Problem Decomposition Revisited (Again): Object Oriented Design

Overview

- Object Oriented Design
  - Identify the Classes
  - Identify what Information each Class Needs
  - Identify what each Class Needs to Do

Software Development Life Cycle

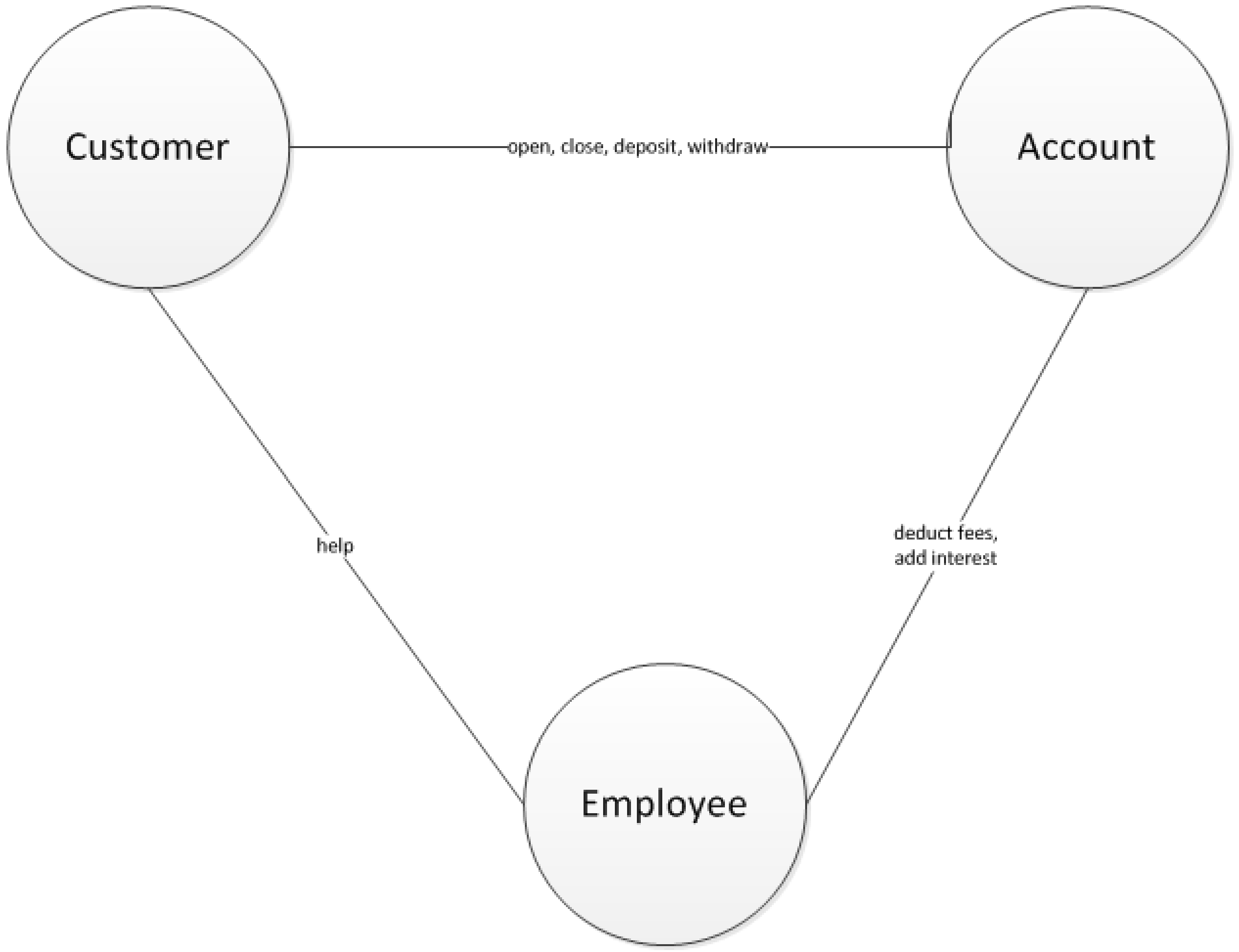
1. Understand the Problem = Requirements Analysis
2. Work out the Logic = Design
3. Convert it to Code = Implementation
4. Test/Debug
5. Maintenance

Today we will talk about requirements analysis and object oriented design.

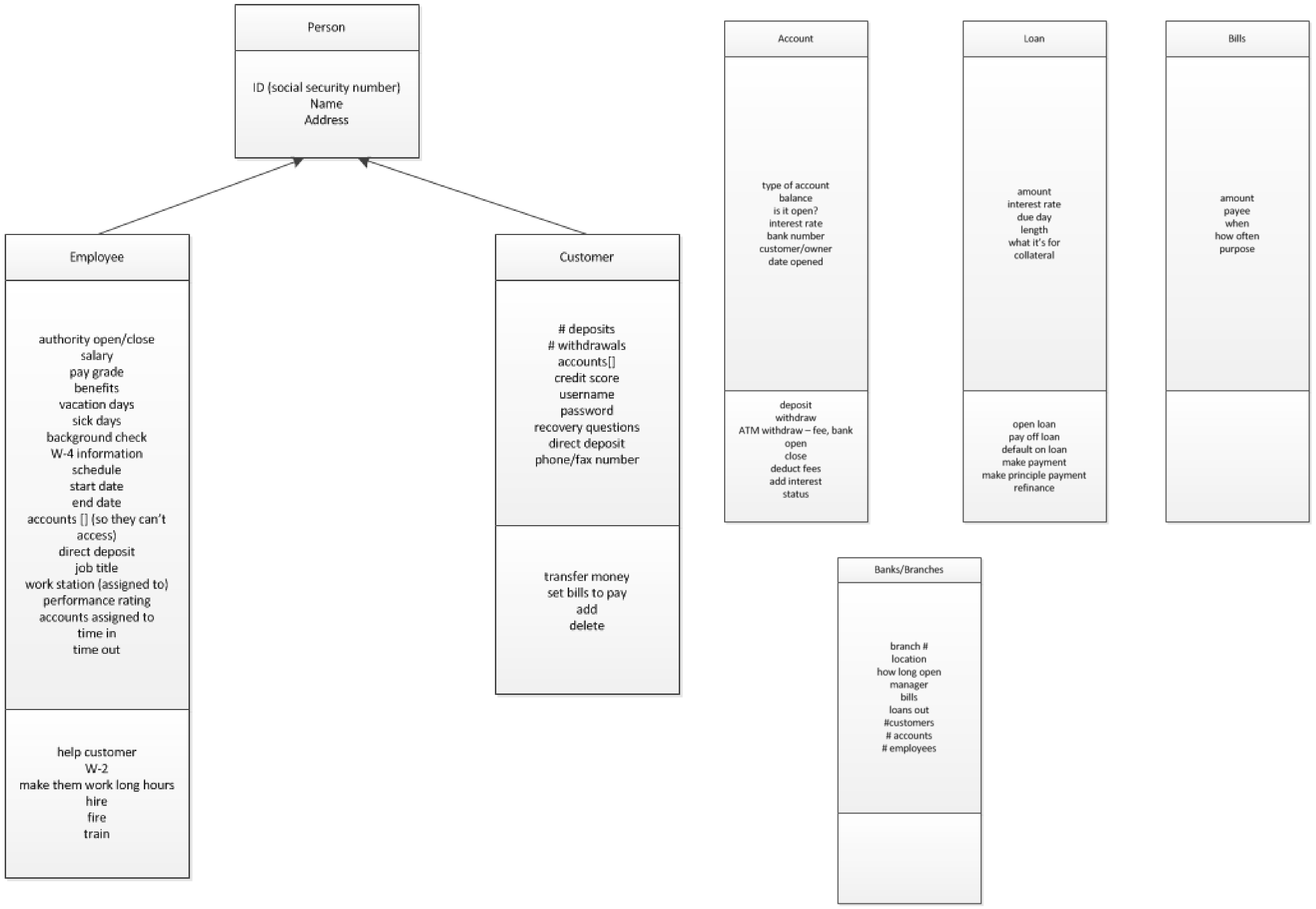
What are the Nouns?

- You have been hired to automate bank operations for a local credit union. They have told you that their business operates as follows:
  - Customers can open accounts. They can make deposits and withdrawals and can close accounts also. On some accounts interest needs to be added, and sometimes fees are deducted.
  - All employees can help customers with deposits and withdrawals. Only some employees are authorized to open and close accounts.

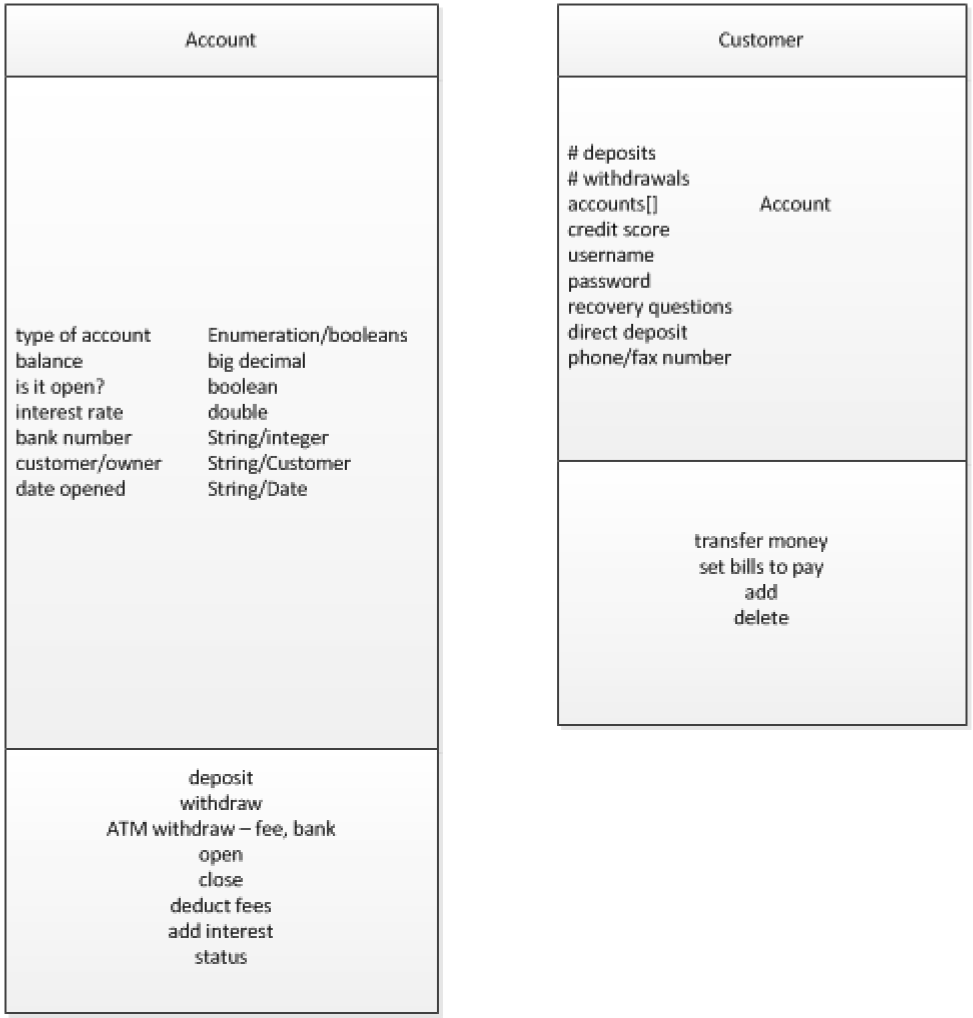
Initial Diagram



UML Diagram


Person
ID (social security number)
Name
Address
Employee
authority open/close
salary
pay grade
benefits
vacation days
sick days
background check
W-4 information
schedule
start date
end date
accounts [] (so they can't access)
direct deposit
job title
work station (assigned to)
performance rating
accounts assigned to
time in
time out
help customer
W-2
make them work long hours
hire
fire
train
Customer
# deposits
# withdrawals
accounts[]
credit score
username
password
recovery questions
direct deposit
phone/fax number
transfer money
set bills to pay
add
delete
Account
type of account
balance
is it open?
interest rate
bank number
customer/owner
date opened
deposit
withdraw
ATM withdraw – fee, bank
open
close
deduct fees
add interest
status
Loan
amount
interest rate
due day
length
what it's for
collateral
open loan
pay off loan
default on loan
make payment
make principle payment
refinance
Bills
amount
payee
when
how often
purpose
Banks/Branches
branch #
location
how long open
manager
bills
loans out
#customers
# accounts
# employees

UML with Some Data Types Added



Simplified Bank

Let’s ignore some of the complexity and assume a bank employee is running our program. The employee can work with Customers and Accounts.

For one scenario, assume a person comes into our bank and wants to open an account. This person is not yet a customer, so the bank employee needs to add them as a customer and then open the account for them, and make that first deposit into the account.

(By the way, this way of thinking about a problem, by looking at scenarios, is called developing use cases.)

Our job is to first define the API.

Customer:
- Instance Variables:
  - Name
  - Address
  - SSN
  - Accounts
- Methods:
  - Add Customer
  - Delete Customer

Account:
- Instance Variables:
  - Balance
  - Account Number
  - Customer
- Methods:
  - Open Account
  - Close Account
  - Deposit
  - Withdraw
  - Transfer Money

Simplified Bank

Our job is to first define the API.

What will our methods need in order to run, and what will they return to the client program?

Customer – Add Customer
Delete Customer

Account – Open Account
Close Account
Deposit
Withdraw
Transfer Money

Customer:
- Instance Variables:
  - Name
  - Address
  - SSN
  - Accounts
- Methods:
  - Add Customer
  - Delete Customer

Account:
- Instance Variables:
  - Balance
  - Account Number
  - Customer
- Methods:
  - Open Account
  - Close Account
  - Deposit
  - Withdraw
  - Transfer Money

API

Customer

```
                    Customer(String firstName, String lastName,
                             String SSN, String street, String city,
                             String state, String zipCode)
  Customer          DeleteCustomer()
```

Account

```
                    Account(Customer customer, long acctNumber)
                    Account(Customer customer, long acctNumber,
                            double initAmt)
  Account           DeleteAccount()
                    Deposit(double amount)
                    Withdraw(double amount)
                    TransferMoney(double amount, Account account)
```

Instance Variables

Now that the API is defined, we need to make sure our instance variables are adequate to support the API.

1. **What are the data types of each?**
2. **Do we need to refine any of them further?**

Customer:
- Name
- Address
- SSN
- Accounts

Account:
- Balance
- Account Number
- Customer

Instance Variables

Customer:

- String firstName
- String lastName
- String SSN
- String street
- String city
- String state
- String zipCode
- Account [] accounts

Account:

- double Balance
- long accountNumber
- Customer customer

Simplified Bank

Once we are happy with our class definitions, let’s write some code!!

Summary

- Object Oriented Design
  - Identify the classes
  - Identify what information each class needs
  - Identify what each class needs to do
  - Identify use cases
  - Define the API
  - Define the instance variables
  - Finally – write some code!