Objects and Classes

Part 1 – Concepts of OOP

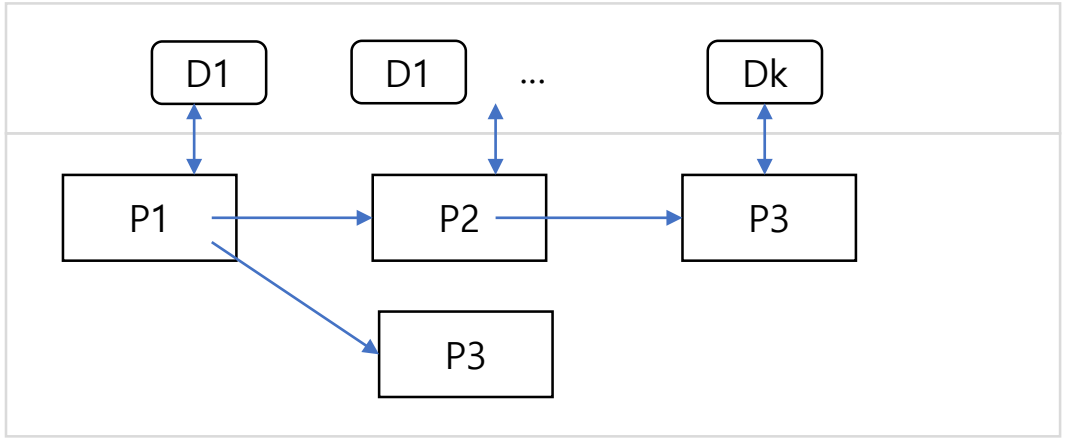
Chapter 4, Core Java, Volume I

Contents

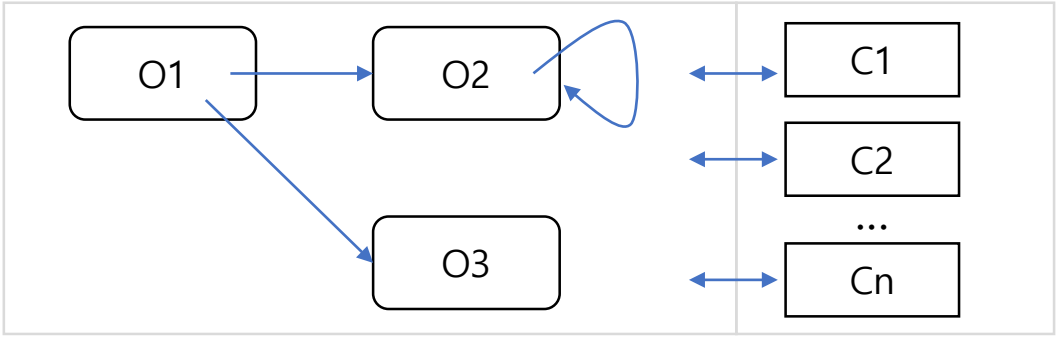
- Why OOP
- Classes and Objects
- Identifying Classes and Methods
- Relationship between Classes

Why OOP?

- 1970s: **Structured** Programming
  - Program = Algorithms + Data Structures (Niklaus Wirth, 1975)
  - A program = a set of procedures
  - Procedures operate on shared data.
  - A program excutes by calling sequence of procedure calls.

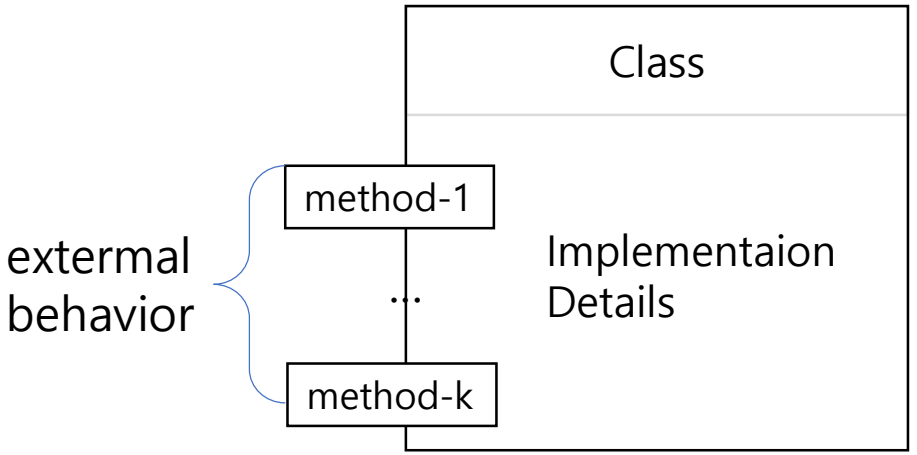


- 1980s: **Object-Oriented** Programming
  - An object = Data + Methods
  - A program = a set of classes
  - Program execution is viewed as a collaboration of objects

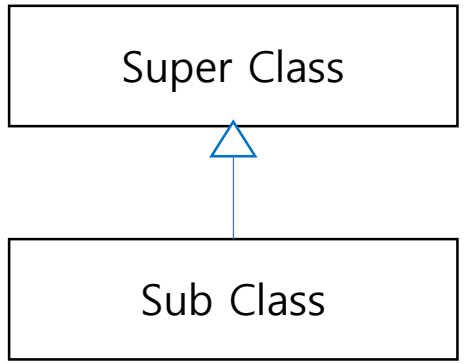


Why OOP?

- Encapsulation (information hiding)
  - Hiding implementation details
  - Exposing external behavior (public methods)
  - Classes are the unit of encapsulation

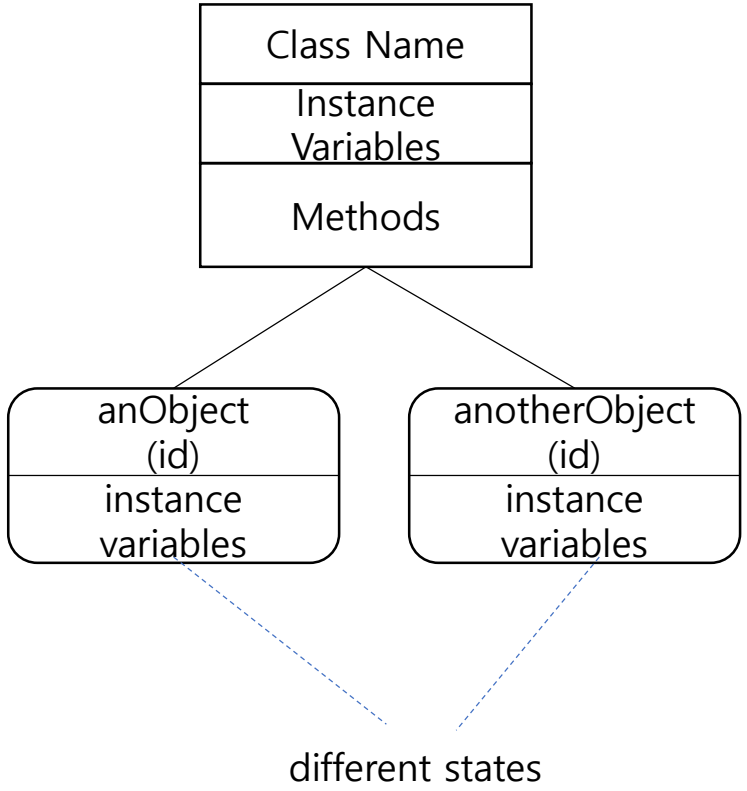


- Inheritance
  - Reusing existing classes
  - Superclass-subclass relationship



Classes and Objects

- Class
  - The template or blueprint from which objects are created. (Think about a cookie cutter)
  - Describes object data (instance variables or fields) and behavior (methods or functions).
- Object = instance of a class (class is a type).
  - Methods – defined in its class
  - Its own copy of instance variables
  - Identity
- Message Sending (or Method Call)
  - An object can call a method in another object.



Identifying Classes and Methods

- How to start object-oriented programming
  - identify classes
  - identify attributes(data) of the class
  - identify methods of the class (assignment of responsibility)
- Identifying classes and methods (a rule of thumb)
  - Nouns are often classes
  - Verbs are often methods
- Example : order-processing system

  Items are added to orders

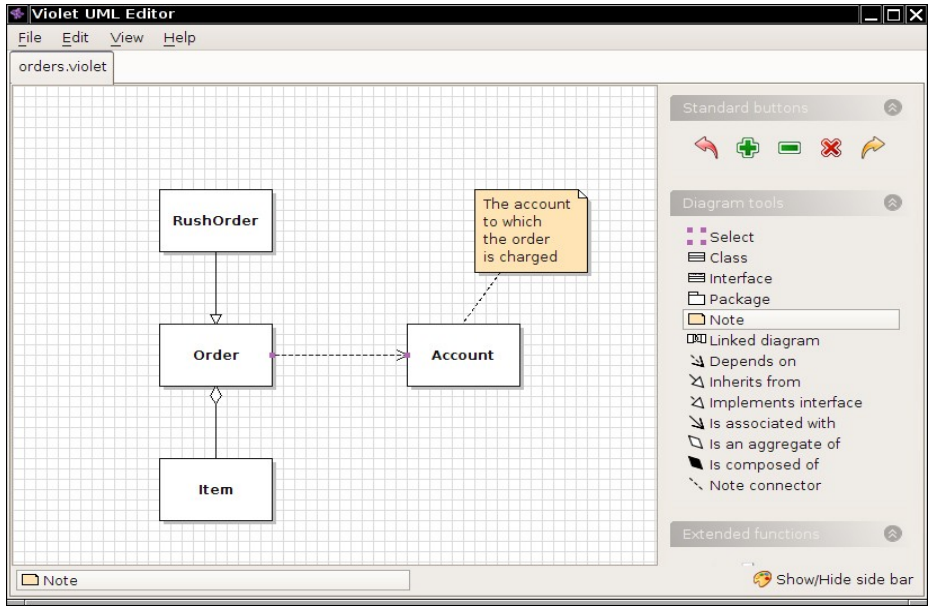
  Orders are shipped or canceled

  Payments are applied to orders

  …
  - Classes : Item, Order, Payment, etc.
  - Methods : add an item, ship an order, cancel an order, apply a payment to order

Relationship between Classes

- The most common relationships between classes
  - Association ("use-a") (⟶)
  - Aggregation and Composition ("has-a") (◇⟵ , ◆⟵)
  - Inheritance ("is-a") (◁⟵)
- Examples
  - an `Order` object *uses* an `Account` object
  - an `Order` object *has* some `Item` objects
  - a `RushOrder` object *is* an `Order` object
- UML (Unified Modeling Language)
  - Class Diagram
  - Sequence Diagram
  - Activity Diagram
  - Use Case Diagram
  - …



- Object-Oriented Analysis and Design (Object-oriented Development Methodology)