Problem Decomposition Revisited (Again):
Object Oriented Design

There’s more...?
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**.
## UML with Some Data Types Added

<table>
<thead>
<tr>
<th>Account</th>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>type of account</td>
<td># deposits</td>
</tr>
<tr>
<td>balance</td>
<td># withdrawals</td>
</tr>
<tr>
<td>is it open?</td>
<td>accounts[]</td>
</tr>
<tr>
<td>interest rate</td>
<td>credit score</td>
</tr>
<tr>
<td>bank number</td>
<td>username</td>
</tr>
<tr>
<td>customer/owner</td>
<td>password</td>
</tr>
<tr>
<td>date opened</td>
<td>recovery questions</td>
</tr>
<tr>
<td></td>
<td>direct deposit</td>
</tr>
<tr>
<td></td>
<td>phone/fax number</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>deposit</td>
<td>transfer money</td>
</tr>
<tr>
<td>withdraw</td>
<td>set bills to pay</td>
</tr>
<tr>
<td>ATM withdraw—fee,</td>
<td>add</td>
</tr>
<tr>
<td>bank open</td>
<td>delete</td>
</tr>
<tr>
<td>open</td>
<td></td>
</tr>
<tr>
<td>close</td>
<td></td>
</tr>
<tr>
<td>deduct fees</td>
<td></td>
</tr>
<tr>
<td>add interest</td>
<td></td>
</tr>
<tr>
<td>status</td>
<td></td>
</tr>
</tbody>
</table>
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.
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!