Design an Online Stock Brokerage System

Let's design an Online Stock Brokerage System.

We'll cover the following

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

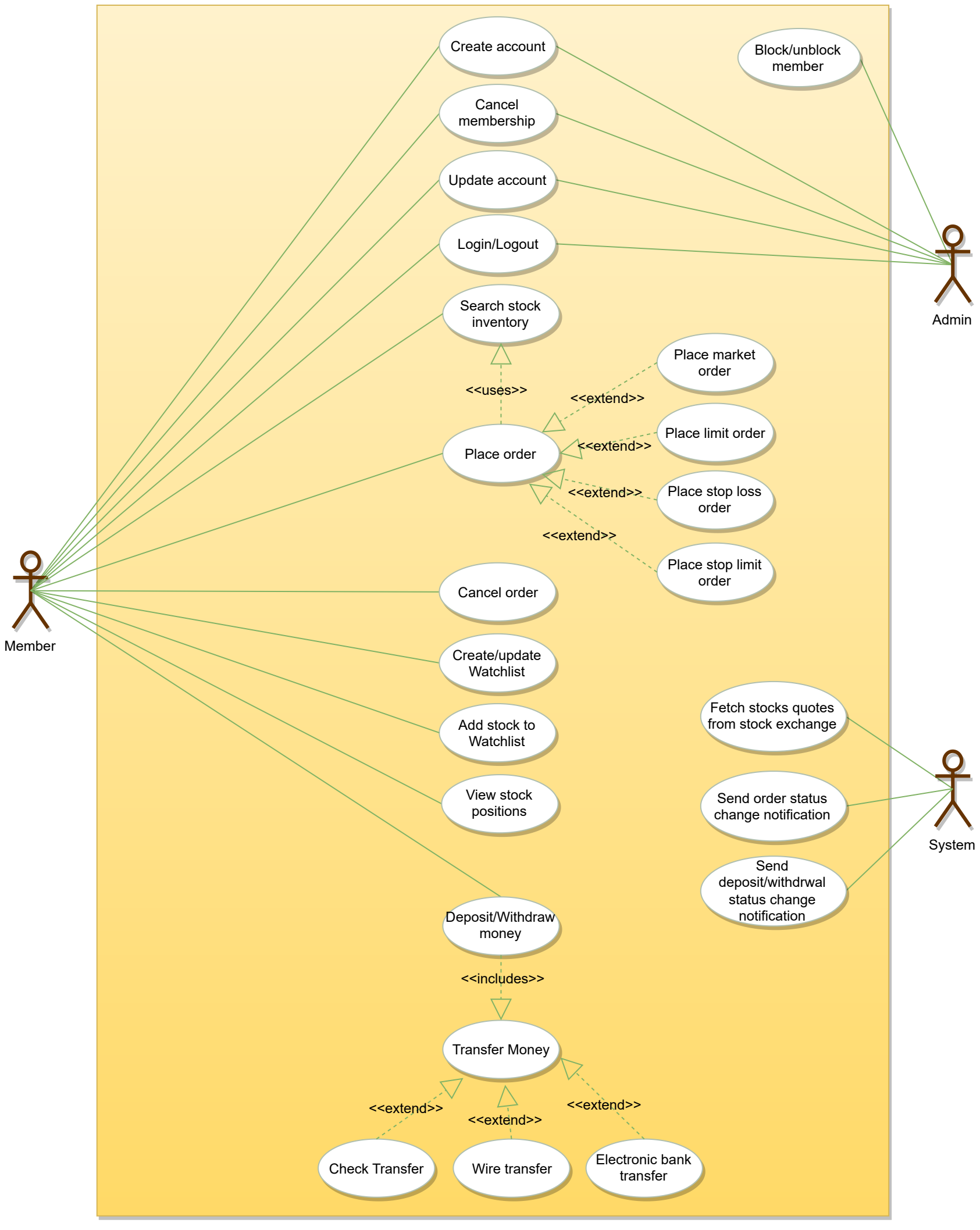
We have three main Actors in our system:

- **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.

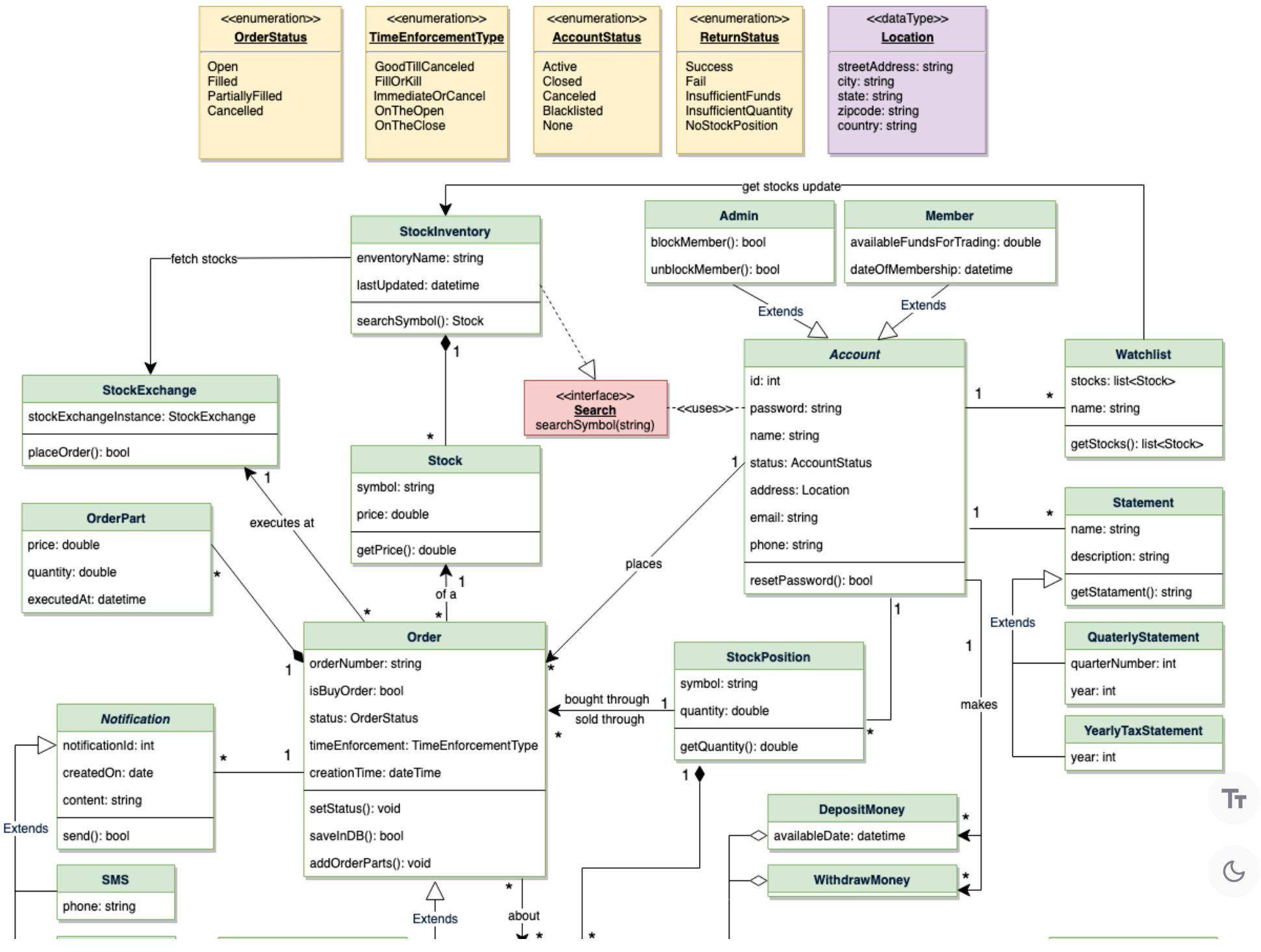


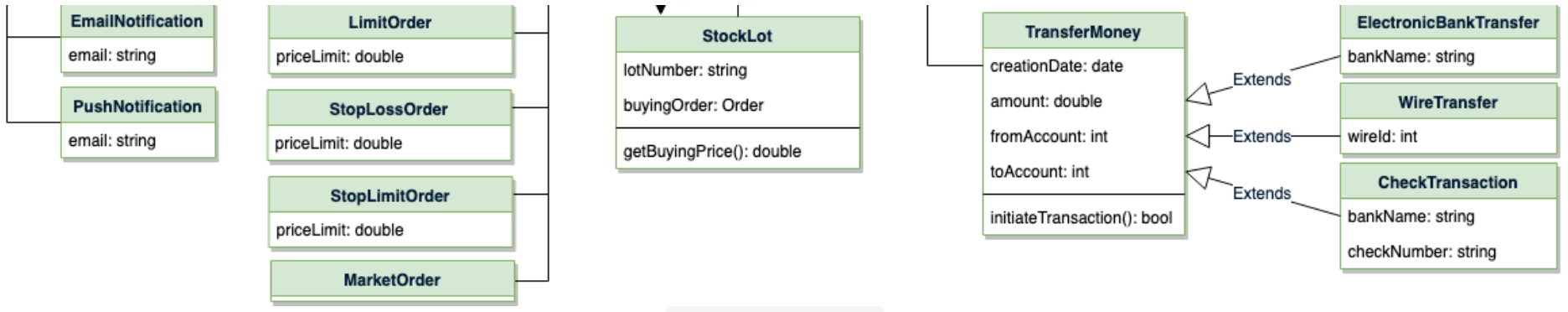
Use case diagram for Stock Brokerage System

Class diagram

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

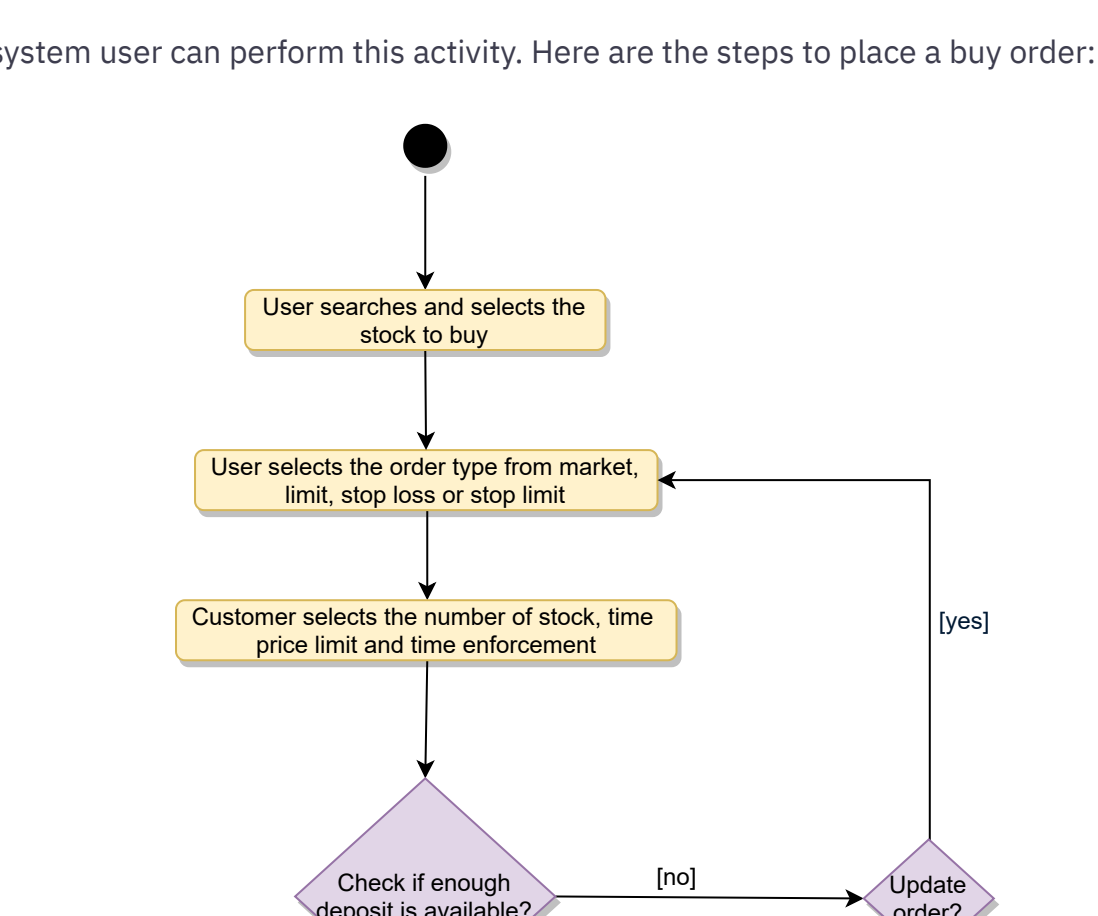
- **Account:** Consists of the member's name, address, e-mail, phone, total funds, funds that are available for trading, etc. We'll have two types of accounts in the system: one will be a general member, and the other will be an Admin. The Account class will also contain all the stocks the member is 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.
- **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.

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

Use Interface: A uses interface B.

Association: A and B call each other.

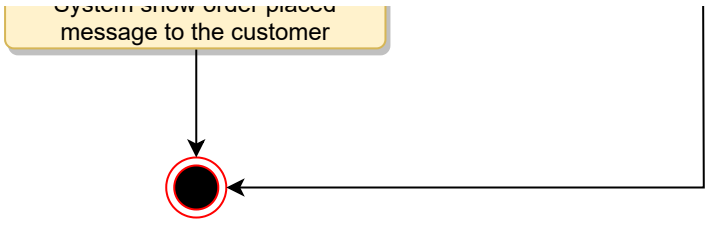
Uni-directional Association: A can call B, but not vice versa.

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

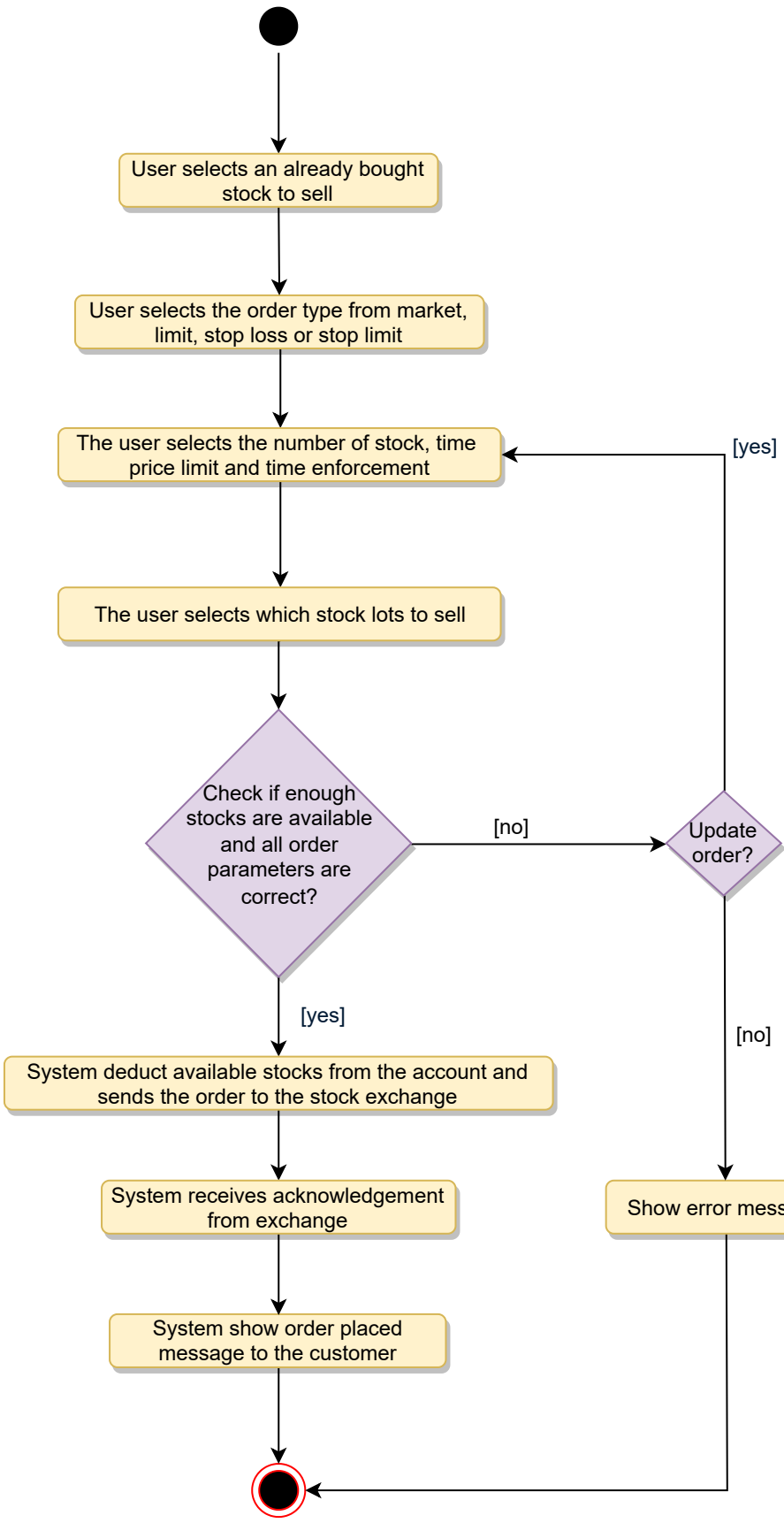
Composition: A "has-an" instance of B. B cannot exist without A.

Activity diagrams

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:

Java | Python

```
class ReturnStatus(Enum):
  SUCCESS, FAIL, INSUFFICIENT_FUNDS, INSUFFICIENT_QUANTITY, NO_STOCK_POSITION = 1, 2, 3, 4, 5, 6


class OrderStatus(Enum):
  OPEN, FILLED, PARTIALLY_FILLED, CANCELLED = 1, 2, 3, 4


class TimeEnforcementType(Enum):
  GOOD_TILL_CANCELLED, FILL_OR_KILL, IMMEDIATE_OR_CANCEL, ON_THE_OPEN, ON_THE_CLOSE = 1, 2, 3, 4, 5


class AccountStatus(Enum):
```

```
    ACTIVE, CLOSED, CANCELED, BLACKLISTED, NONE = 1, 2, 3, 5


class Location:
  def __init__(self, street, city, state, zip_code, country):
    self.__street_address = street
    self.__city = city
    self.__state = state
    self.__zip_code = zip_code
    self.__country = country


class Constants:
```

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

Java | Python

```
class StockExchange:
  # singleton, used for restricting to create only one instance
  instance = None

  class __OnlyOne:
    def __init__(self):
      None

  def __init__(self):
    if not StockExchange.instance:
      StockExchange.instance = StockExchange.__OnlyOne()

  def place_order(self, order):
    return_status = self.get_instance().submit_order(Order)
    return return_status
```

Order: To encapsulate all buy or sell orders:

Java | Python

```
from abc import ABC, abstractmethod
import datetime

class Order(ABC):
  def __init__(self, id):
    self.__order_id = id
    self.__is_buy_order = False
    self.__status = OrderStatus.OPEN
    self.__time_enforcement = TimeEnforcementType.ON_THE_OPEN
    self.__creation_time = datetime.datetime.now()

    self.__parts = {}

  def set_status(self, status):
    self.status = status

  def save_in_DB(self):
  # save in the database

  def add_order_parts(self, parts):
    for part in parts:
      self.parts[part.get_id()] = part


class LimitOrder(Order):
  def __init__(self):
    self.__price_limit = 0.0
```

Member: Members will be buying and selling stocks:

Java | Python

```
from abc import ABC, abstractmethod

class Account(ABC):
  def __init__(self, id, password, name, address, email, phone, status=AccountStatus.NONE):
    self.__id = id
    self.__password = password
    self.__name = name
    self.__address = address
    self.__email = email
    self.__phone = phone
    self.__status = AccountStatus.NONE

  def reset_password(self):
    None

import datetime

class Member(Account):
  def __init__(self):
    self.__available_funds_for_trading = 0.0
    self.__date_of_membership = datetime.date.today()
    self.__stock_positions = {}
    self.__active_orders = {}

  def place_sell_limit_order(self, stock_id, quantity, limit_price, enforcement_type):
    # check if member has this stock position
    if stock_id not in  stock_positions:
```

Back

Design Chess

Next

Design a Car Rental System

Mark as Completed