Object-Oriented Analysis and Design Methodology

Romi Satria Wahono
Email: romi@romisatriawahono.net
HP: http://romisatriawahono.net
Department of Information and Computer Sciences
Graduate School of Science and Engineering
Saitama University
Contents

- An Introduction to the Object-Orientation
- An Introduction to the Object-Oriented Methodology
- Object-Oriented Notation Guide
- Object-Oriented Analysis and Design
- Object-Oriented Implementation
An Introduction to the Object-Orientation
What is Object-Orientation

- A new technology based on objects and classes
- A way of thinking to organizing software as a collection of discrete objects that incorporate both data structure and behaviour
- An abstraction of the real world based on objects and their interactions with other objects
Three Characteristics of OO

- **Abstraction and Classification**:
  - Focusing on essential, inherent aspects of an entity and ignoring its accidental.
  - The idea of grouping software ideas into classes of things.

- **Encapsulation and Information Hiding**:
  - Separating the external aspects of an object, which are accessible to other objects, from the internal implementation details of object, which are hidden from other objects.

- **Polymorphism and Inheritance**:
  - Ability of abstractions to share properties by inheritance hierarchy.

Abstraction, Encapsulation, Polymorphism, Inheritance.
Object and Classes

- **Object**
  - An object is a thing or concept. It can be a real-world thing or concept, or an abstraction of a thing or concept expressed as a software representation.
  - An object has state (attributes) and behavior (method)
  - Individual objects, also called instances, have identity and are distinct things, and can be distinguished from other objects.

- **Classes**
  - A class is a description of a collection of objects with common attributes and behavior.
  - In practice, the definition or specification of a class includes the definitions of the attributes comprising the state, the methods implementing the behavior, and how to handle creation and destruction of an object.
An Introduction to the Object-Oriented Methodology
What Are Analysis and Design For

- Testing a physical entity before building system
- Communicating with Customers
- Visualization
- Reduction of Complexity
Various Type of Methodologies

- Shlaer/Mellor Method [Shlaer-1988]
- Coad/Yourdon Method [Coad-1991]
- Booch Method [Booch-1991]
- OMT Method [Rumbaugh-1991]
- Wirfs-Brock Method [Wirfs-Brock-1990]
- OOSE Objectory Method [Jacobson-1992]
- UML (Unified Modeling Language) [UML-1997]
Development Process

Object-Oriented Analysis

Object-Oriented Design

Object-Oriented Implementation
Object-Oriented Notation Guide
Class and Object

- Class
  - Class Name
  - Attribute
  - Operation

- Object Instances
  - Class Name
  - Attribute
  - Operation

- Instantiation Relationship
  - Class Name
  - Class Name
Aggregation

- Aggregation 1
  - Assembly Class
    - Part 1 Class
    - Part 2 Class

- Aggregation 2
  - Assembly Class
    - Part 1 Class
    - Part 2 Class
Association

- Association

Class 1  Association Name  Class 2
Role 1  Role 2

- Qualified Association

Class 1  qualifier  Association Name  Class 2
Role 1  Role 2

- Multiplicity of Associations

Class 1  Class 2
Exactly One  Many  One or More

1+
Object-Oriented Analysis and Design
Analysis and Design Process

- Problem Statement
- System Architecture
- Object Modeling
  - Identifying Object Classes
  - Preparing a Data Dictionary for Classes
  - Identifying Associations
  - Identifying Attributes
  - Refining with Inheritance
  - Grouping Classes into Modules
- Dynamic Modeling
- Functional Modeling
Problem Statement

- Requirements Statement
  - Problem Scope
  - What is needed
  - Application Context
  - Assumptions
  - Performance Needs
Example: ATM Network
Identifying Object Classes

- Extract nouns
- Eliminate spurious classes

Requirements
Statement
- Tentative Object Classes
- Object Classes

Discard Unnecessary and Incorrect Classes

- Redundant classes
- Irrelevant classes
- Vague classes
- Attributes
- Operations
- Roles
- Implementation constructs
Example: IOC for ATM Network

Bad Classes

- System
- Banking Network
- Security Provision
- Record Keeping Provision
- Receipt
- Transaction Data
- Account Data
- Access
- Transaction Log
- Software
- Comm Line

User

Cost

Redundant

Irrelevant

Good Classes

- Account
- ATM
- Bank
- Consortium
- Customer
- Cashier
- Cashier Station
- Central Computer
- Bank Computer
- Cash Card
- Transaction

Vague

Attribute

Implementation
Preparing a Data Dictionary

- Isolated word have many interpretations, so prepare a data dictionary for all modeling entities.
- Describe the scope of the class within the current problem, including assumptions or restrictions on its membership or use.
- The data dictionary also describes associations, attributes, and operation.
Example: DD for ATM Network

- **Account**: a single account in a bank against which transactions can be applied. Account may be of various types, at least checking or savings. A customer can hold more than one account.
- **Bank**: A financial institution that holds accounts for customers and that issues cash cards authorizing access to accounts over the ATM network.
- **ATM**: ...
- **Bank Computer**: ...
- **Cash Card**: ...
- **Cashier**: ...
- etc.
Identifying Associations

1. Extract verbs
2. Tentative associations
3. Eliminate spurious associations
4. Associations

Discard
- Associations between eliminated classes
- Irrelevant or implementation associations
- Actions
- Ternary associations
- Derived associations
- Misnamed associations
- Multiplicity

Object Classes

Discard Unnecessary and Incorrect Associations
Identifying Attributes

Extract object properties → Tentative attributes → Eliminate spurious attributes

Object
Classes

Discard Unnecessary and Incorrect Attributes

- Objects
- Qualifiers
- Names
- Identifiers
- Link attributes
- Internal values
- Fine detail
- Discordant attributes
Example: IAT for ATM Network
This step is to organize classes by using inheritance to share common structure.

Inheritance can be added in two directions:

- **Bottom Up**: By generalizing common aspect of existing classes into a superclasses.
  - By searching for classes with similar attributes, associations, or operations.
  - For each generalization, define a superclass to share **common features**.

- **Top Down**: By refining existing classes into specialized subclasses.
Example: RWI for ATM Network

- ATM
  - cash on hand dispensed
  - Communicates with
    - Station Code
    - Bank Computer
- Central Computer
  - Bank Code
  - Owns
- Consortium
  - Bank Code
  - consists of
- Bank
  - name
  - Account Code
  - Employee Code
  - Card Code
- Cashier Station
  - Station Code
  - Communicates with
    - Bank Computer
- Cashier
  - name
  - Authorized by
  - Cash Card
    - password
  - Employs
    - Customer
      - name
      - address
    - Accesses
      - Account
        - Balance
        - credit limit
        - type
- Transaction
  - Kind
date-time
  - amount
  - Entered by
    - Cashier
  - Entered on
  - Concerns
  - Owns
  - Owns
  - Owns
  - Consists of
A module is a set of classes that captures some logical subset of entire model.

For example: a model of computer operating system might contain modules for process control, device control, file maintenance, and memory management.
Example: GCIM for ATM Network

- Tellers: Cashier, Entry Station, Cashier Station, ATM
- Banks: Consortium, Bank
The dynamic model shows the time-dependent behavior of the system and the objects in it.

Begin dynamic analysis by looking for event, externally visible stimuli and responses.

The dynamic model is important for interactive systems, but insignificant for purely static data repository, such as database.
The following steps are performed in constructing a dynamic model:

- Prepare scenarios of typical interaction sequences
- Identify events between objects
- Prepare an event trace for each scenario
- Build a state diagram
- Match events between objects to verify consistency
Example: DM for ATM Network

User
- Insert card
- request password
- enter password
- request kind
- enter kind
- request amount
- enter amount
- dispense cash
- request take cash
- take cash
- take card
- display main screen

ATM
- verify account
- account OK
- process transaction
- transaction succeed

Consortium
- verify card with bank
- account OK
- bank account OK
- process bank transaction
- bank transaction succeed

Bank
The functional model shows how values are computed, without regard for sequencing, decisions, or object structure.

The functional model shows which values depend on which other values and the functions that relate them.

Data flow diagrams are useful for showing functional dependencies.
Example: FM for ATM Network

Cash Card

Account

read inputs
perform transaction
generate outputs

User

bank code, card code
balance
password, transaction kind, amount, account type
Messages, cash, receipt

password, transaction kind, amount, account type

User

Account

read inputs
perform transaction
generate outputs

Cash Card

bank code, card code
balance
password, transaction kind, amount, account type
Messages, cash, receipt
Object-Oriented Implementation
Implementation Process

- Class Definition
- Creating Objects
- Calling Operations
- Using Inheritance
- Implementing Association
References


References -3-