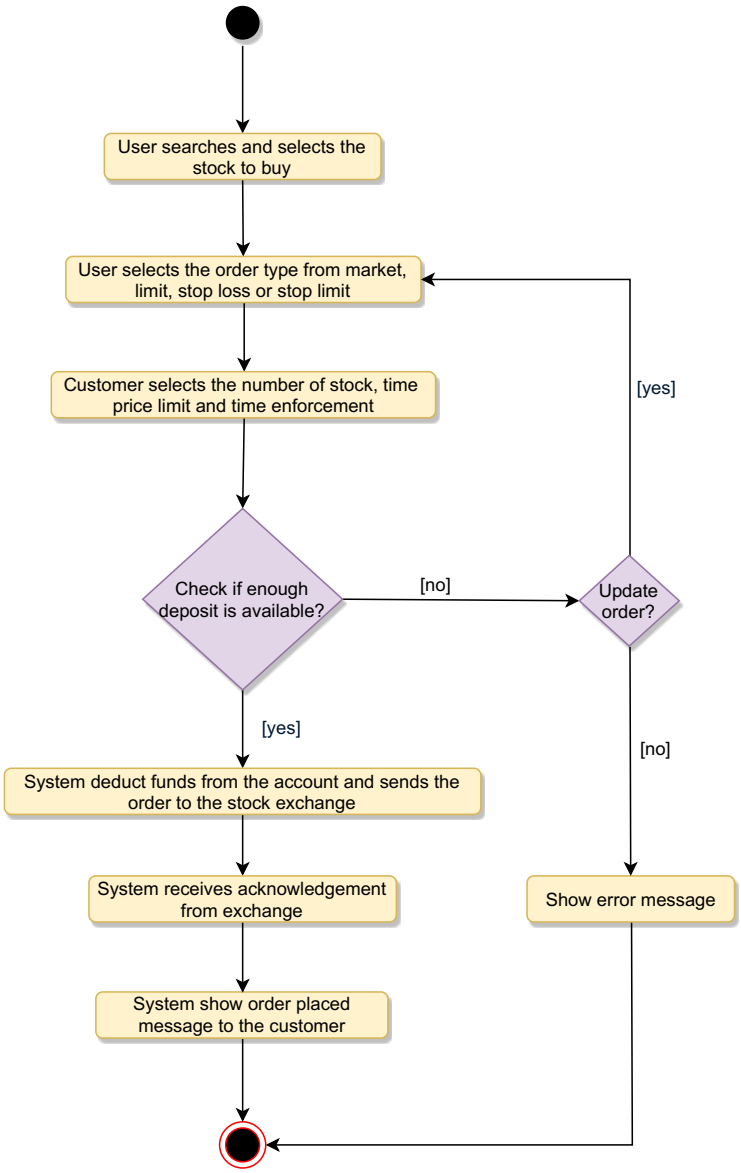
**Class**: Every class can have properties and methods.
Abstract classes are identified by their *Italic* names.

**Generalization**: A implements B.

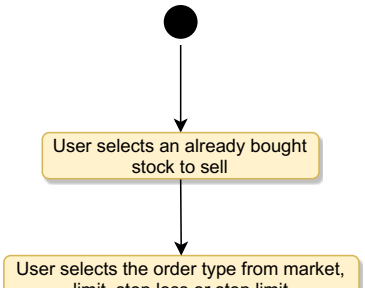


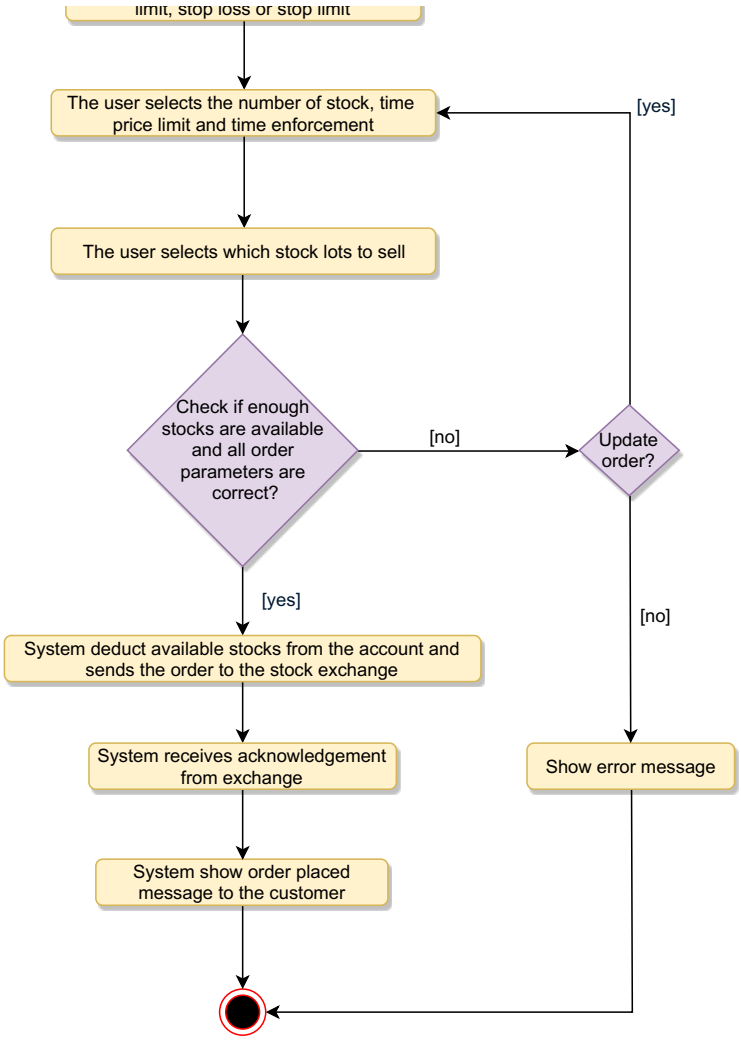

## Activity diagrams

**Place a buy order:** Any system user can perform this activity. Here are the steps to place a buy order:




**Place a sell order:** Any system user can perform this activity. Here are the steps to place a buy order:






## Code

Here is the code for the top use cases.

**Enums and Constants:** Here are the required enums and constants:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
```

**StockExchange:** To encapsulate all the interactions with the stock exchange:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```

**Order:** To encapsulate all buy or sell orders:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
```

**Member:** Members will be buying and selling stocks:

```
1
2
```

```
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
```

Mark as Completed

← Back

Design Chess

Next →

Design a Car Rental System

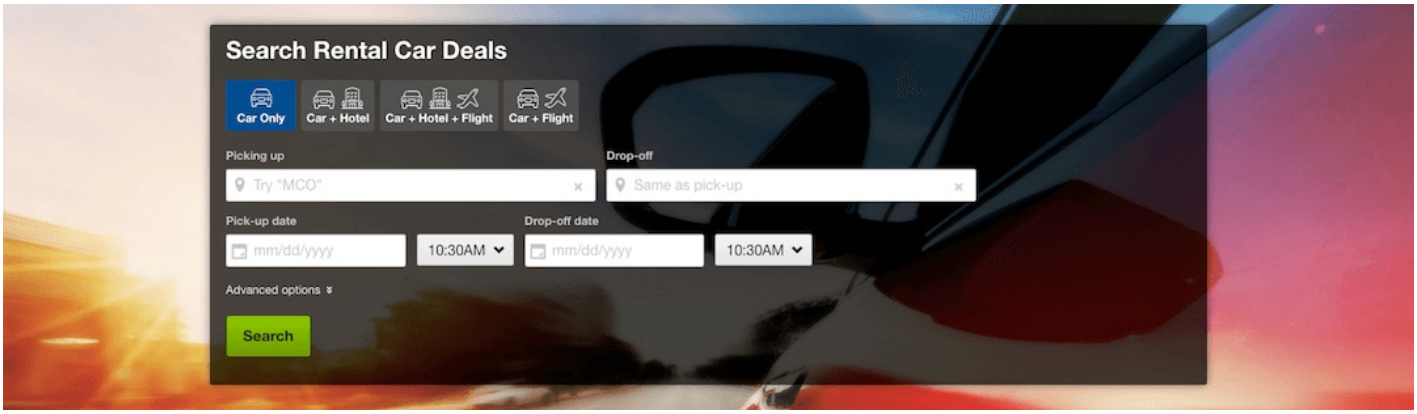
Send feedback 7 Recommendations

Stuck? Get help on DISCUSS

# Design a Car Rental System

Let's design a car rental system where customers can rent vehicles.

A Car Rental System is a software built to handle the renting of automobiles for a short period of time, generally ranging from a few hours to a few weeks. A car rental system often has numerous local branches (to allow its user to return a vehicle to a different location), and primarily located near airports or busy city areas.




## System Requirements

We will focus on the following set of requirements while designing our Car Rental System:

1. The system will support the renting of different automobiles like cars, trucks, SUVs, vans, and motorcycles.
2. Each vehicle should be added with a unique barcode and other details, including a parking stall number which helps to locate the vehicle.
3. The system should be able to retrieve information like which member took a particular vehicle or what vehicles have been rented out by a specific member.
4. The system should collect a late-fee for vehicles returned after the due date.
5. Members should be able to search the vehicle inventory and reserve any available vehicle.
6. The system should be able to send notifications whenever the reservation is approaching the pick-up date, as well as when the vehicle is nearing the due date or has not been returned within the due date.
7. The system will be able to read barcodes from vehicles.
8. Members should be able to cancel their reservations.
9. The system should maintain a vehicle log to track all events related to the vehicles.
10. Members can add rental insurance to their reservation.
11. Members can rent additional equipment, like navigation, child seat, ski rack, etc.

12. Members can add additional services to their reservation, such as roadside assistance, additional driver, wifi, etc.

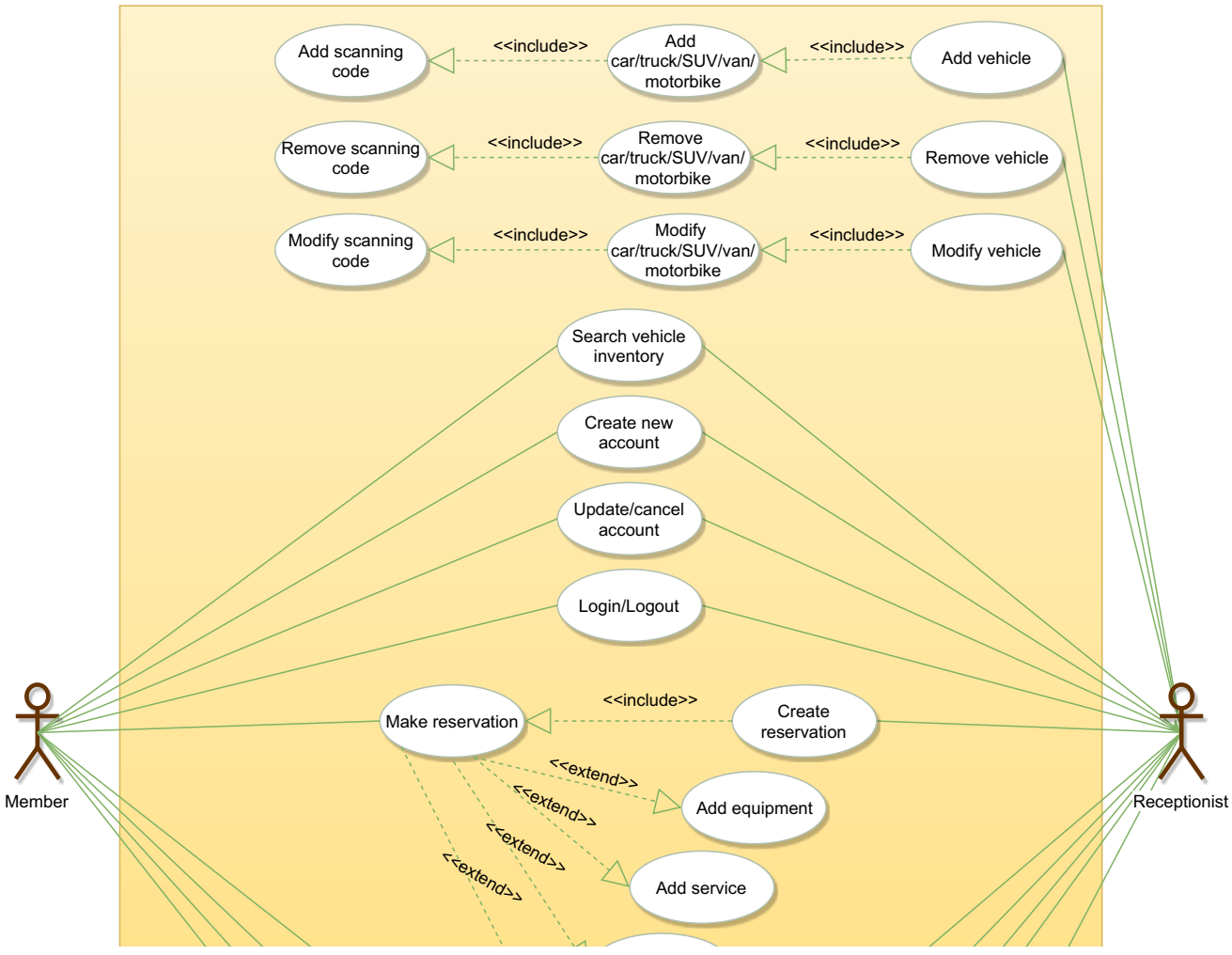
## Use case diagram

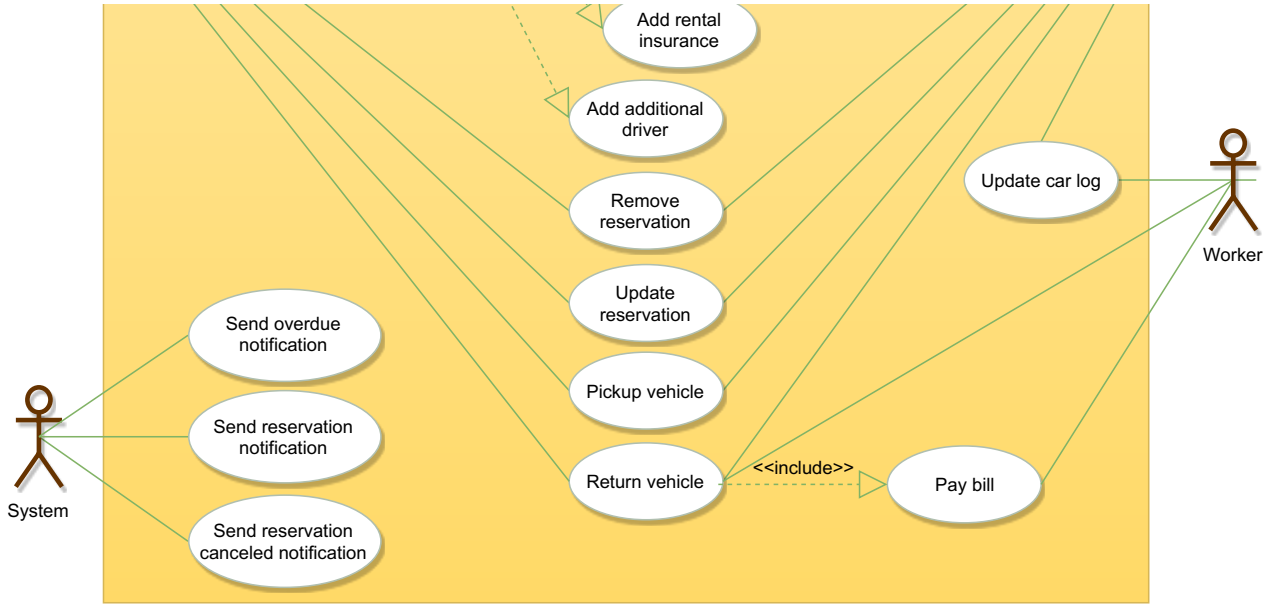
We have four main Actors in our system:

- **Receptionist:** Mainly responsible for adding and modifying vehicles and workers. Receptionists can also reserve vehicles.
- **Member:** All members can search the catalog, as well as reserve, pick-up, and return a vehicle.
- **System:** Mainly responsible for sending notifications about overdue vehicles, canceled reservation, etc.
- **Worker:** Mainly responsible for taking care of a returned vehicle and updating the vehicle log.

Here are the top use cases of the Car Rental System:

- **Add/Remove/Edit vehicle:** To add, remove or modify a vehicle.
- **Search catalog:** To search for vehicles by type and availability.
- **Register new account/Cancel membership:** To add a new member or cancel an existing membership.
- **Reserve vehicle:** To reserve a vehicle.
- **Check-out vehicle:** To rent a vehicle.
- **Return a vehicle:** To return a vehicle which was checked-out to a member.
- **Add equipment:** To add an equipment to a reservation like navigation, child seat, etc.
- **Update car log:** To add or update a car log entry, such as refueling, cleaning, damage, etc.



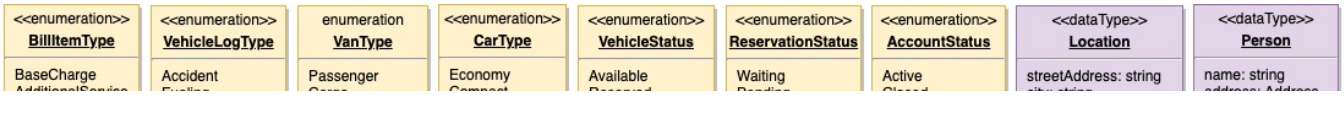


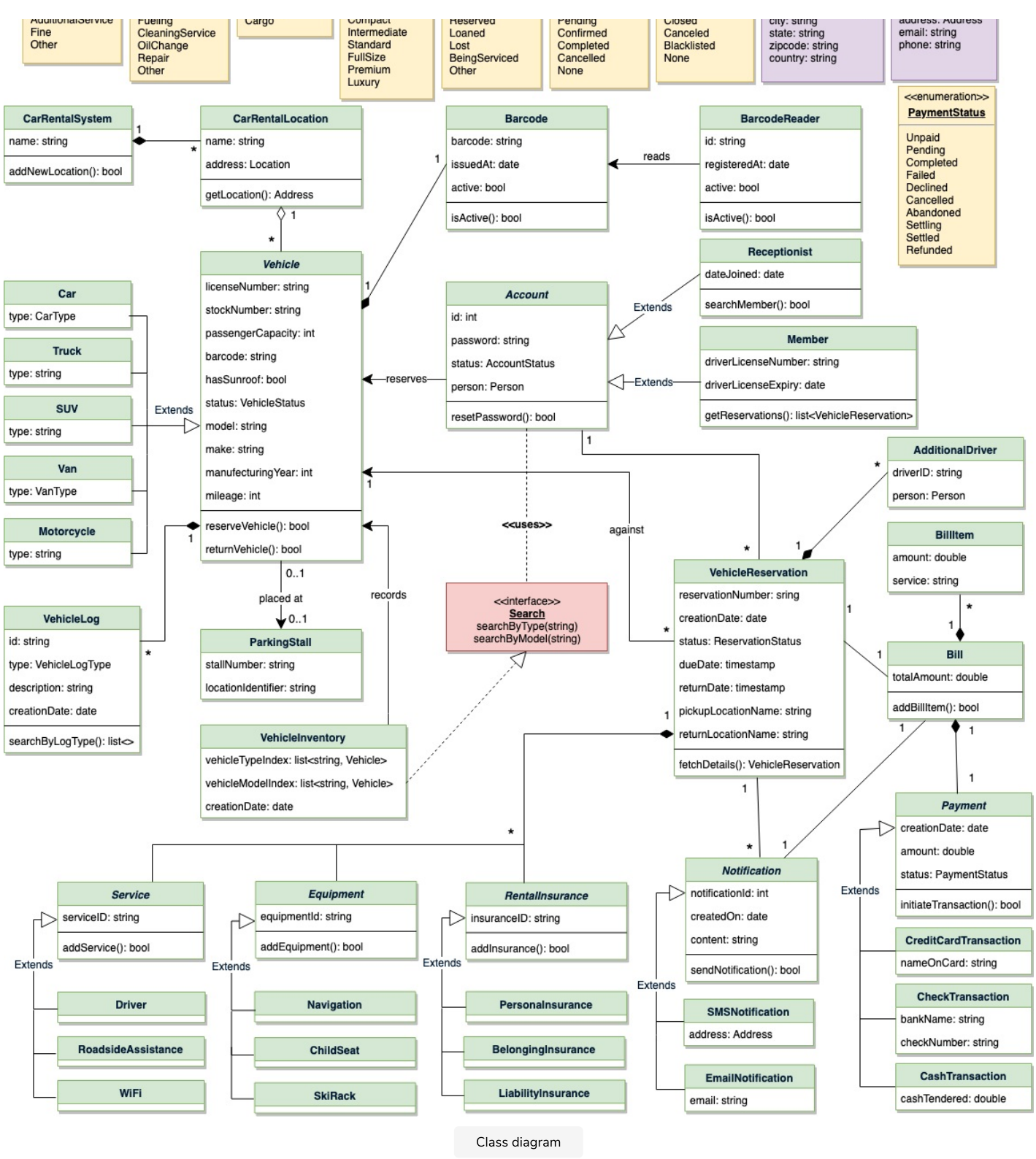

Use case diagram

## Class diagram

Here are the main classes of our Car Rental System:

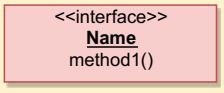
- **CarRentalSystem:** The main part of the organization for which this software has been designed.
- **CarRentalLocation:** The car rental system will have multiple locations, each location will have attributes like ‘Name’ to distinguish it from any other locations and ‘Address’ which defines the address of the rental location.
- **Vehicle:** The basic building block of the system. Every vehicle will have a barcode, license plate number, passenger capacity, model, make, mileage, etc. Vehicles can be of multiple types, like car, truck, SUV, etc.
- **Account:** Mainly, we will have two types of accounts in the system, one will be a general member and the other will be a receptionist. Another account can be of the worker taking care of the returned vehicle.
- **VehicleReservation:** This class will be responsible for managing reservations for a vehicle.
- **Notification:** Will take care of sending notifications to members.
- **VehicleLog:** To keep track of all the events related to a vehicle.
- **RentalInsurance:** Stores details about the various rental insurances that members can add to their reservation.
- **Equipment:** Stores details about the various types of equipment that members can add to their reservation.
- **Service:** Stores details about the various types of service that members can add to their reservation, such as additional drivers, roadside assistance, etc.
- **Bill:** Contains different bill-items for every charge for the reservation.



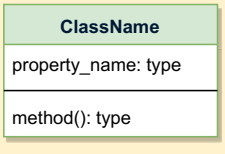



Class diagram

## UML conventions




**Interface**: Classes implement interfaces, denoted by Generalization.



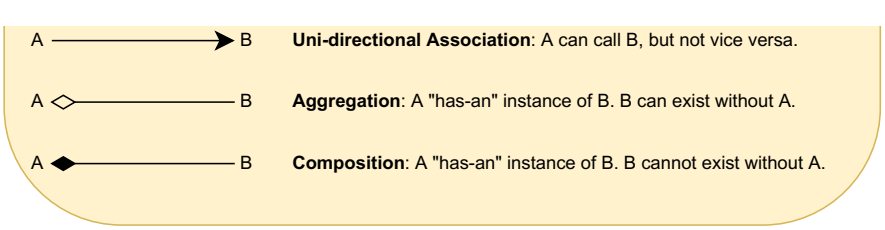

**Class**: Every class can have properties and methods.
Abstract classes are identified by their *Italic* names.

A ----------▷ B **Generalization**: A implements B.

A ——————▷ B **Inheritance**: A inherits from B. A "is-a" B.

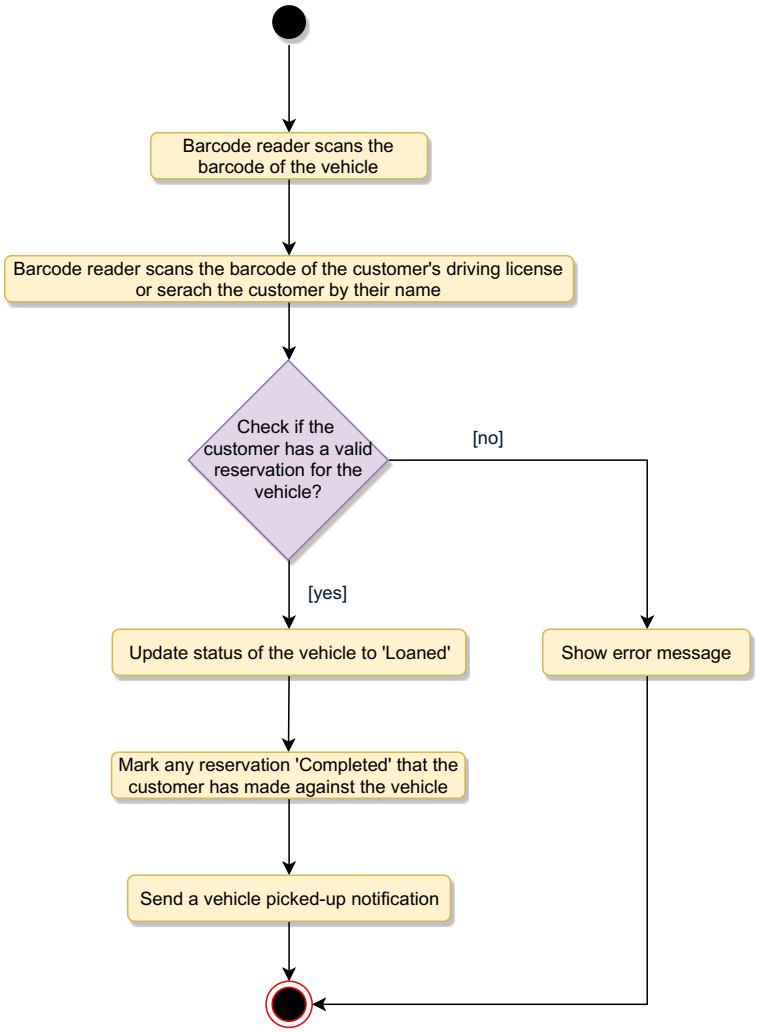
A ---------- B **Use Interface:** A uses interface B.

A —————— B **Association**: A and B call each other.

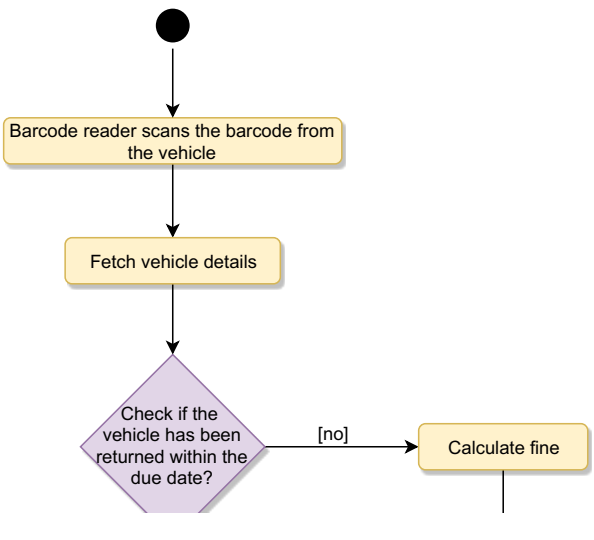


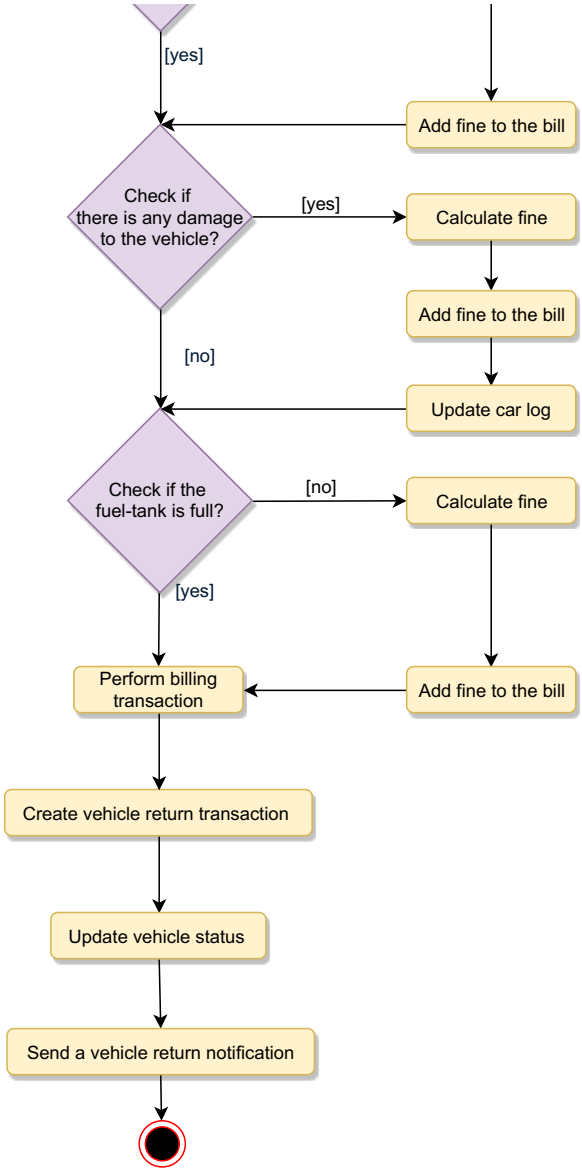

## Activity diagrams

**Pick up a vehicle:** Any member can perform this activity. Here are the steps to pick up a vehicle:




**Return a vehicle:** Any worker can perform this activity. While returning a vehicle, the system must collect a late fee from the member if the return date is after the due date. Here are the steps for returning a vehicle:






## Code

Here is the high-level definition for the classes described above.

**Enums, data types and constants:** Here are the required enums, data types, and constants:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

```
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
```

**Account, Member, Receptionist, and Additional Driver:** These classes represent different people that interact with our system:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
```

**CarRentalSystem and CarRentalLocation:** These classes represent the top level classes:

```
1
2
3
4
5
6
7
8
```

```
 9
10
11
12
13
14
```

**Vehicle, VehicleLog, and VehicleReservation:** To encapsulate a vehicle, log, and reservation. The VehicleReservation class will be responsible for processing the reservation and return of a vehicle:

```
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
```

```
60
61
62
63
64
65
66
67
68
```

**VehicleInventory and Search:** VehicleInventory will implement an interface ‘Search’ to facilitate the searching of vehicles:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

Mark as Completed

← Back

Next →

Design an Online Stock Brokerage Sys...

Design LinkedIn

Send feedback 4 Recommendations

Stuck? Get help on DISCUSS

# Design LinkedIn

Let's design LinkedIn.

LinkedIn is a social network for professionals. The main goal of the site is to enable its members to connect with people they know and trust professionally, as well as to find new opportunities to grow their careers.

A LinkedIn member's profile page, which emphasizes their skills, employment history, and education, has professional network news feeds with customizable modules.

LinkedIn is very similar to Facebook in terms of its layout and design. These features are more specialized because they cater to professionals, but in general, if you know how to use Facebook or any other similar social network, LinkedIn is somewhat comparable.


## System Requirements

We will focus on the following set of requirements while designing LinkedIn:

1. Each member should be able to add information about their basic profile, experiences, education, skills, and accomplishments.
2. Any user of our system should be able to search for other members or companies by their name.
3. Members should be able to send or accept connection requests from other members.
4. Any member will be able to request a recommendation from other members.
5. The system should be able to show basic stats about a profile, like the number of profile views, the total number of connections, and the total number of search appearances of the profile.
6. Members should be able to create new posts to share with their connections.
7. Members should be able to add comments to posts, as well as like or share a post or comment.
8. Any member should be able to send messages to other members.
9. The system should send a notification to a member whenever there is a new message, connection

invitation or a comment on their post.

10. Members will be able to create a page for a Company and add job postings.
11. Members should be able to create groups and join any group they like.
12. Members should be able to follow other members or companies.

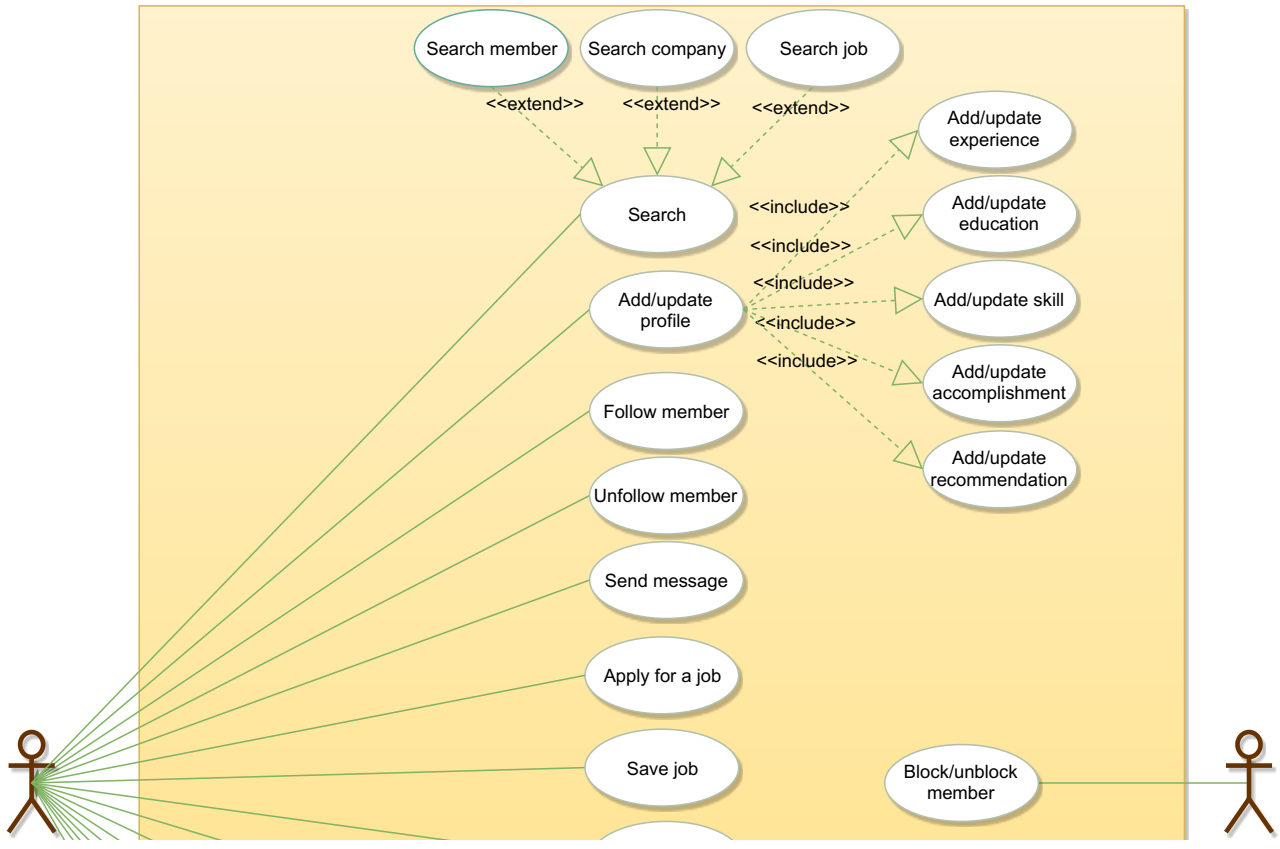
## Use case diagram

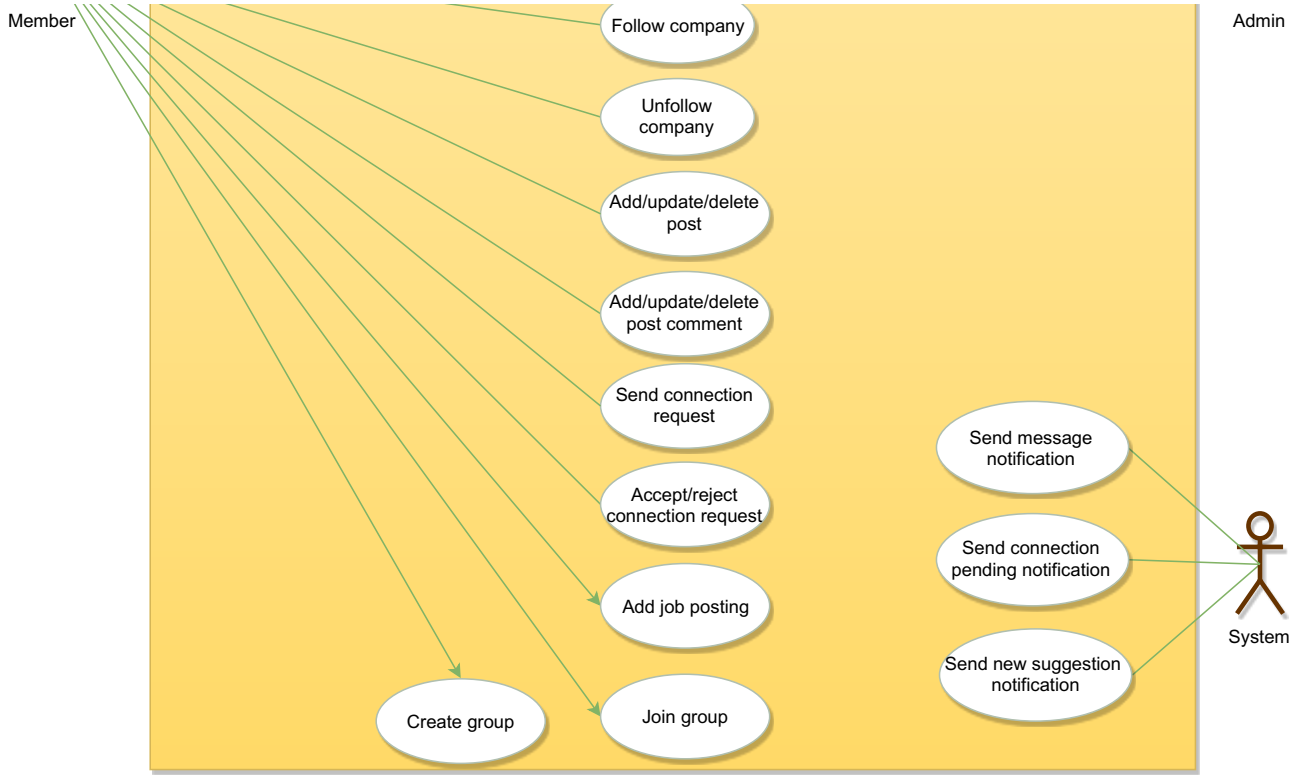
We have three main Actors in our system:

- **Member:** All members can search for other members, companies or jobs, as well as send requests for connection, create posts, etc.
- **Admin:** Mainly responsible for admin functions such as blocking and unblocking a member, etc.
- **System:** Mainly responsible for sending notifications for new messages, connections invites, etc.

Here are the top use cases of our system:

- **Add/update profile:** Any member should be able to create their profile to reflect their experiences, education, skills, and accomplishments.
- **Search:** Members can search other members, companies or jobs. Members can send a connection request to other members.
- **Follow or Unfollow member or company:** Any member can follow or unfollow any other member or a company.
- **Send message:** Any member can send a message to any of their connections.
- **Create post:** Any member can create a post to share with their connections, as well as like other posts or add comments to any post.
- **Send notifications:** The system will be able to send notifications for new messages, connection invites, etc.



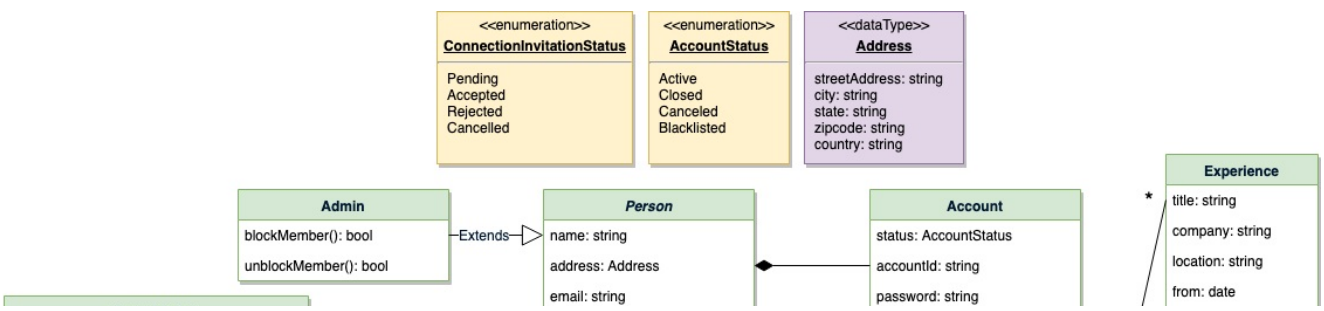


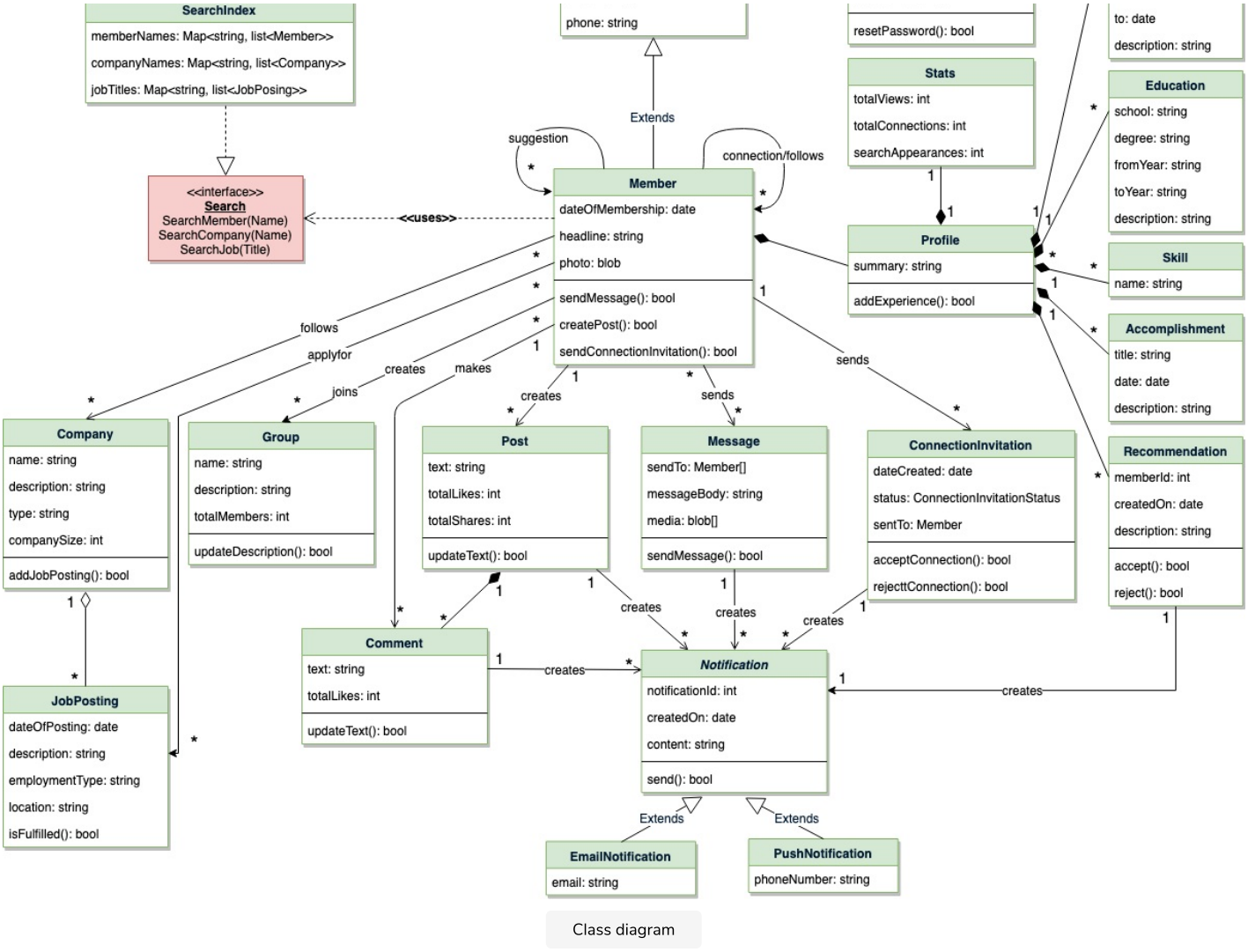

Use case diagram

## Class diagram

Here are the main classes of the LinkedIn system:

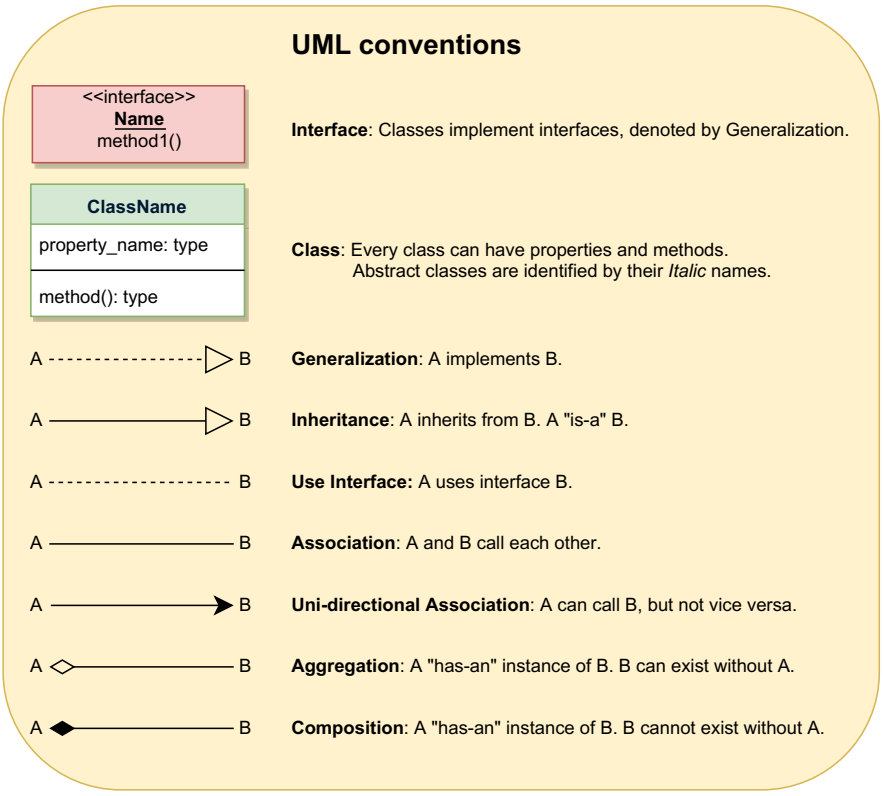
- **Member:** This will be the main component of our system. Each member will have a profile which includes their Experiences, Education, Skills, Accomplishments, and Recommendations. Members will be connected to other members and they can follow companies and members. Members will also have suggestions to make connections with other members.
- **Search:** Our system will support searching for other members and companies by their names, and jobs by their titles.
- **Message:** Members can send messages to other members with text and media.
- **Post:** Members can create posts containing text and media.
- **Comment:** Members can add comments to posts as well as like them.
- **Group:** Members can create and join groups.
- **Company:** Company will store all the information about a company's page.
- **JobPosting:** Companies can create a job posting. This class will handle all information about a job.
- **Notification:** Will take care of sending notifications to members.






Class diagram

## UML conventions




**Interface**: Classes implement interfaces, denoted by Generalization.

**Class**: Every class can have properties and methods.
Abstract classes are identified by their *Italic* names.

**Generalization**: A implements B.

**Inheritance**: A inherits from B. A "is-a" B.

**Use Interface:** A uses interface B.

**Association**: A and B call each other.

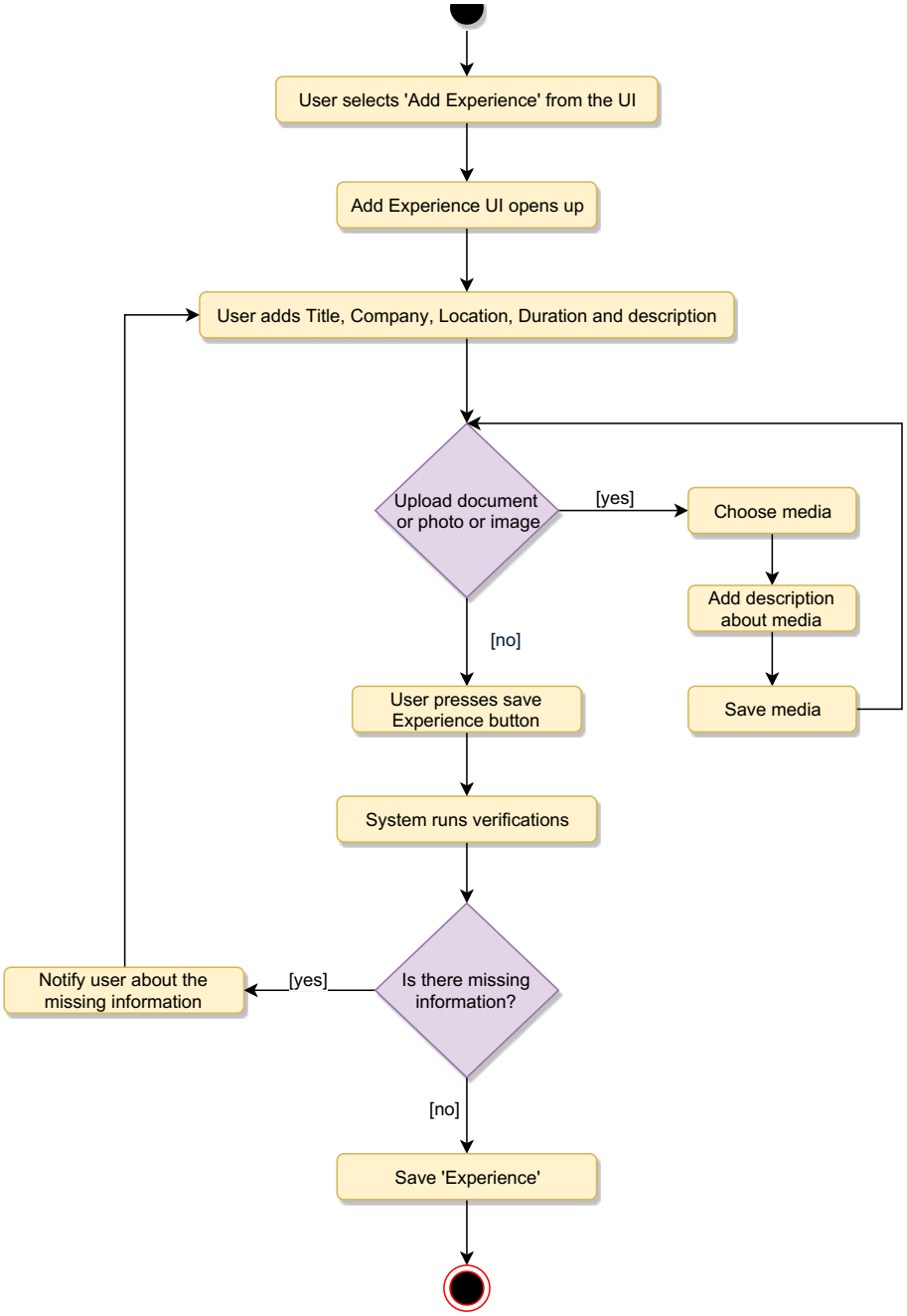
**Uni-directional Association**: A can call B, but not vice versa.

**Aggregation**: A "has-an" instance of B. B can exist without A.

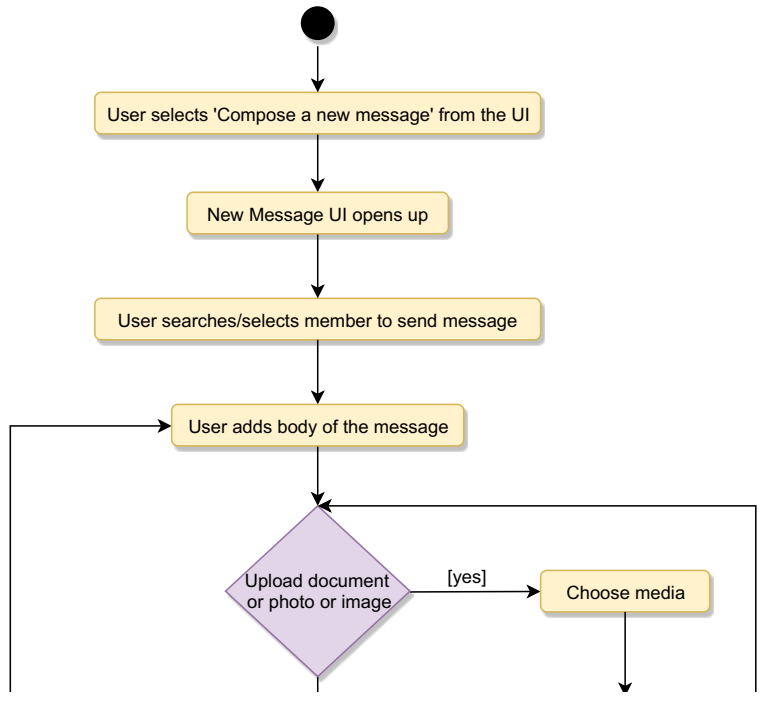
**Composition**: A "has-an" instance of B. B cannot exist without A.

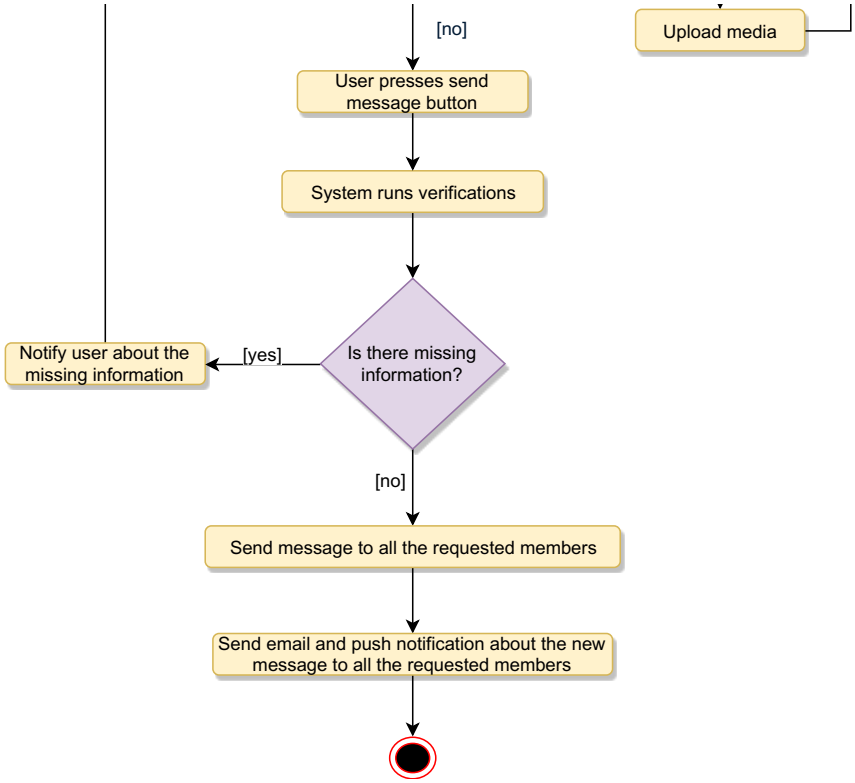
## Activity diagrams

**Add experience to profile:** Any LinkedIn member can perform this activity. Here are the steps to add experience to a member profile:




**Send message:** Any Member can perform this activity. After sending a message, the system needs to send a notification to all the requested members. Here are the steps for sending a message:






## Code

Here is the high-level definition for the classes described above:

**Enums, data types, and constants:** Here are the required enums, data types, and constants:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

**Account, Person, Member, and Admin:** These classes represent the different people that interact with our system:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
```

```
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
```

**Profile, Experience, etc:** A member's profile will have their job experiences, educations, skills, etc:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

**Company and JobPosting:** Companies can have multiple job postings:

```
1
2
3
4
5
```

```
```

**Group, Post, and Message:** Members can create posts, send messages, and join groups:

```
```

**Search interface and SearchIndex:** SearchIndex will implement the Search interface to facilitate searching for members, companies and job postings:

```
```

```
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
```

Mark as Completed

← Back

Design a Car Rental System

Next →

Design Cricinfo

Send feedback 8 Recommendations

Stuck? Get help on DISCUSS

# Design Cricinfo

Let's design Cricinfo.

Cricinfo is a sports news website exclusively for the game of cricket. The site features live coverage of cricket matches containing ball-by-ball commentary and a database for all the historic matches. The site also provides news and articles about cricket.


## System Requirements

We will focus on the following set of requirements while designing Cricinfo:

1. The system should keep track of all cricket-playing teams and their matches.
2. The system should show live ball-by-ball commentary of cricket matches.
3. All international cricket rules should be followed.
4. Any team playing a tournament will announce a squad (a set of players) for the tournament.
5. For each match, both teams will announce their playing-eleven from the tournament squad.
6. The system should be able to record stats about players, matches, and tournaments.
7. The system should be able to answer global stats queries like, “Who is the highest wicket taker of all time?”, “Who has scored maximum numbers of 100s in test matches?”, etc.
8. The system should keep track of all ODI, Test and T20 matches.

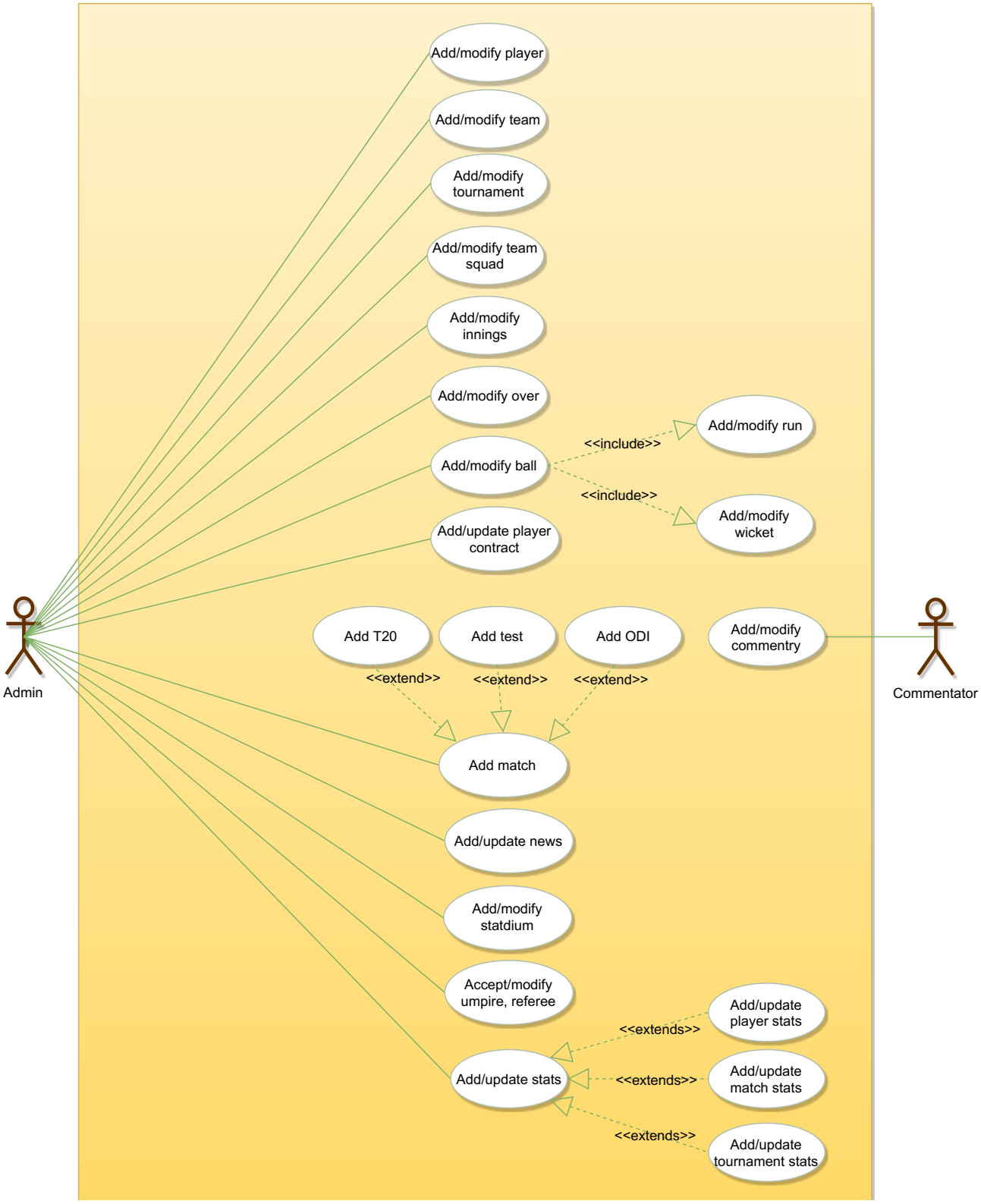
## Use case diagram

We have two main Actors in our system:

- **Admin:** An Admin will be able to add/modify players, teams, tournaments, and matches, and will also record ball-by-ball details of each match.
- **Commentator:** Commentators will be responsible for adding ball-by-ball commentary for matches.

Here are the top use cases of our system:

- **Add/modify teams and players:** An Admin will add players to teams and keeps up-to-date information about them in the system.
- **Add tournaments and matches:** Admins will add tournaments and matches in the system.
- **Add ball:** Admins will record ball-by-ball details of a match.
- **Add stadium, umpire, and referee:** The system will keep track of stadiums as well as of the umpires and referees managing the matches.
- **Add/update stats:** Admins will add stats about matches and tournaments. The system will generate certain stats.
- **Add commentary:** Add ball-by-ball commentary of matches.

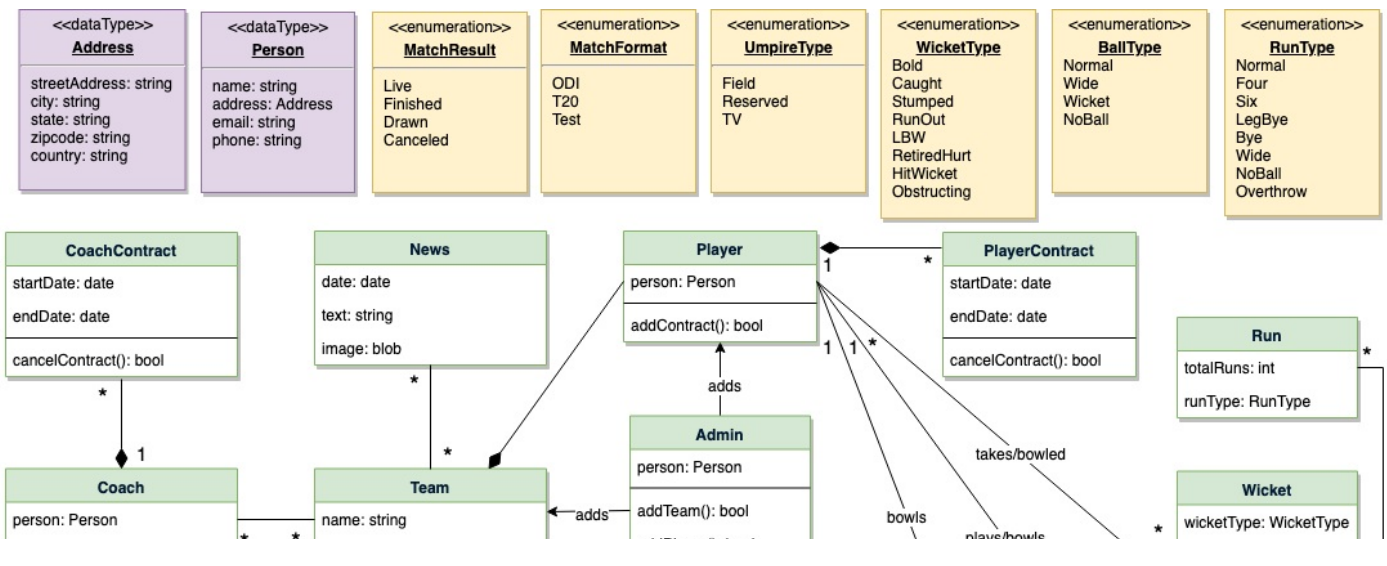


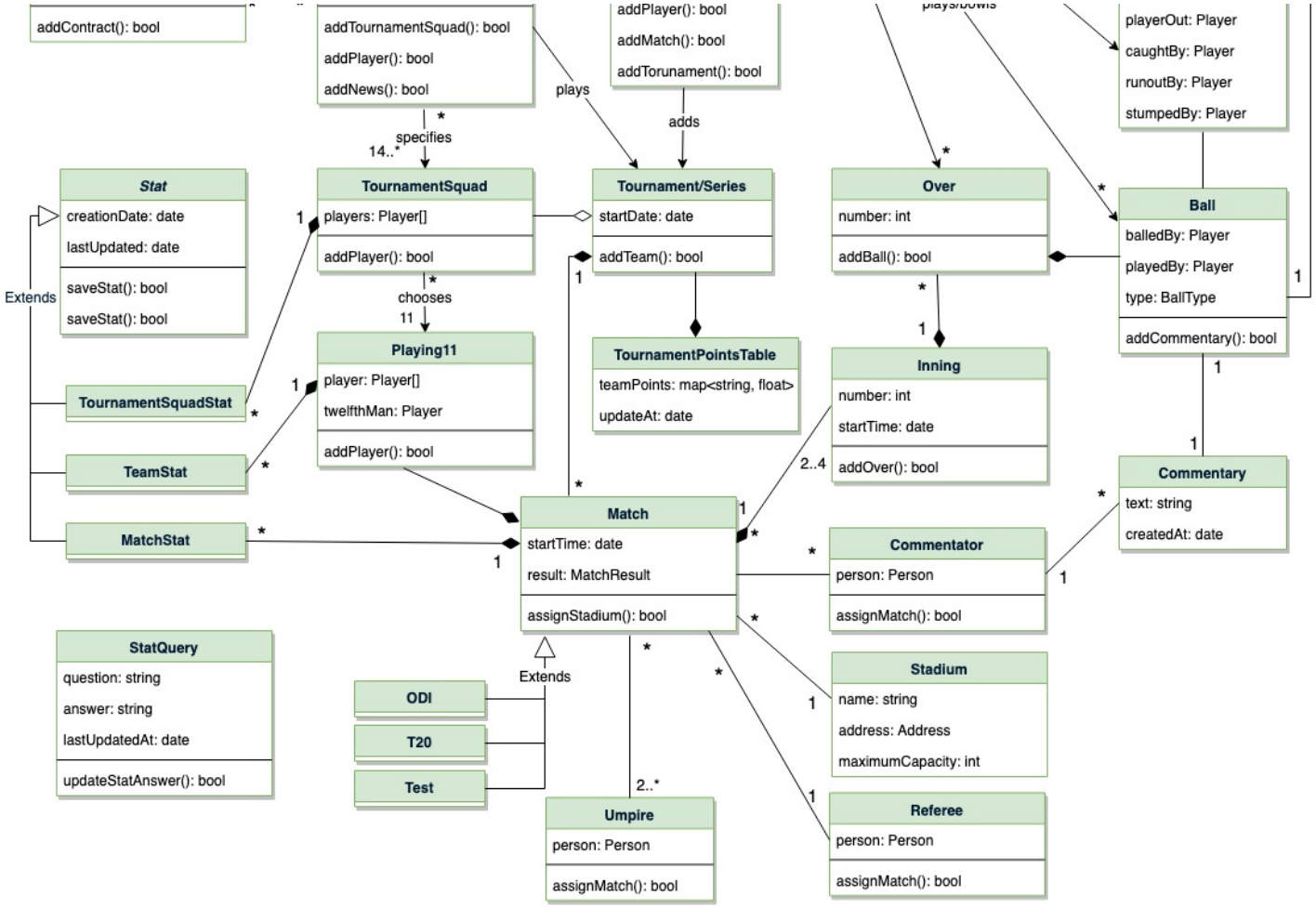
Use case diagram

## Class diagram

Here are the main classes of the Cricinfo system:

- **Player:** Keeps a record of a cricket player, their basic profile and contracts.
- **Team:** This class manages cricket teams.
- **Tournament:** Manages cricket tournaments and keeps track of the points table for all playing teams.
- **TournamentSquad:** Each team playing a tournament will announce a set of players who will be playing the tournament. TournamentSquad will encapsulate that.
- **Playing11:** Each team playing a match will select 11 players from their announced tournaments squad.
- **Match:** Encapsulates all information of a cricket match. Our system will support three match types: 1) ODI, 2) T20, and 3) Test
- **Innings:** Records all innings of a match.
- **Over:** Records details about an Over.
- **Ball:** Records every detail of a ball, such as the number of runs scored, if it was a wicket-taking ball, etc.
- **Run:** Records the number and type of runs scored on a ball. The different run types are: Wide, LegBy, Four, Six, etc.
- **Commentator and Commentary:** The commentator adds ball-by-ball commentary.
- **Umpire and Referee:** These classes will store details about umpires and referees, respectively.
- **Stat:** Our system will keep track of the stats for every player, match and tournament.
- **StatQuery:** This class will encapsulate general stat queries and their answers, like “Who has scored the maximum number of 100s in ODIs?” or, “Which bowler has taken the most wickets in test matches?”, etc.






Class diagram

## UML conventions

<<interface>>
**Name**
method1()

**Interface**: Classes implement interfaces, denoted by Generalization.

**ClassName**
property_name: type
method(): type

**Class**: Every class can have properties and methods.
Abstract classes are identified by their *Italic* names.

A ----------▷ B **Generalization**: A implements B.

A ——————▷ B **Inheritance**: A inherits from B. A "is-a" B.

A ---------- B **Use Interface:** A uses interface B.

A —————— B **Association**: A and B call each other.

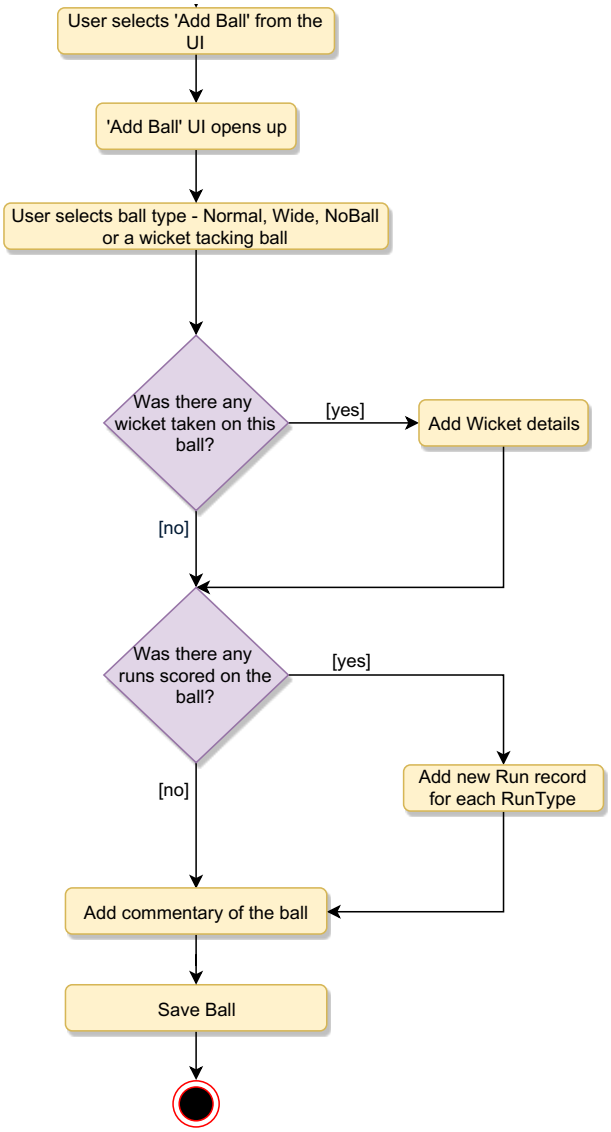
A ——————→ B **Uni-directional Association**: A can call B, but not vice versa.

A ◇—————— B **Aggregation**: A "has-an" instance of B. B can exist without A.

A ◆—————— B **Composition**: A "has-an" instance of B. B cannot exist without A.

## Activity diagrams

**Record a Ball of an Over:** Here are the steps to record a ball of an over in the system:




## Code

Here is the high-level definition for the classes described above.

**Enums, data types, and constants:** Here are the required enums, data types, and constants:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
```

**Admin, Player, Umpire, Referee, and Commentator:** These classes represent the different people that interact with our system:

```
32
33
34
35
36
37
38
```

**Team, TournamentSquad, and Playing11:** Team will announce a squad for a tournament, out of which, the playing 11 will be chosen:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

**Over, Ball, Wicket, Commentary, Inning, and Match:** Match will be an abstract class, extended by ODI, Test, and T20:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
```

```
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
```

Mark as Completed

← Back

Design LinkedIn

Next →

Design Facebook - a social network

Send feedback

6 Recommendations

Stuck? Get help on DISCUSS

# Design Facebook - a social network

Facebook is an online social networking service where users can connect with other users to post and read messages. Users access Facebook through their website interface or mobile apps.


## System Requirements

We will focus on the following set of requirements while designing Facebook:

1. Each member should be able to add information about their basic profile, work experience, education, etc.
2. Any user of our system should be able to search other members, groups or pages by their name.
3. Members should be able to send and accept/reject friend requests from other members.
4. Members should be able to follow other members without becoming their friend.
5. Members should be able to create groups and pages, as well as join already created groups, and follow pages.
6. Members should be able to create new posts to share with their friends.
7. Members should be able to add comments to posts, as well as like or share a post or comment.
8. Members should be able to create privacy lists containing their friends. Members can link any post with a privacy list to make the post visible only to the members of that list.
9. Any member should be able to send messages to other members.
10. Any member should be able to add a recommendation for any page.
11. The system should send a notification to a member whenever there is a new message or friend request or comment on their post.

12. Members should be able to search through posts for a word.

**Extended Requirement:** Write a function to find a connection suggestion for a member.

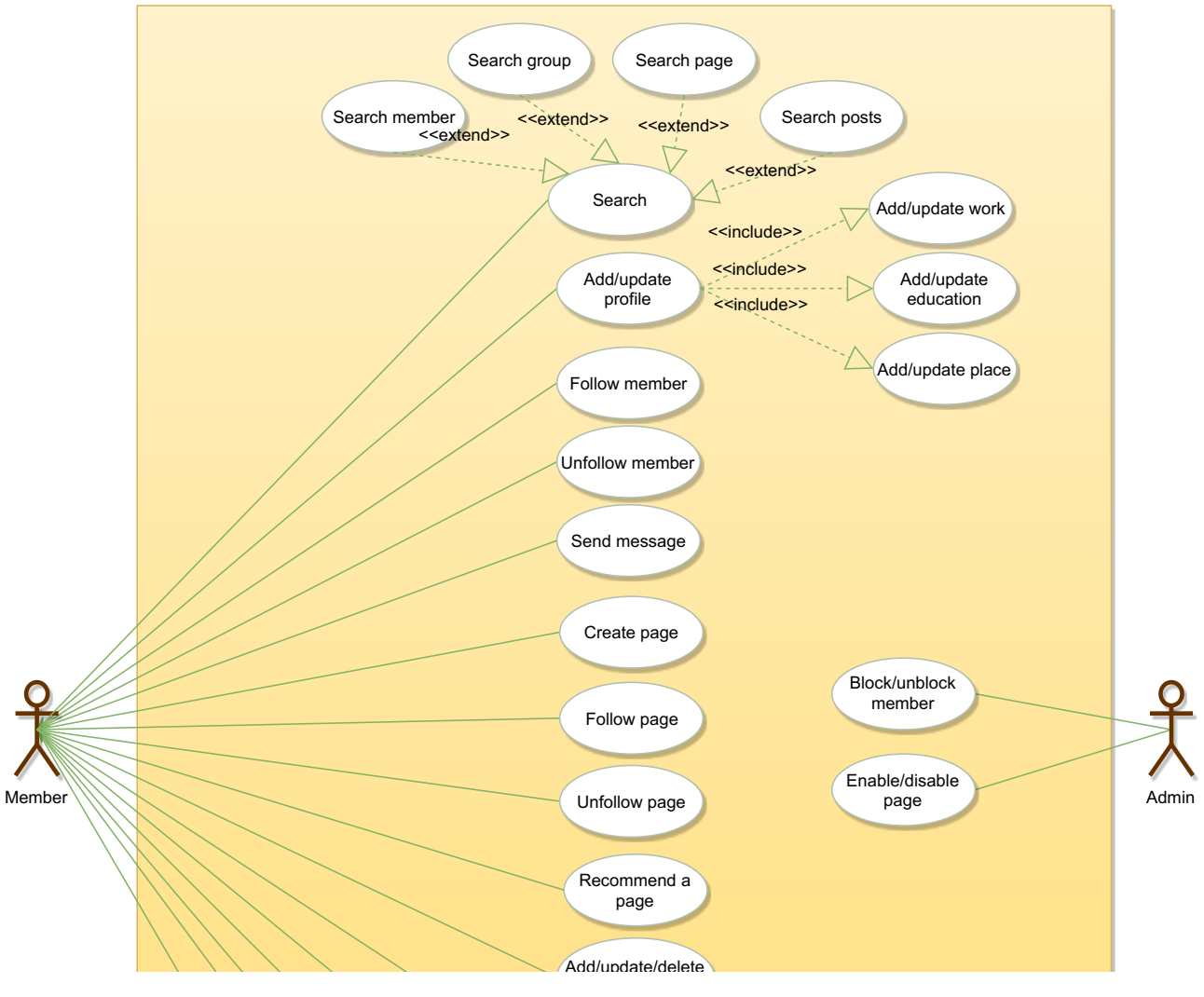
## Use case diagram

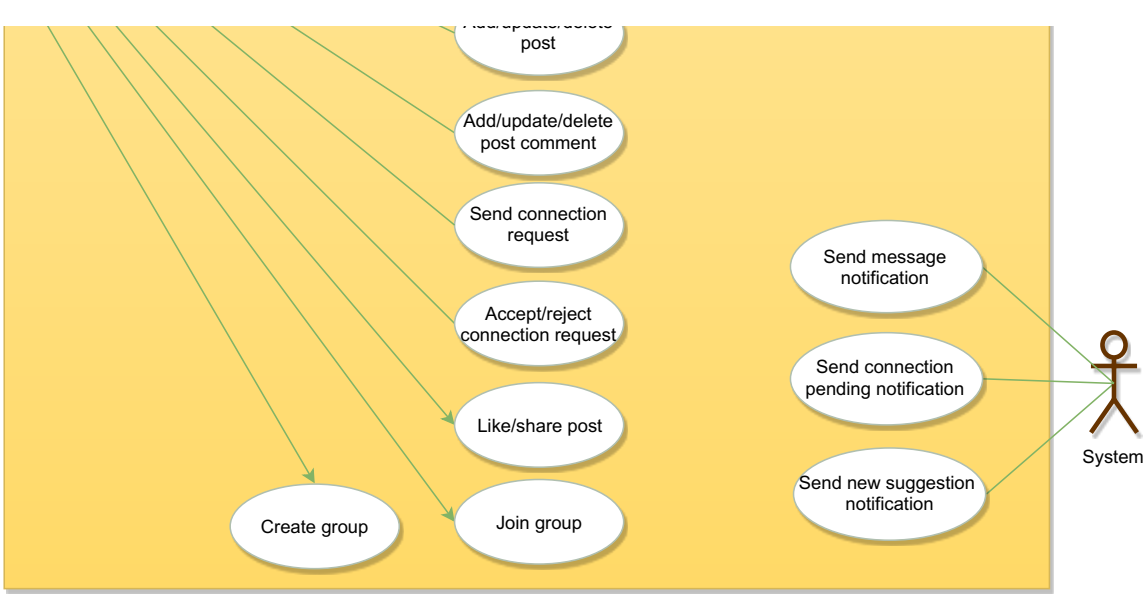
We have three main Actors in our system:

- **Member:** All members can search for other members, groups, pages, or posts, as well as send friend requests, create posts, etc.
- **Admin:** Mainly responsible for admin functions like blocking and unblocking a member, etc.
- **System:** Mainly responsible for sending notifications for new messages, friend requests, etc.

Here are the top use cases of our system:

- **Add/update profile:** Any member should be able to create their profile to reflect their work experiences, education, etc.
- **Search:** Members can search for other members, groups or pages. Members can send a friend request to other members.
- **Follow or Unfollow a member or a page:** Any member can follow or unfollow any other member or page.
- **Send message** Any member can send a message to any of their friends.
- **Create post** Any member can create a post to share with their friends, as well as like or add comments to any post visible to them.
- **Send notification** The system will be able to send notifications for new messages, friend requests, etc.

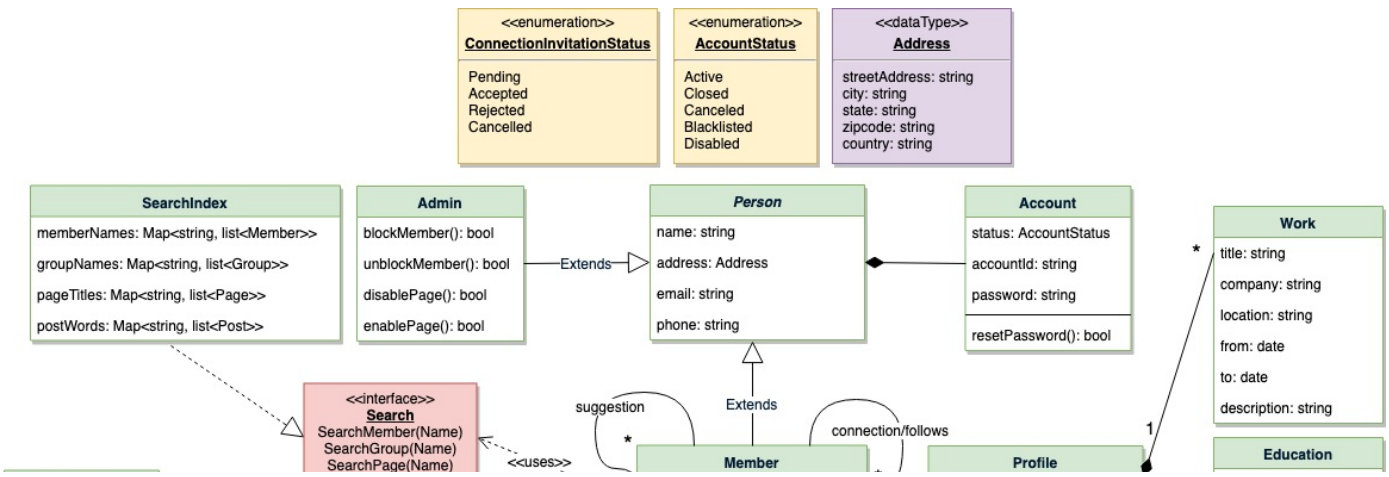


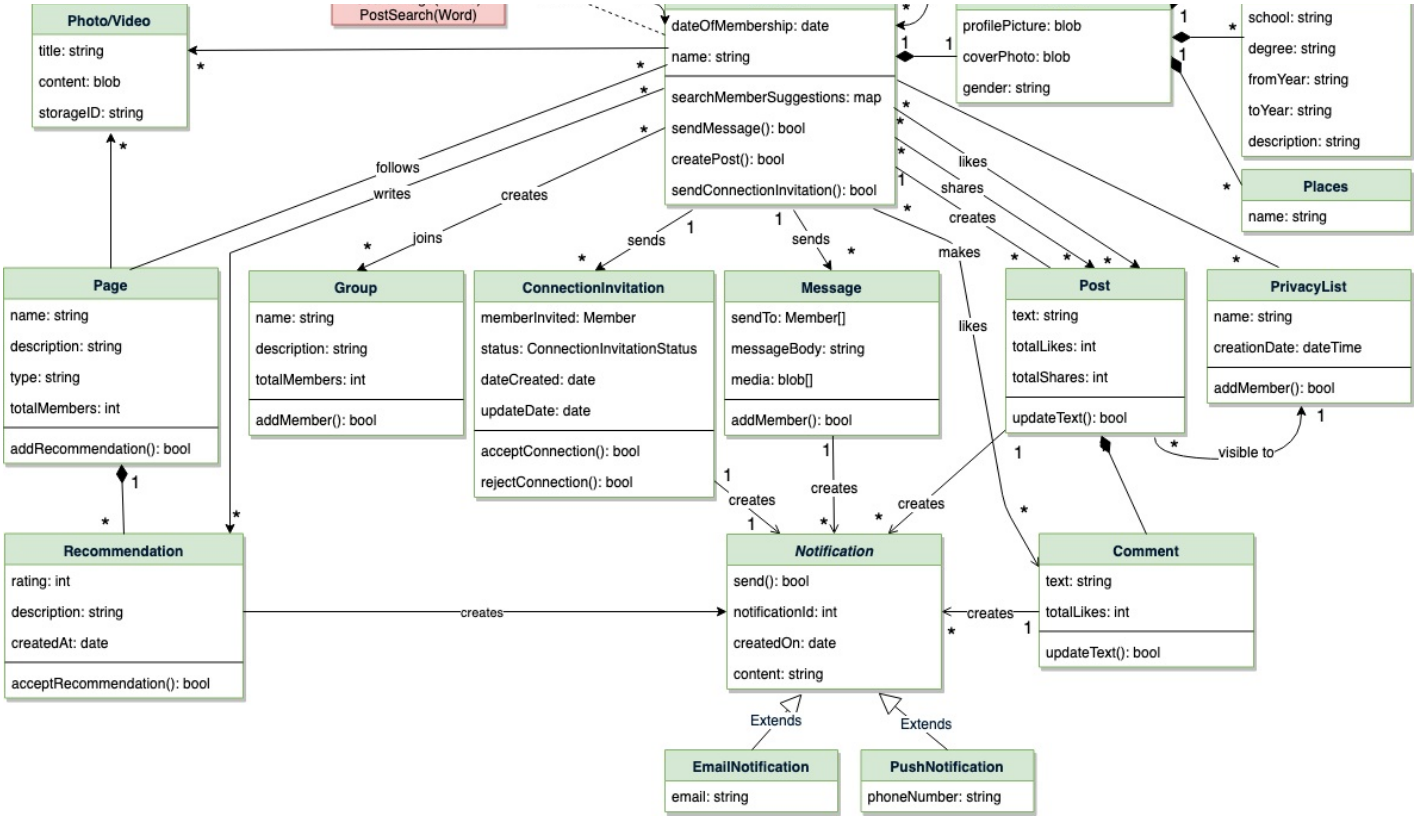



## Class diagram

Here are the main classes of the Facebook system:

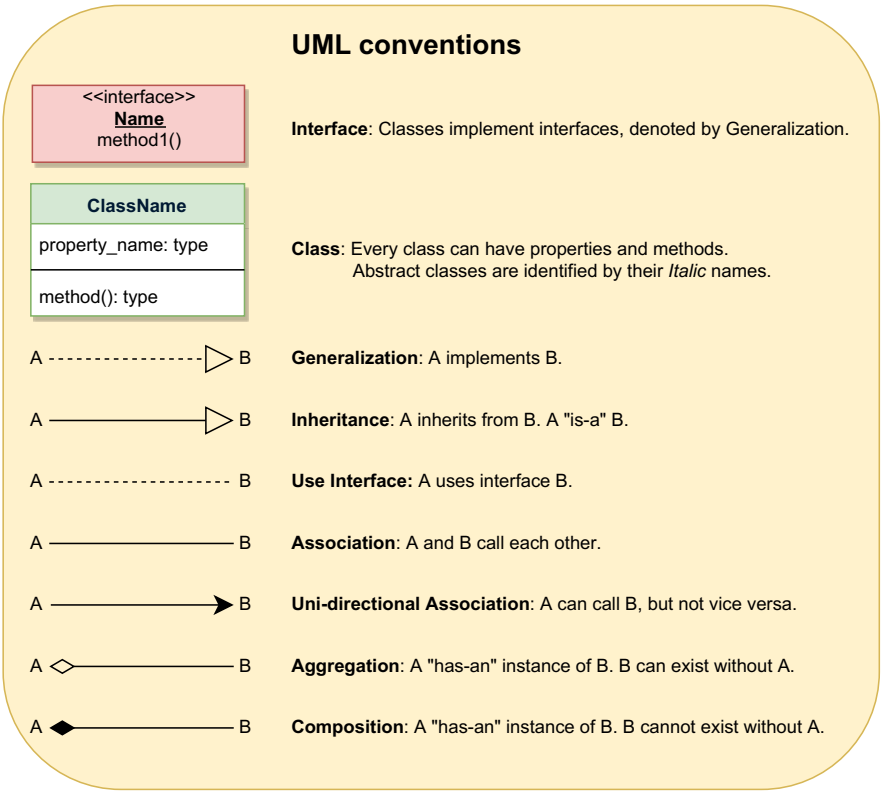
- **Member:** This will be the main component of our system. Each member will have a profile which includes their Work Experiences, Education, etc. Members will be connected to other members and they can follow other members and pages. Members will also have suggestions to send friend requests to other members.
- **Search:** Our system will support searching for other members, groups and pages by their names, and through posts for any word.
- **Message:** Members can send messages to other members with text, photos, and videos.
- **Post:** Members can create posts containing text and media, as well as like and share a post.
- **Comment:** Members can add comments to posts as well as like any comment.
- **Group:** Members can create and join groups.
- **PrivacyList:** Members can create privacy lists containing their friends. Members can link any post with a privacy list, to make the post visible only to the members of that list.
- **Page:** Members can create pages that other members can follow, and share messages there.
- **Notification:** This class will take care of sending notifications to members. The system will be able to send a push notification or an email.





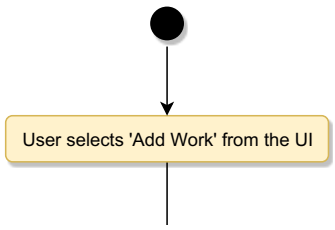

Class diagram

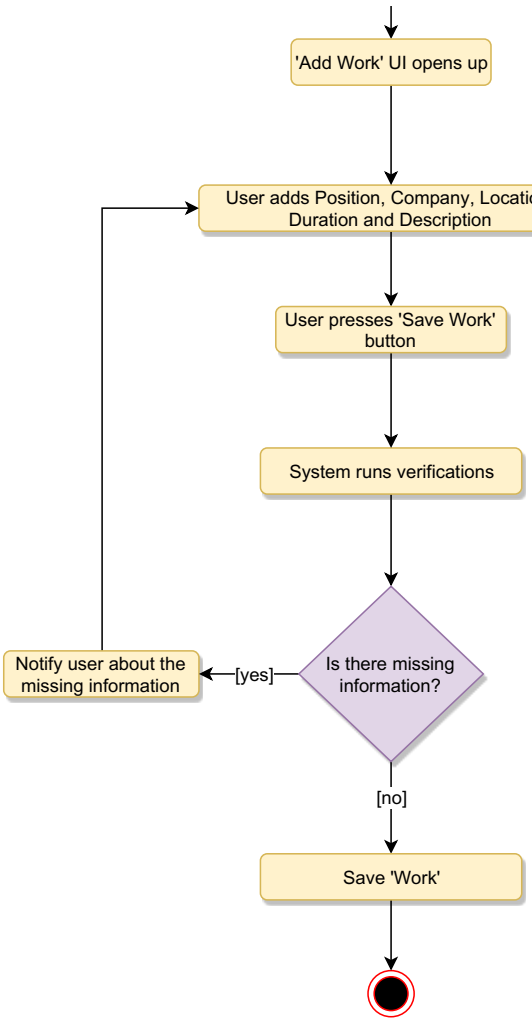
## UML conventions



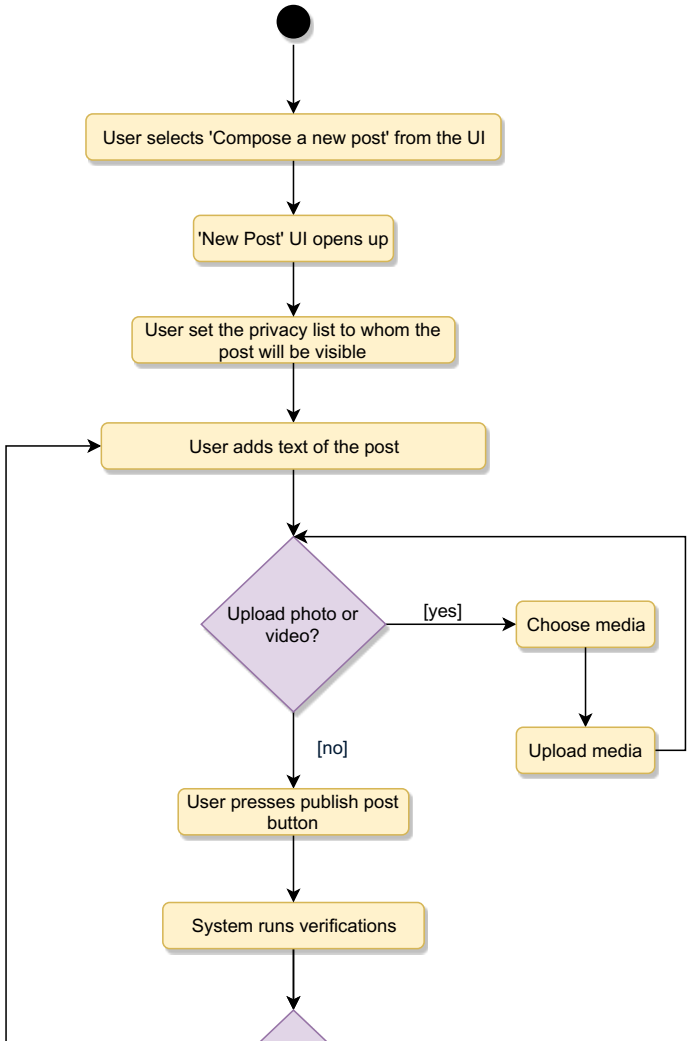

## Activity diagrams

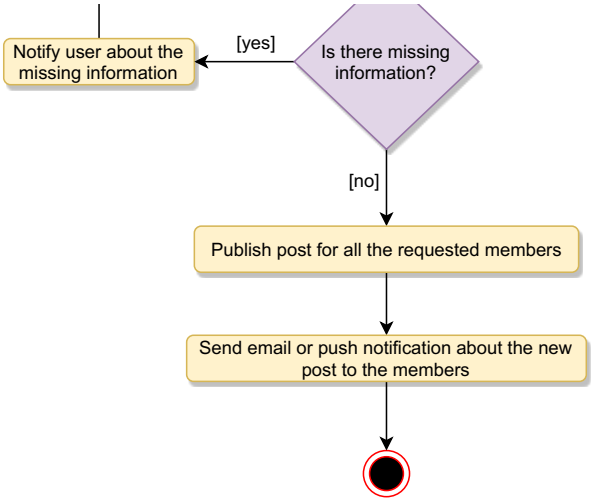
**Add work experience to profile:** Any Facebook member can perform this activity. Here are the steps to add work experience to a member's profile:






**Create a new post:** Any Member can perform this activity. Here are the steps for creating a post:






## Code

Here is the high-level definition for the classes described above.

**Enums, data types, and constants:** Here are the required enums, data types, and constants:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```

**Account, Person, Member, and Admin:** These classes represent the different people that interact with our system:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

**Profile and Work:** A member's profile will have their work experiences, educations, places, etc:

**Page and Recommendation:** Each page can have multiple recommendations, and members will follow/like pages:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```

**Group, Post, Message, and Comment:** Members can create posts, comment on posts, send messages and join groups:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
```

**Search interface and SearchIndex:** SearchIndex will implement Search to facilitate searching of members, groups, pages, and posts:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
```

## Extended requirement

Here is the code for finding connection suggestions for a member.

There can be many strategies to search for connection suggestions; we will do a two-level deep breadth-first search to find people who have the most connections with each other. These people could be good candidates for a connection suggestion, here is the sample Java code:

```
1
2
3
4
5
6
7
8
9
10
11
12
```

```
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
```

Mark as Completed

← Back

Next →

Design Cricinfo

Contact us

Send feedback

10 Recommendations

Stuck? Get help on DISCUSS