An Online Stock Brokerage System

→ An online stock brokerage system acts as a intermediary between the buyer & seller during the trade of the stock.

→ It enables users to monitor & carryout their transactions & displays performance graph for the various stocks in their portfolios.

→ The system facilitates its users to buy & sell stocks online.

→ It enables users to monitor & carry out their transactions & display performance graphs for various stock in their portfolios.

→ The method provides quicker access to stock information, current market trends, & current stock prices.

Expectations from the interviewee

→ How do the members search the stock inventory?

→ How will the search surface result?

→ Can every member see the current levels of stock position at any time.

→ How many types of stock trade orders are the users able to place, for example, a market order, loss order, etc?

→ Can the members have multiple watchlists containing multiple stock quote?

→ Can a member buy multiple lot of the same stock at different times?

Requirement Collection

R1: The system should allow the user to easily trade in stocks (buy or sell the stocks)

R2: Users are allowed to have numerous watchlists consisting of different stock quots.

R3: Users may own different lots of the same stock. This implies that the system should be able to distinguish between several lots of the same stock if a user has purchased the same stock more than once.

R4: Every time a trade order is carried out, the system should be able to notify users.

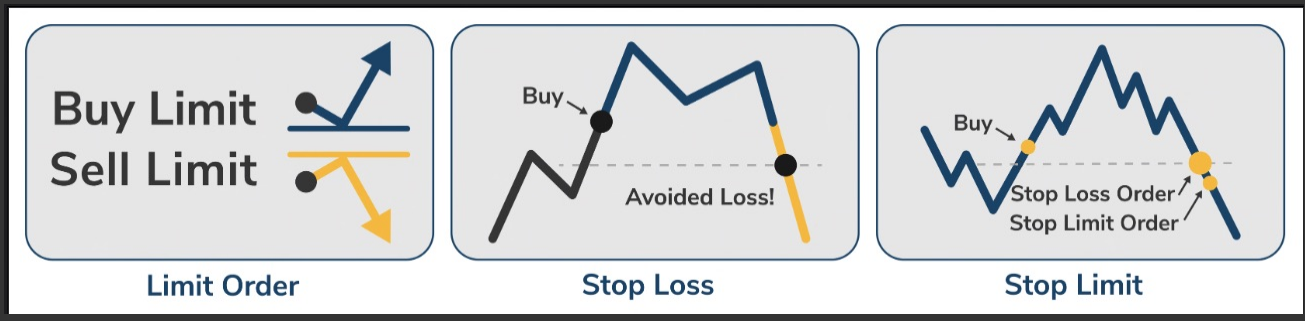
R5: The system should allow the users the order the stock trade of the types given below.

→ Market order: Buy or sell stocks at the current market price.

→ Limit Order: Buy or sell stocks at the price set by the user.

→ Stop-loss order: Buy or sell stocks when they reach at a certain price

→ Stop-limit orders: Buy or sell stocks with a restriction on the limit price (maximum price to be paid, minimum price to be received, etc)



R6: The system should allow the user to make deposit & withdrawals using checks, wire transfers, or electronic bank transfers.

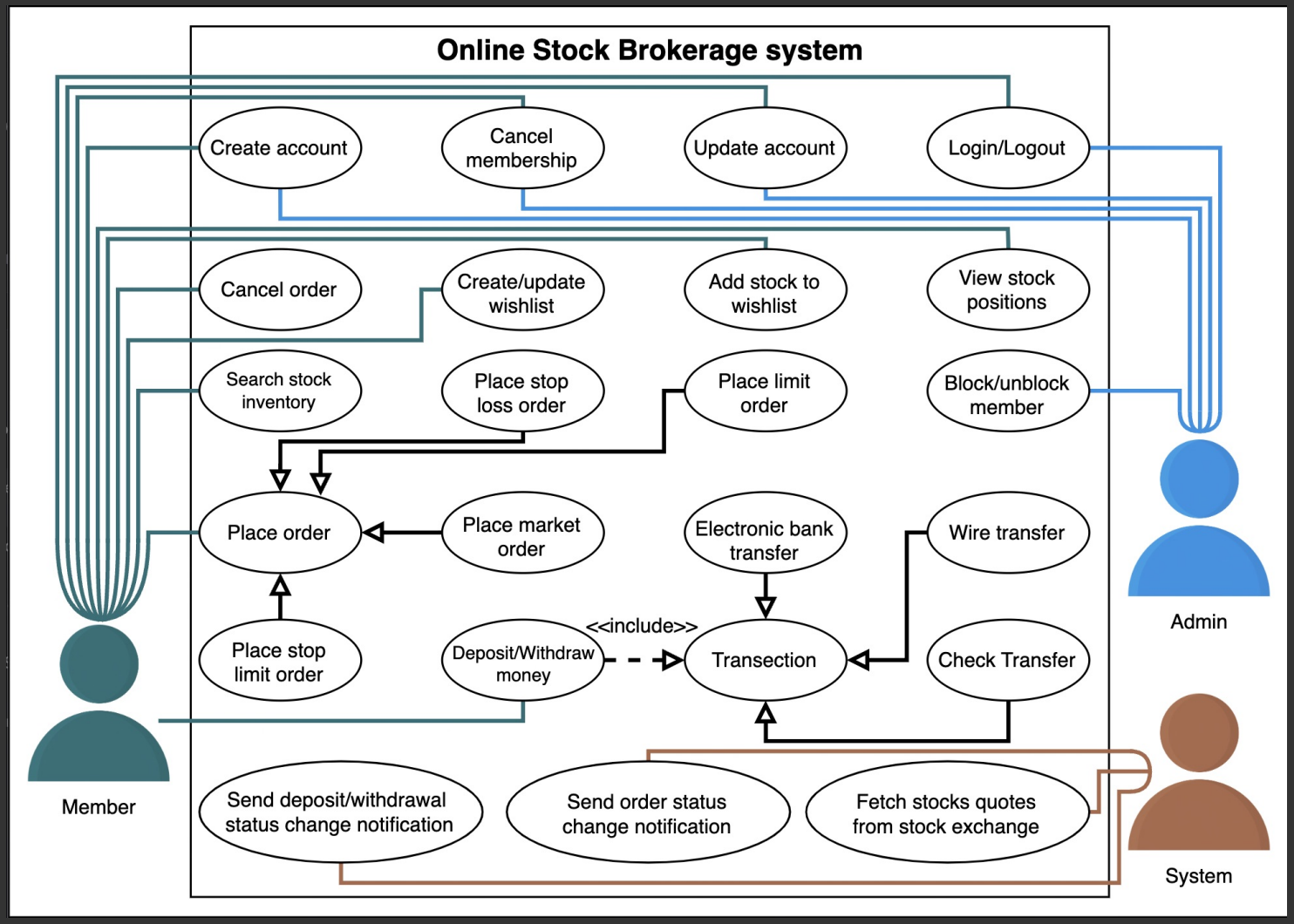
Actions:

→ Primary Actors : • Member

→ Secondary Actors : • Admin
• System

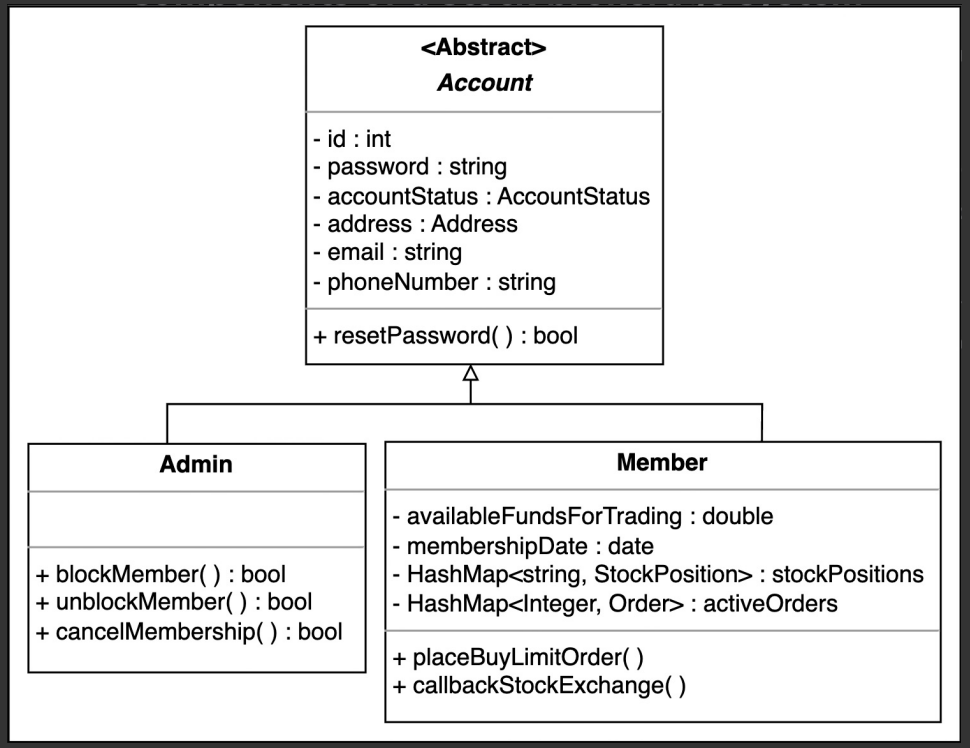
Member	Admin	System
Create account	Create account	Fetch stocks quotes from stock exchange
Cancel membership	Cancel membership	Send order status change notification
Update account	Update account	Send deposit/withdrawal status change notification
Login/Logout	Login/Logout	
Cancel order	Block/unblock member	
View stock positions		
Add stock to wishlist		
Create/update wishlist		
Search stock inventory		
Place order		
Deposit/withdraw money		

→ Use Case Diagram

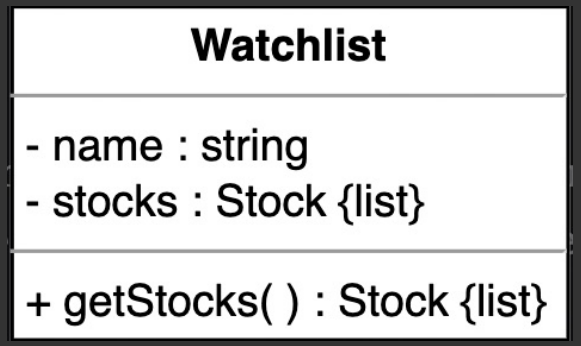


Class Diagram

1. Account

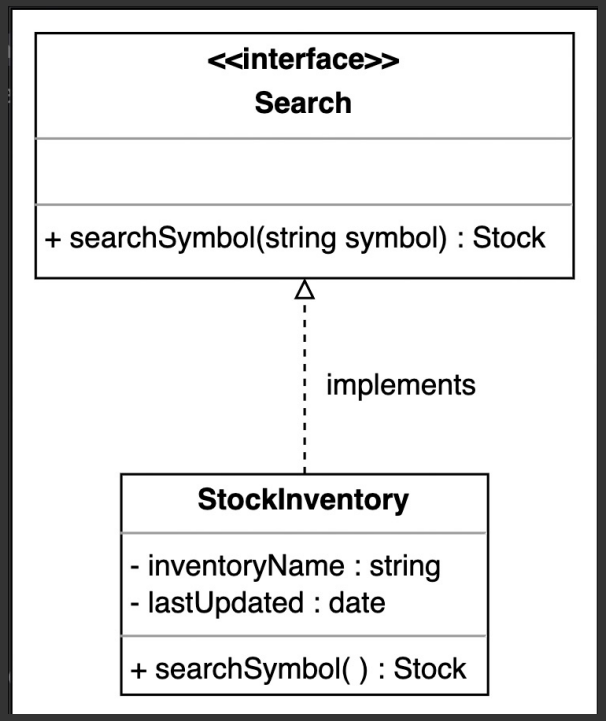


2. Watchlist

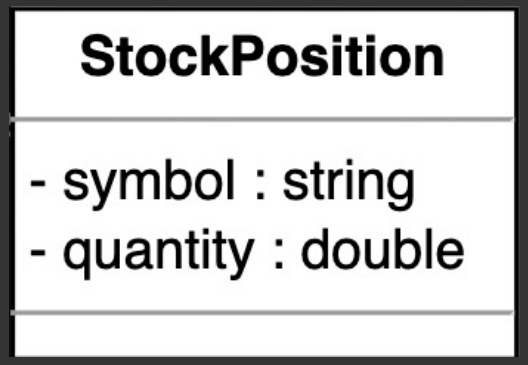


3. Stock

4. Search & Stock Inventory

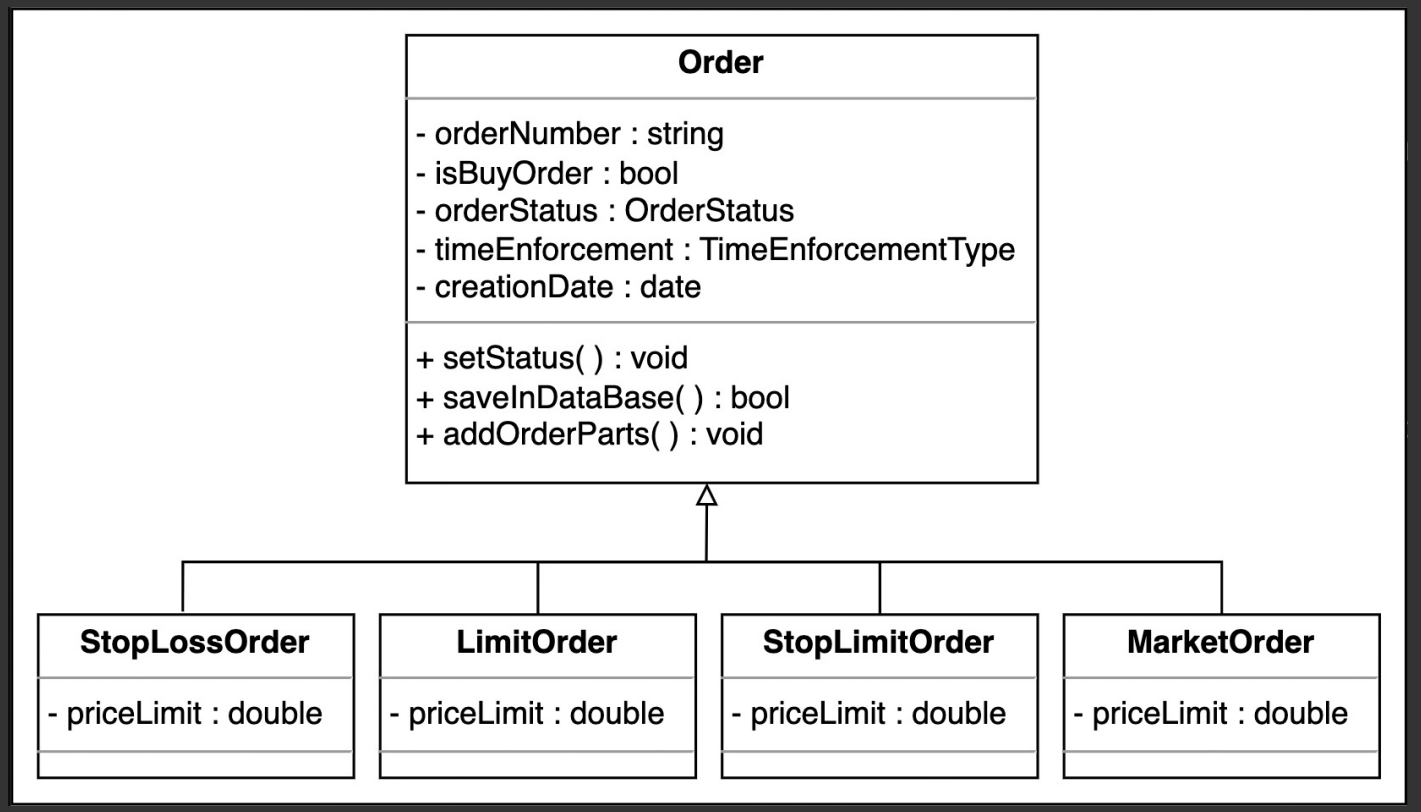


5. Stock Position

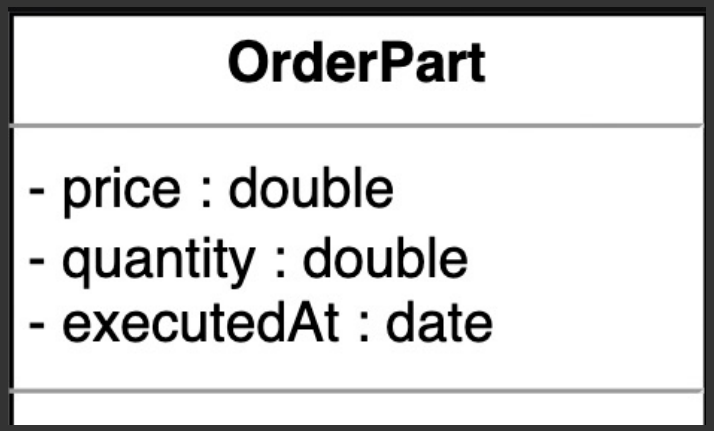


6. Stock Lot

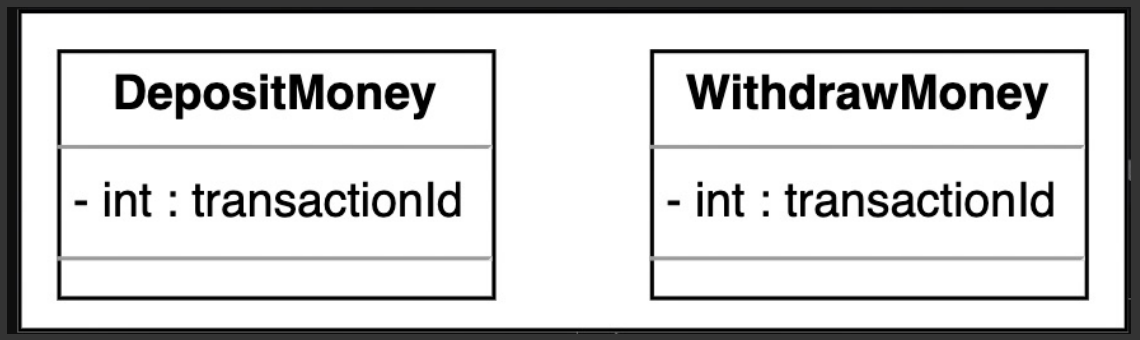
7. Order



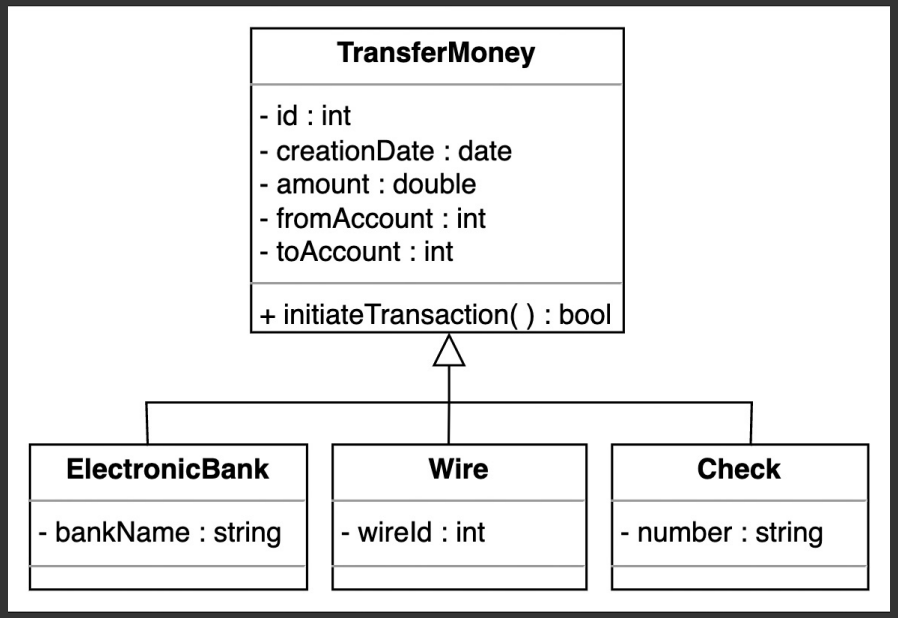
8. Order Part



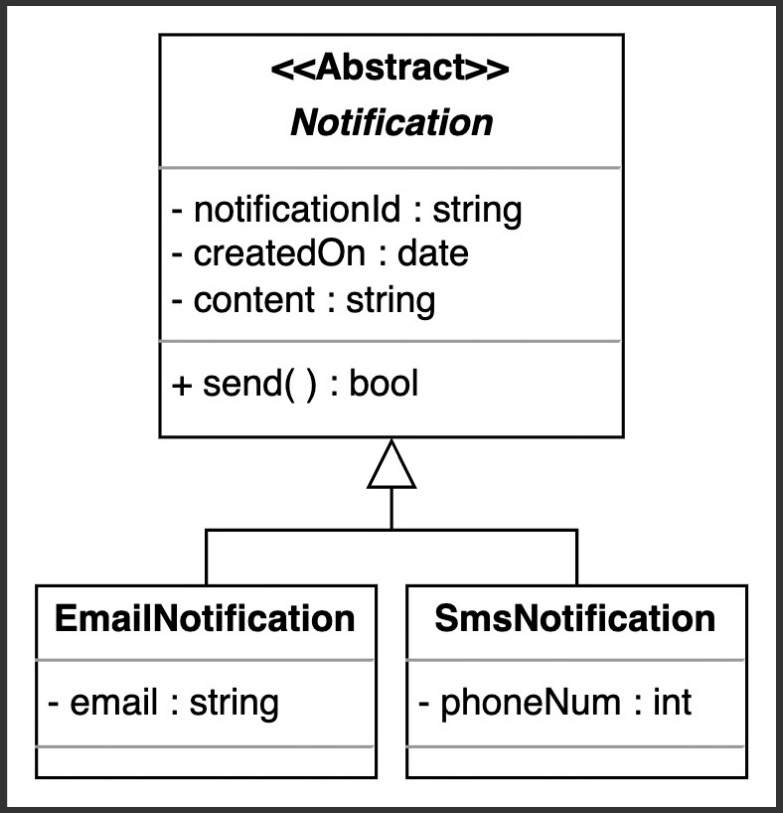
9. Deposit Money & Withdraw Money



10. Transfer Money

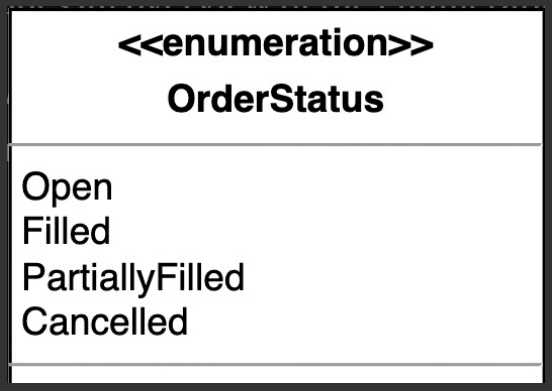


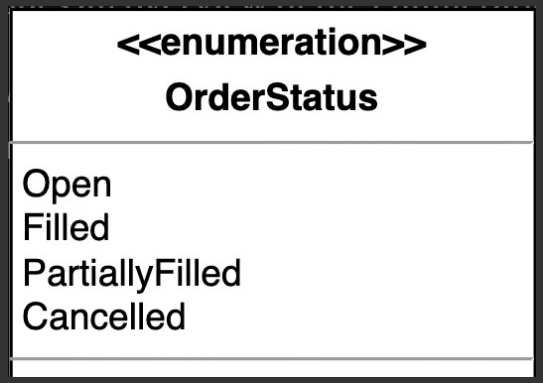
11. Notification Service

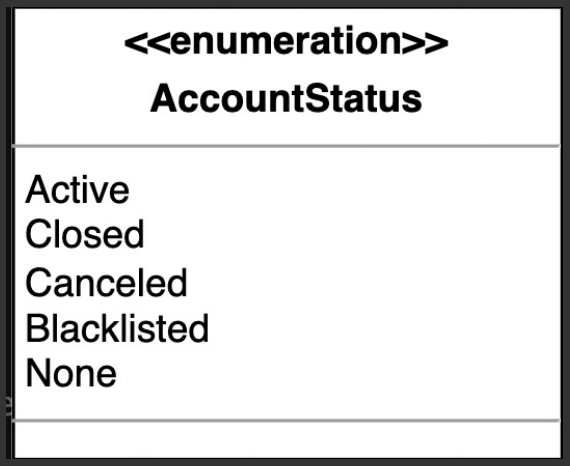


12. Stock Exchanges

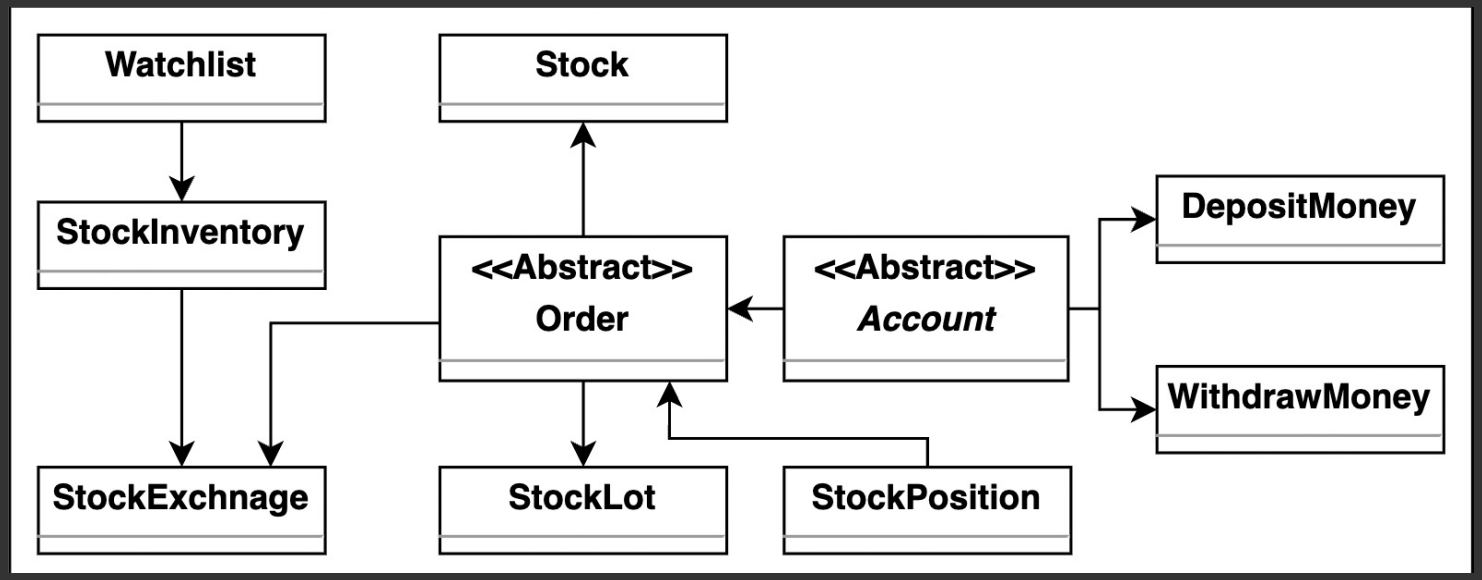
13. Enumeration



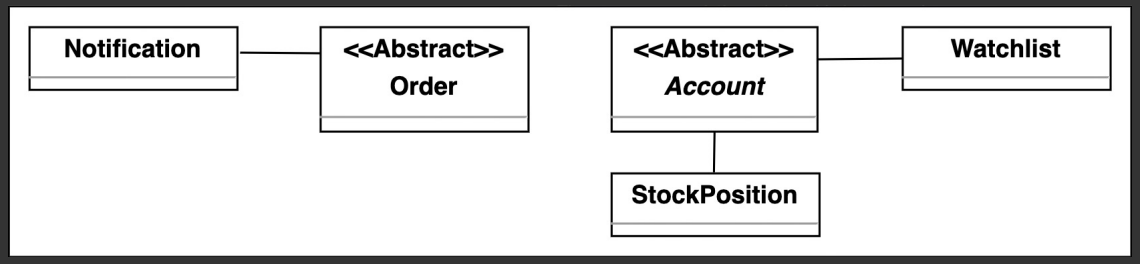




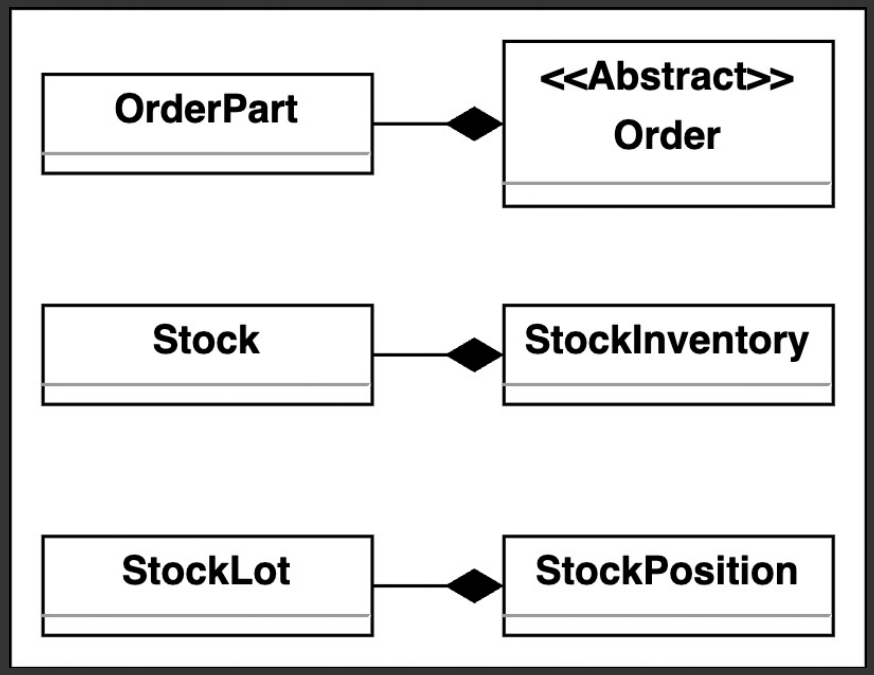
Relationship between classes

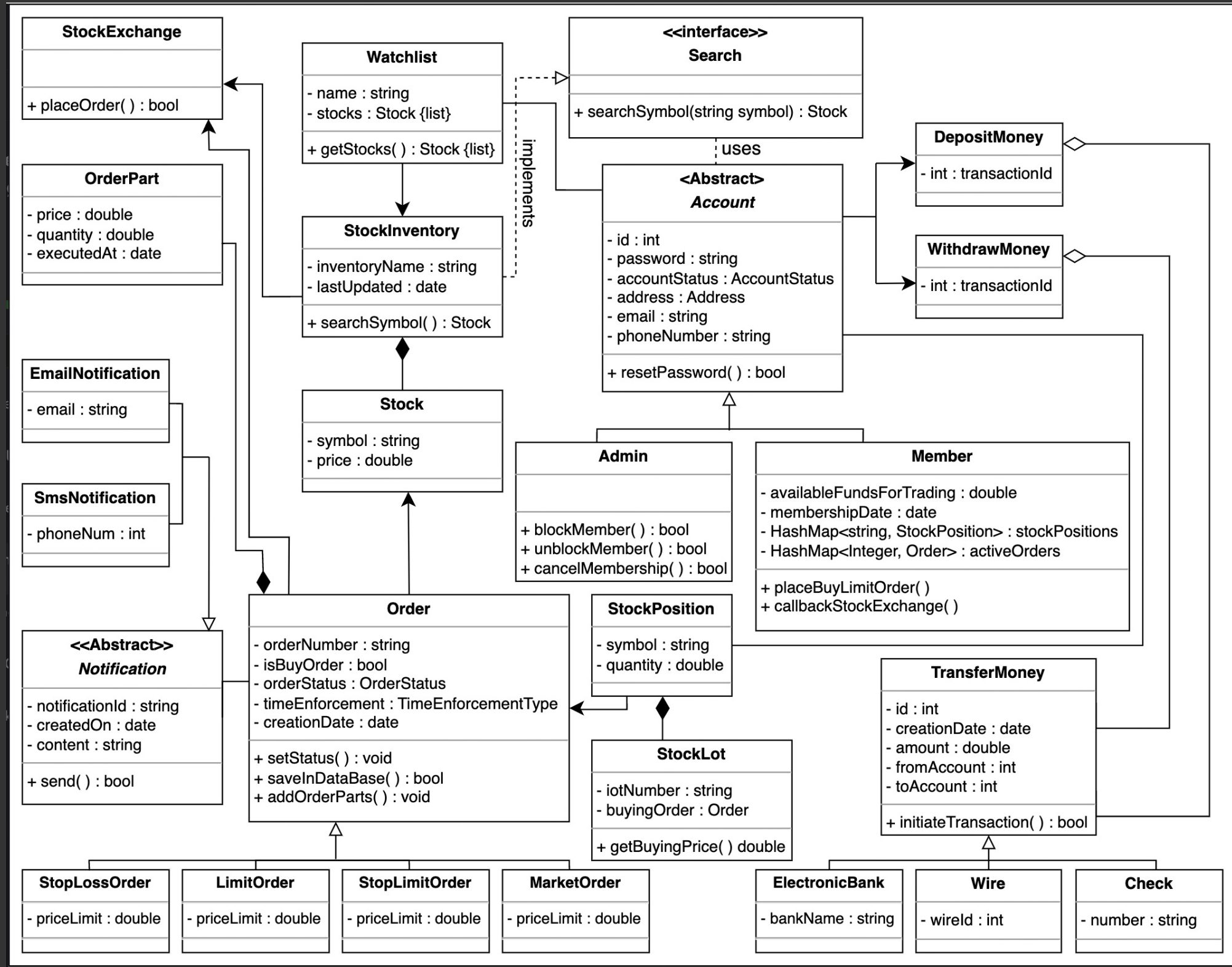


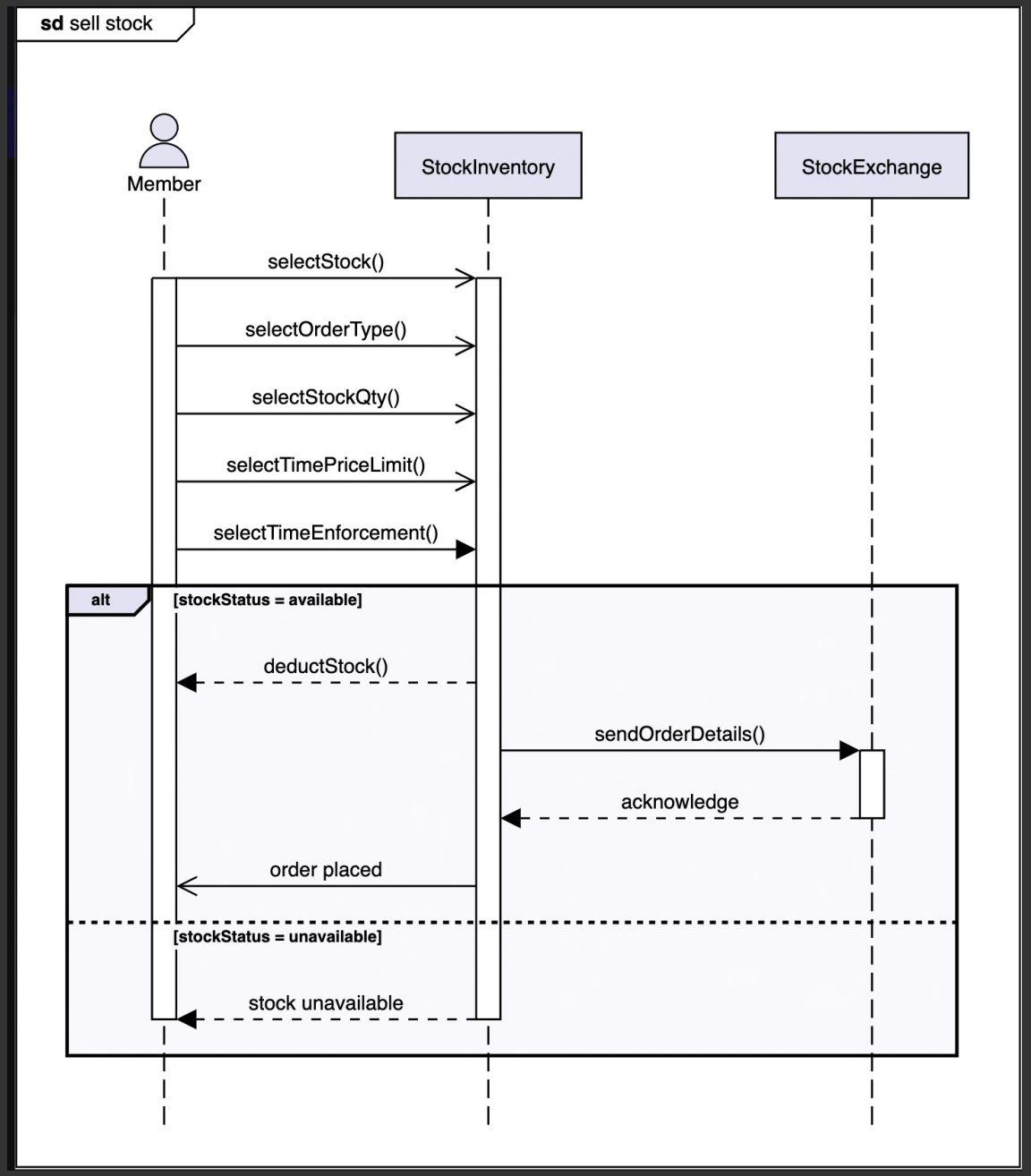
Two way association



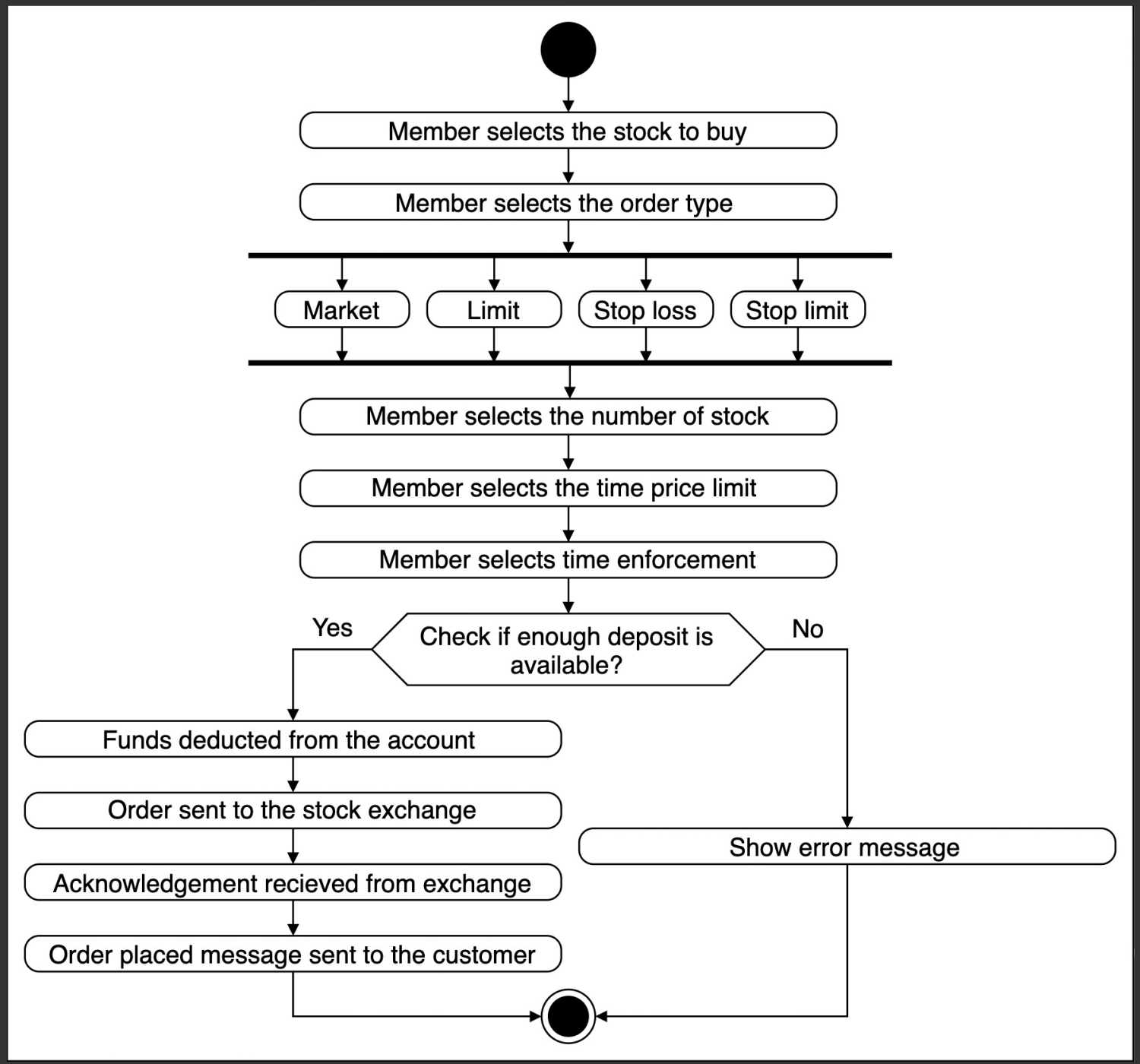
Composition




StockExchange
+ placeOrder( ) : bool
Watchlist
- name : string
- stocks : Stock {list}
+ getStocks( ) : Stock {list}
<<interface>>
Search
+ searchSymbol(string symbol) : Stock
implements
uses
DepositMoney
- int : transactionId
OrderPart
- price : double
- quantity : double
- executedAt : date
StockInventory
- inventoryName : string
- lastUpdated : date
+ searchSymbol( ) : Stock
<Abstract>
Account
- id : int
- password : string
- accountStatus : AccountStatus
- address : Address
- email : string
- phoneNumber : string
+ resetPassword( ) : bool
WithdrawMoney
- int : transactionId
EmailNotification
- email : string
Stock
- symbol : string
- price : double
SmsNotification
- phoneNum : int
Admin
+ blockMember( ) : bool
+ unblockMember( ) : bool
+ cancelMembership( ) : bool
Member
- availableFundsForTrading : double
- membershipDate : date
- HashMap<string, StockPosition> : stockPositions
- HashMap<Integer, Order> : activeOrders
+ placeBuyLimitOrder( )
+ callbackStockExchange( )
<<Abstract>>
Notification
- notificationId : string
- createdOn : date
- content : string
+ send( ) : bool
Order
- orderNumber : string
- isBuyOrder : bool
- orderStatus : OrderStatus
- timeEnforcement : TimeEnforcementType
- creationDate : date
+ setStatus( ) : void
+ saveInDataBase( ) : bool
+ addOrderParts( ) : void
StockPosition
- symbol : string
- quantity : double
StockLot
- iotNumber : string
- buyingOrder : Order
+ getBuyingPrice( ) double
TransferMoney
- id : int
- creationDate : date
- amount : double
- fromAccount : int
- toAccount : int
+ initiateTransaction( ) : bool
StopLossOrder
- priceLimit : double
LimitOrder
- priceLimit : double
StopLimitOrder
- priceLimit : double
MarketOrder
- priceLimit : double
ElectronicBank
- bankName : string
Wire
- wireId : int
Check
- number : string



Activity Diagram to Sell stock



Activity diagram of buying a stock

Code for the Online Stock Brokerage System

1. Enumerations

```
// Enumeration
enum OrderStatus {
  OPEN,
  FILLED,
  PARTIALLY_FILLED,
  CANCELED
}

enum TimeEnforcementType {
  GOOD_TILL_CANCELED,
  FILL_OR_KILL,
  IMMEDIATE_OR_CANCEL,
  ON_THE_OPEN,
  ON_THE_CLOSE
}

enum AccountStatus {
  ACTIVE,
  CLOSED,
  CANCELED,
  BLACKLISTED,
  NONE
}

// Custom Address data type class
public class Address {
  private int zipCode;
  private String address;
  private String city;
  private String state;
  private String country;
}
```

2. Account

```
// Account is an abstract class
public abstract class Account {
  private String id;
  private String password;
  private String name;
  private AccountStatus status;
  private Address address;
  private String email;
  private String phone;

  public abstract boolean resetPassword();
}

public class Member extends Account {
  private double availableFundsForTrading;
  private Date dateOfMembership;
  private HashMap<string, StockPosition> stockPositions;
  private HashMap<Integer, Order> activeOrders;

  public ErrorCode placeSellLimitOrder(string stockId, float quantity,
int limitPrice, TimeEnforcementType enforcementType);
  public ErrorCode placeBuyLimitOrder(string stockId, float quantity,
int limitPrice, TimeEnforcementType enforcementType);
  public void callbackStockExchange(int orderId, List<OrderPart>
orderParts, OrderStatus status);
  public boolean resetPassword(){
    // definition
  }
}

public class Admin extends Account {
  public boolean blockMember();
  public boolean unblockMember();
  public boolean cancelMembership();
  public boolean resetPassword(){
    // definition
  }
}
```

3. Stock & Watch List

```
public class Watchlist {
  private String name;
  private List<Stock> stocks;

  public List<Stock> getStocks();
}

public class Stock {
  private String symbol;
  private double price;
}
```

4. Search & Stock Inventory

```
public interface Search {
    // Interface method (does not have a body)
    public Stock searchSymbol(String symbol);
}

public class StockInventory implements Search {
  private String inventoryName;
  private Date lastUpdate;
  public Stock searchSymbol(String symbol) {
      // definition
  }
}
```

5. Stock Position & Stock Lot

```
public class StockPosition {
  private String symbol;
  private double quantity;
}

public class StockLot {
  private String iotNumber;
  private Order buyingOrder;

  public double
getBuyingPrice();
}
```

6. Order

```
public class OrderPart {
  private double price;
  private double quantity;
  private Date executedAt;
}

// Order is an abstract class
public abstract class Order {
  private String orderNumber;
  public boolean isBuyOrder;
  private OrderStatus status;
  private TimeEnforcementType timeEnforcement;
  private Date creationTime;
  private HashMap<int, OrderPart> parts;

  public void setStatus(OrderStatus status);
  public boolean saveInDatabase();
  public void addOrderParts(OrderParts parts);
}

public class LimitOrder extends Order {
}

public class StopLimitOrder extends Order {
}

public class StopLossOrder extends Order {
}

public class MarketOrder extends Order {
}
```

7. Transfer Money

```
// TransferMoney is an abstract class
public abstract class TransferMoney {
  private int id;
  private Date creationDate;
  public int fromAccount;
  private int toAccount;

  public abstract boolean initiateTransaction();
}

public class ElectronicBank extends TransferMoney {
  private String bankName;

  public boolean initiateTransaction(){
    // definition
  }
}

public class Wire extends TransferMoney {
  private int wire;

  public boolean initiateTransaction(){
    // definition
  }
}

public class Check extends TransferMoney {
  private String checkNumber;

  public boolean initiateTransaction(){
    // definition
  }
}

public class DepositMoney {
  private int transactionId;

  public boolean initiateTransaction(){
    // definition
  }
}

public class WithdrawMoney {
  private int transactionId;
}
```

8. Notification

```
public abstract class Notification {
    private String notificationId;
    private Date creationDate;
    private String content;

    public abstract boolean
sendNotification();
}

public class SmsNotification extends
Notification {
    private int phoneNumber;

    public boolean sendNotification(){
        // definition
    }
}

public class EmailNotification extends
Notification {
    private String email;

    public boolean sendNotification(){
        // definition
    }
}
```

9. Stock Exchange

```
public class StockExchange {
  // The StockExchange is a singleton class that ensures it will have only one active
instance at a time
  private static StockExchange instance = null;

  // Created a private constructor to add a restriction (due to Singleton)
  private StockExchange() {}

  // Created a static method to access the singleton instance of StockExchange
  public static StockExchange getInstance()
  {
    if(instance == null) {
      instance = new StockExchange();
    }
    return instance;
  }

  public boolean placeOrder(Order order);
}
```