ER and UML Assignment

Thought Process

For the user and prospect, thought to create a base class Person and extend to User and Prospect, but the only common attribute is the name. So doesnt make sense to make them siblings.

For user, each user can have multiple accounts, and each account can be of a different type.

Bank has multiple products, but there are very little in common between an account, and a bond. Might as well separate Products from Accounts. Bank sells products, and bonds are one of them. Accounts are separate entities. Assuming other products are debt instruments like credit cards, maybe we could make another

Vishy mentioned anyone can buy a bond without an account. Then it means there are brokers that sell these bonds. Probably. (Assumption #1) One such broker could be the bank itself, for example. So we need a class for Broker, which each broker adheres to. The fields listed in the Broker class may not be complete, but this is the thought process.

Does the bank have info on the broker only, or will they also have info on the buyer who bought from the broker? It would be logistically easier for the bank to only hold details of the broker who bought the bonds, and let the broker handle the end buyer details. (Assumption #2)

Classes

Note: These classes reflect whatever is in the ER Diagram. They have been rewritten here for ease of access.

```
class User:
- userID: String
- passwordHash: String
- name: String
- Address: String
- accounts: List[Account]
- verifiedStatus: Boolean

- func createAccount(account: Account) -> Void
- func deleteAccount(account: Account) -> Void
- func verifyPassword(username: String, passwordHash: string) -> Boolean
```

How do prospects link to the rest of the system? (Assumption #3) is that they are handled separately. If the only operations that can be performed with a database of Prospects is marketting and trying to convert them to customers, then the pipeline is the website that lets anyone sign up for an account. In the case of celebrity prospects maybe, the sign-up may be handled internally, and those are implementation concerns that don't factor into the ER Diagram.

```
class Prospect:
- name: String
- priority: Int
- contact: String
- status: some Enum
```

```
abstract class Account:
- accountNum: Integer
- type: some Enum // this prolly wont be needed if we're subclassing for
every type
- balance: Double
```

Assuming one savings account can only have one debit card.

```
class SavingsAccount extends Account:
- debitCardNumber: String
- pinHash: String

- private func checkConstraints() -> Boolean

- func getBalance() -> Double
- func deposit(amount: Int) -> Void
- func withdraw(amount: Int) -> Void
- func transfer(to: Account, amount: Float) -> Void // Could be boolean,
or an Int if error codes
```

```
class CurrentAccount extends Account:
- idkSomeProperty: Any
- someOtherProperty: Int
```

```
abstract class Product:
- productID: String
- broker: Broker
```

```
class Bond extends Product:
- bondName: String
- additionalBondInfo: String?
```

```
class SomeOtherProduct extends Product:
- productAttributeOne: String
- productAttributeTwo: String?
```

```
class Broker:
- name: String
- brokerID: String
- moarBrokerDetails: String?
- stock: [Product]
```