Object Oriented Analysis & Design
面向对象分析与设计

Lecture_04 面向对象分析(二)

主讲: 姜宁康 博士

4、CRC方法建模案例：ATM取款机软件

- 构建领域模型的完整过程

4.1 ATM取款机软件-需求描述

- **University Bank will be opening in January, 2020. We plan to use a full service automated teller machine (ATM) system**
  - The ATM system will interact with the customer through a display screen, numeric and special input keys, a bankcard reader, a deposit slot, and a receipt printer
  - Customers may make deposits, withdrawals, and balance inquires using the ATM machine, but the update to accounts will be handled through an interface to the Accounts system
  - Customers will be assigned a Personal Identification Number (PIN) and clearance level by the Security system. The PIN can be verified prior to any transaction
  - In the future, we would also like to support routine operations such as a change of address or phone number using the ATM

4.2 建议的概念类

- 通过头脑风暴，列出如下的一群概念 By brainstorming, we might identify the following candidate classes for University Bank's ATM system

ATM	Financial Transaction	BankCard
Bank Customer	PIN	Account
Savings Account	Checking Account	Transfer
Withdrawal	Deposit	Balance Inquiry
Receipt	Receipt Printer	Keypad
Screen	Cash Dispenser	ScreenMessage
Display	Funds Available	DepositEnvelopeFailure
Balance	TimeOutKey	TransactionLog
Key	AccountHolder	Printer
ScreenSaver	Prompt	NumericKey

4.3 标识核心概念类 Identifying Core Classes

- **Moving from brainstorming to analysis, divide the candidate classes into categories in the following order:**
- **关键的critical classes (i.e., the "winners"), for which we will definitely write CRC cards Items that directly relate to the main entities of the application**
  - **In the ATM example:**
    - **Account, Deposit, Withdrawal, BalanceInquiry**
- **无关的 irrelevant candidates (i.e., the "losers"), which we will definitely eliminate Items that are clearly outside the system scope**
  - **In the ATM example:**
  - **Printer, ScreenSave, and Prompt -- related to the user interface subsystem to be done later, but not to the core banking application**
- **待定的 undecided candidates (i.e., the "maybes"), which we will review further for categorization**
  - **Items that we may not be able to categorize without first clarifying the system boundaries and definition**
  - **In the ATM example: What is an "ATM"?**

4.4 明确系统范围 Clarifying System Scope

- To continue design, we must determine
  - what is and what is not part of the system
- Scope of the ATM system
  - Does it handle everything--the banking application, the user interface, and the interactions between them?
  - Is the ATM responsible for updating accounting records?
  - or just recording and mediating促成 the transaction activity?
- It is often useful to draw a diagram to record system boundaries
  - ATM example: limit its scope to the banking information capture
    - -- leaving the user interface and actual account update to other systems

4.5 去掉不必要的核心类 Eliminate Unnecessary Core Classes

- 移除“幽灵”类 Remove ghost classes
  - -- 即，进一步检查，发现这个类不适合应用系统
  - 检查那些与系统相关、但是在系统外部的类；系统需要有接口与它们通信，但不需要在系统内部为它们建模
  - ATM 例子
    - BankCustomer, Printer, and Keypad are relevant but outside the application being developed
    - For example, the system only needs to know about BankCustomer indirectly through the BankCard information such as the PIN

4.5 去掉不必要的核心类 Eliminate Unnecessary Core Classes

- 合并同义词
  - 一个组织内的不同小组，为同一个概念起了不同的名字
  - ATM example:
    - BankCustomer and AccountHolder are probably synonyms. Adopt one of them or create a new name
- 特别注意：同一个名称，在不同的地方表示不同的含义 Be careful when the same word actually refers to different things
  - 这种情况下，需要增加一个新的核心类

4.5 去掉不必要的核心类 Eliminate Unnecessary Core Classes

- 辨析一个概念是属性还是类
  - 符合以下两点的类，一般是属性
  - 它不做具体的事情 it does not do anything -- it has no operations
    - ATM example:
    - Balance and FundsAvailable have few meaningful operations and both are closely associated with Account
  - 它不能改变状态 it cannot change state
    - ATM example: Consider a PIN
    - If a PIN is viewed as being immutable(不变的), not changing state, then it probably should be an attribute of Account However,
    - if a PIN can change state -- say from among valid, invalid, and suspended -- then it should be a class

4.6 ATM新的核心类

Core Classes

FinancialTransaction

Account

BalanceInquiry

Withdrawal

Deposit

AuthorizeSystemInteraction

BankCard

Undecided Classes

BankCustomer (ghost - integrated withAuthorizeSystemInteraction)

PIN (attribute)

SavingsAccount (attribute of Account)

CheckingAccount (attribute of Account)

ATM (ghost -- system name)

FundsAvailable (attribute)

Balance (attribute)

AccountHolder (synonym)

4.7 ATM无关的类

Irrelevant Items

Transfer (not handled in first version)
Receipt
ReceiptPrinter
Keypad
Screen
CashDispenser
ScreenMessage
Display
DepositEnvelopeFailure
TimeOutKey
TransactionLog
Printer
ScreenSaver
Prompt
Numeric Key
Key

这些都已经超出了系统的范围，它们中的一部分可能在“用户接口”子系统中作为概念类

4.8 为核心类分配职责

- 为每个核心类准备一个CRC卡，写上职责 Write responsibilities on CRC cards for each core class
  - responsibilities for doing operation
  - responsibilities for knowing
- 为概念类定义协作者，此过程与定义职责的过程是交错进行的Task of finding collaborators for classes often intermixed with finding responsibilities
  - 可以交叉使用“头脑风暴”和“角色扮演”Use a combination of brainstorming and role-playing of scenarios (see later) to discover responsibilities
- 集中在 what, 而不是 how
  - Grady Booch 说过:
  - 当人们在考虑类和对象的语义是，总是倾向于解释事件是如何发生的。但合适的答案往往是“我不关心”"When considering the semantics of classes and objects, there is the tendency to explain how things work; the proper response is 'I don't care' "

4.8 为核心类分配职责

- 定义职责的方法
  - 首先，以“头脑风暴”为核心类列出各种可能的职责
  - 然后再精化
  - 如果大多数职责都归属于少数几个类，则设计出了点问题
    - 没有充分利用多态、封装等特性
    - 很多类的功能退化为“记录信息”，只知道知己的信息

4.8 为核心类分配职责

- ATM example: class Account
  - A tendency might be to give most of the responsibility to the Account class, which becomes a strong, busy "manager" procedure giving commands to relatively weak, ill-defined "worker" classes
    - Account is probably too inflexible to be directly reused; the other classes are probably too insignificant to be reused
  - Give each class a distinct role in the system. Strive to make each class a well-defined, complete, cohesive abstraction
    - Such a class has higher probability of being reused

4.8 为核心类分配职责

- ATM example: class Withdrawal
  - the responsibility "Withdraw Funds", making it potentially useful to any other class needing to do a withdrawal
  - class Account
    - the responsibility "Accept Withdrawal"
      - be carried out by collaborating with Withdrawal
  - Factoring out complexity 分解复杂性 also involves identifying specialized behaviors that occurs repeatedly and, as appropriate, spinning off 引出new classes
    - ATM example: The capturing and responding to user requests might be factored out into a new class
      - Form ， with a responsibility "ask user for information"

4.8 为核心类分配职责

- 使用抽象 Use abstraction
  - Build hierarchies of classes. Abstract the essence of related classes by identifying
    - where they have common responsibilities
    - where they do the same thing, but do it differently
      - -- same "what", different "how"
  - That is, look for opportunities for the classes to use polymorphism to implement the same responsibility differently
  - The new parent classes may be *abstract classes*. The abstract class exists to link together similar concrete types of objects
    - ATM example:
    - Create an abstract class Transaction that is a superclass for Withdrawal, Deposit, etc.
    - It can have an abstract responsibility "execute a financial transaction", that is implemented differently for each subclass

4.8 为核心类分配职责

- 不要满足一个方案，考虑几个备选方案
  - Remember that CRC cards are inexpensive and erasable !!
  - Do not hesitate to experiment with different configurations of classes or assignments of responsibilities
  - Changing the CRC cards is easy in the early stages of a project changing the code later in the project is not easy

4.9 分配协作 Assigning Collaborations

- 标识类之间的关系 Identify relationships among classes
  - Each class is specialist in some set of knowledge and behaviors
  - Classes must cooperate to accomplish nontrivial tasks
  - Thus collaborations between classes are important
- 使用“基于场景”的角色扮演，发现/测试协作 Use scenario-based role-play to find and/or test these collaborations
  - Scenario is a system behavior and sequence of system events to realize it
  - Simulating execution enables team to discover responsibilities and collaborations and/or check correctness of those already found

4.9 分配协作 Assigning Collaborations

- 列出需要演示的场景: "must do" first, then "can do if"
- 为参与者分配角色 Assign roles to actors
  - distribute CRC cards
  - collaborators to different persons
  - use domain experts for classes if possible
  - rotate assignments from session to session
- 选定初始的场景 Initiate scenario
  - hold up first class
  - ATM example: Start with Customer or BankCard
- 在表演过程中发现问题 Watch the action to detect problems
- 一旦出现错误 When errors discovered
  - 小错即改、中错记录、大错停止演示 correct if minor， take notes if complex，stop enactment immediately if significant and complex -- go to assessment
- 避免不必要的变化 Avoid unnecessary changes
  - Especially after considerable role play, avoid creating more problems

4.9 分配协作 Assigning Collaborations

- **Add collaborations when relationships found**
  - **Identify clients and servers**
    - **Server -- class that provides a resource**
    - **Client -- class that uses a resource**
    - **Server is collaborator of client, not vice versa**
  - **ATM example:**
    - **In a withdrawal operation, Account is a client of the Withdrawal class**
  - **Identify hierarchies of classes**
    - **ATM example:**
      - **Transaction as superclass of Withdrawal, Deposit, etc.**

小节：用CRC创建模型需要下面的步骤

- 1）建立团队：包括客户、设计人员、分析人员和一个导引者，如果没有那么多人，那么可以是客户和你自己
- 2）找出需求中存在的名词和名词词组，特别注意复数（通常是集合），它们对应的单数才是
  - 把你第一次想到的所有概念都写在白板或纸上，不管看起来这些概念是如何荒谬，把他们都写下来
- 3）筛选
  - 把对象分为三类，核心对象（必须首先实现），可选的（目前不能确定），以及不需要的对象，这之前最好确定一下你的项目范围，某些不属于本项目范围的对象可以先放一边
- 4）建CRC卡
  - 把核心类写在每一张卡上，把可选的类和排除的类分别写在不同的纸上
- 5）角色扮演
  - 每个人可以扮演多个类，用例中重要的场景都要演示

- 本讲结束