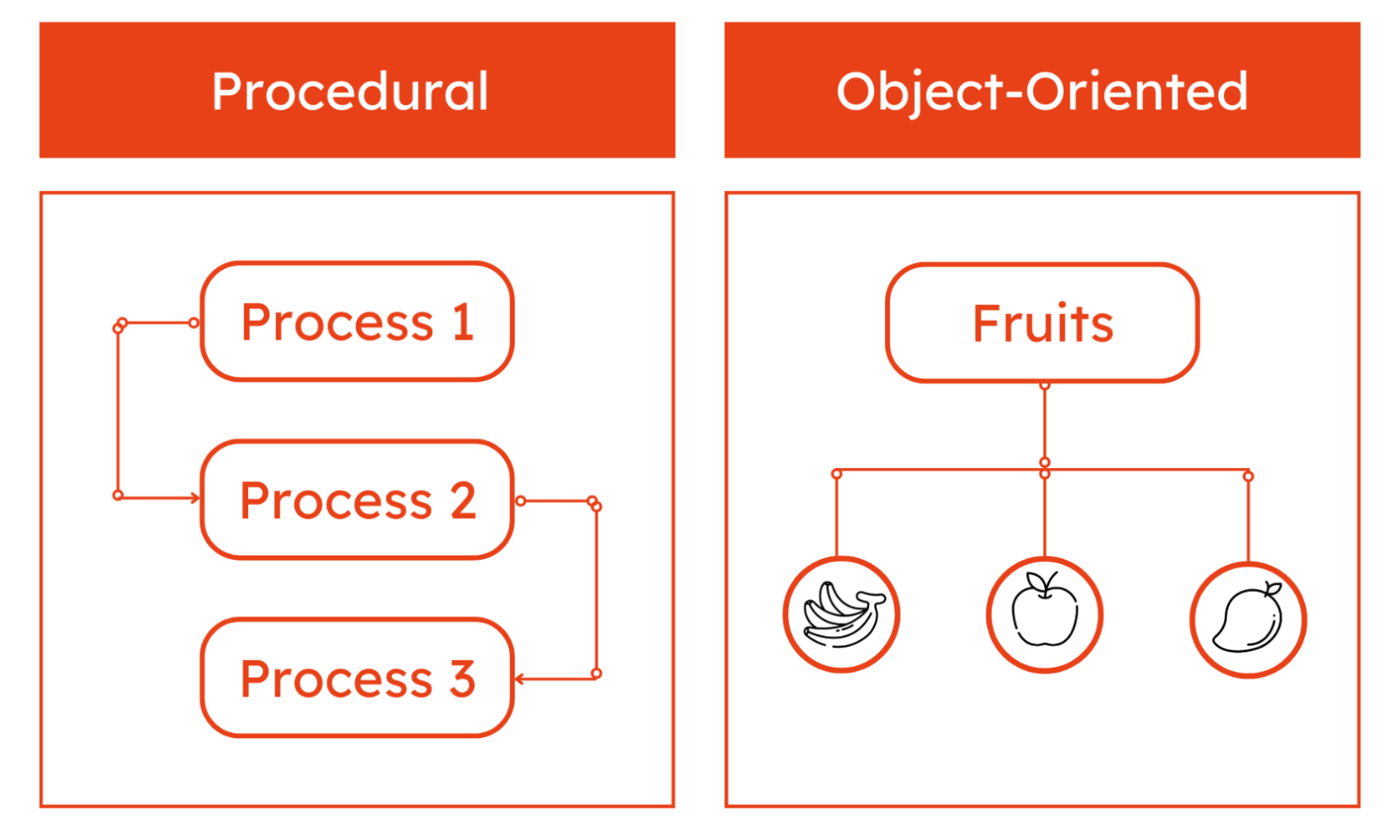
CH2.Object-Oriented Analysis& Design

School of Computer Science
Prof. Euijong Lee

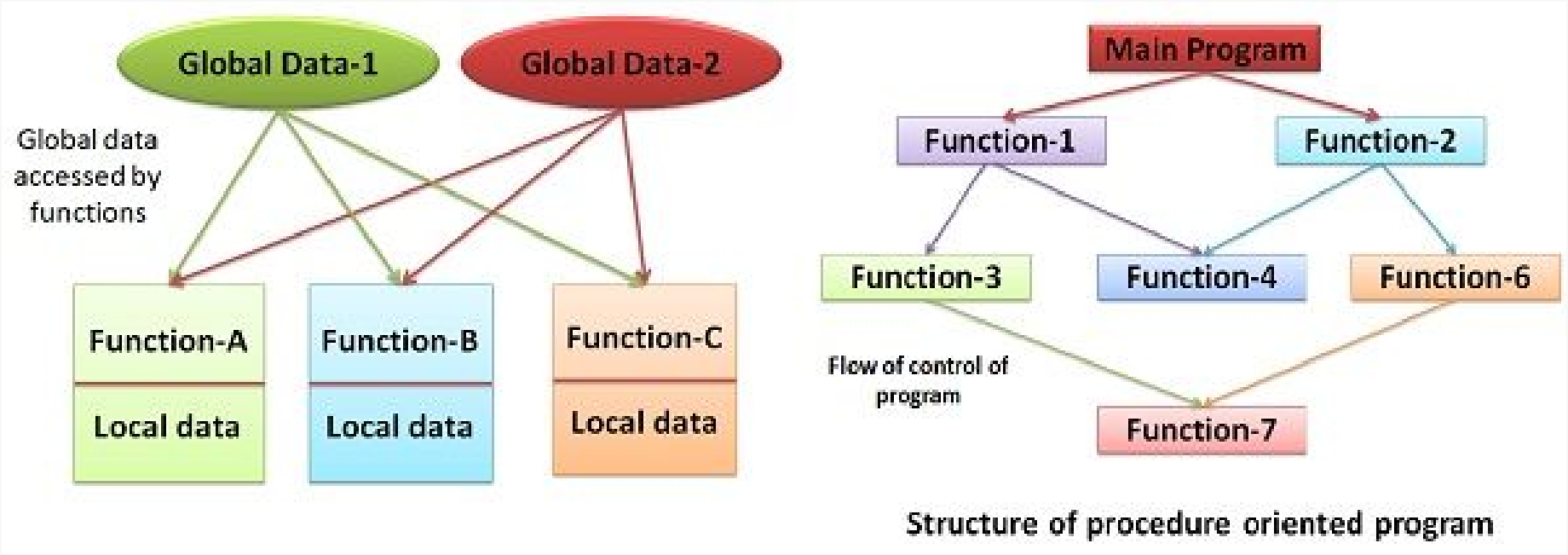
Before we start..

Procedural programming (C) VS Object-Oriented (Java, Python)



Before we start..

Procedural programming (C) VS Object-Oriented (Java, Python)



Objectives

- Understand the basic characteristics of object-oriented systems
- Be familiar with the UML (Unified Modeling Language) 2.x
- Be familiar with Unified Process (UP)
- Quick Tour OOAD with Brief Example

Basic Characteristics of Object-Orientation

- **Focus on capturing the structure and behavior of software system in little modules that encompass both data and process**
- **Basic characteristics**
  - Classes and Objects
  - Methods and Message
  - Encapsulation and Information hiding 캡슐화 정보 은닉
  - Inheritance 상속
  - Polymorphism and Dynamic Binding 다형성과 동적 바인딩

Classes and Objects (1/2)

Class 실제 객체를 표현하기 위한 틀

- Template to define specific instances or objects

Object 클래스에 정보를 추가하고 메모리 상에 올라간 것. 실제로 존재.

- Instantiation of a class

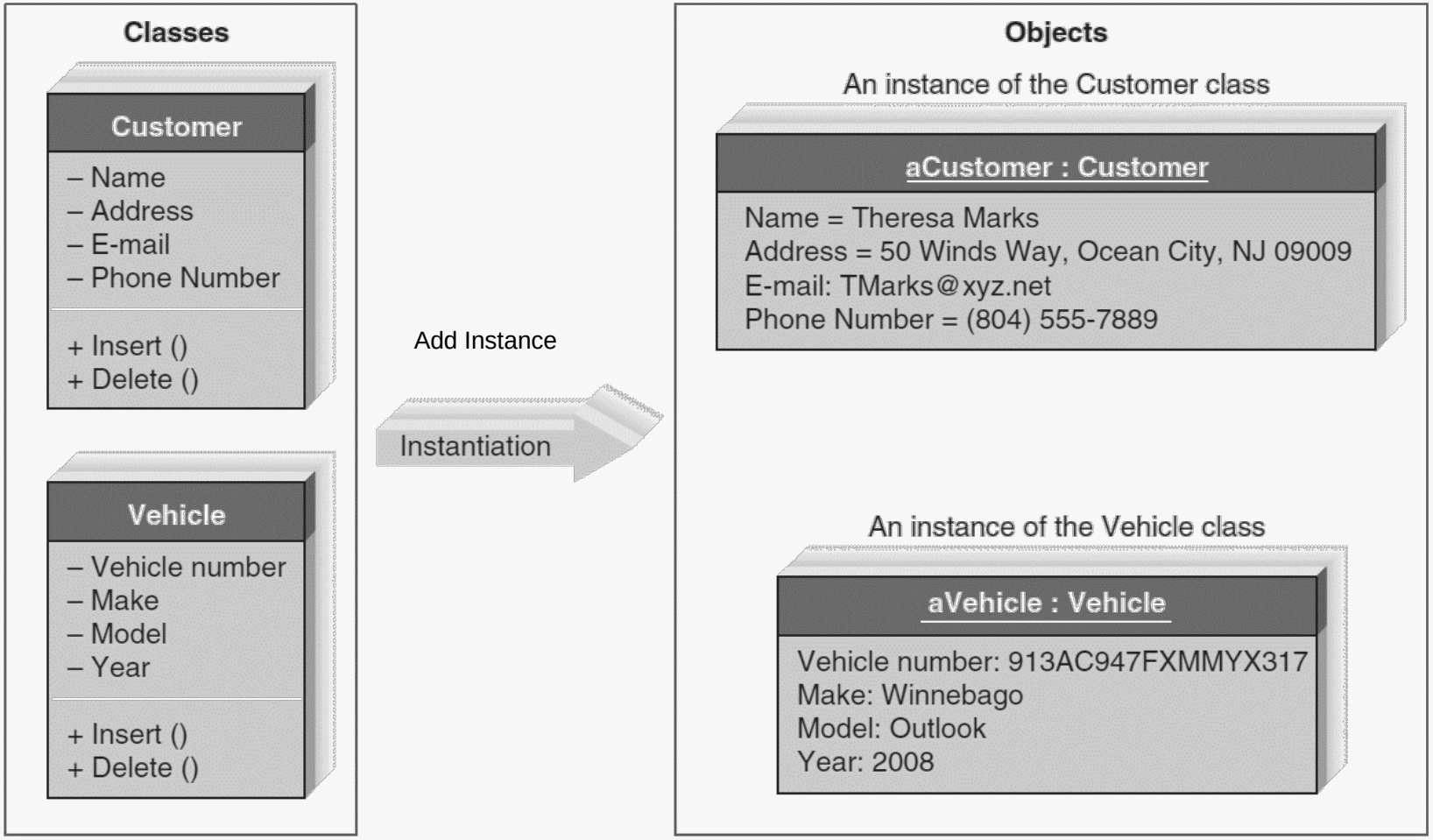
Attributes 어떤 객체를 표현해주는 것

- Describes the object

Behaviors 객체로 할 수 있는 것

- Specify what object can do

Classes and Objects (2/2)

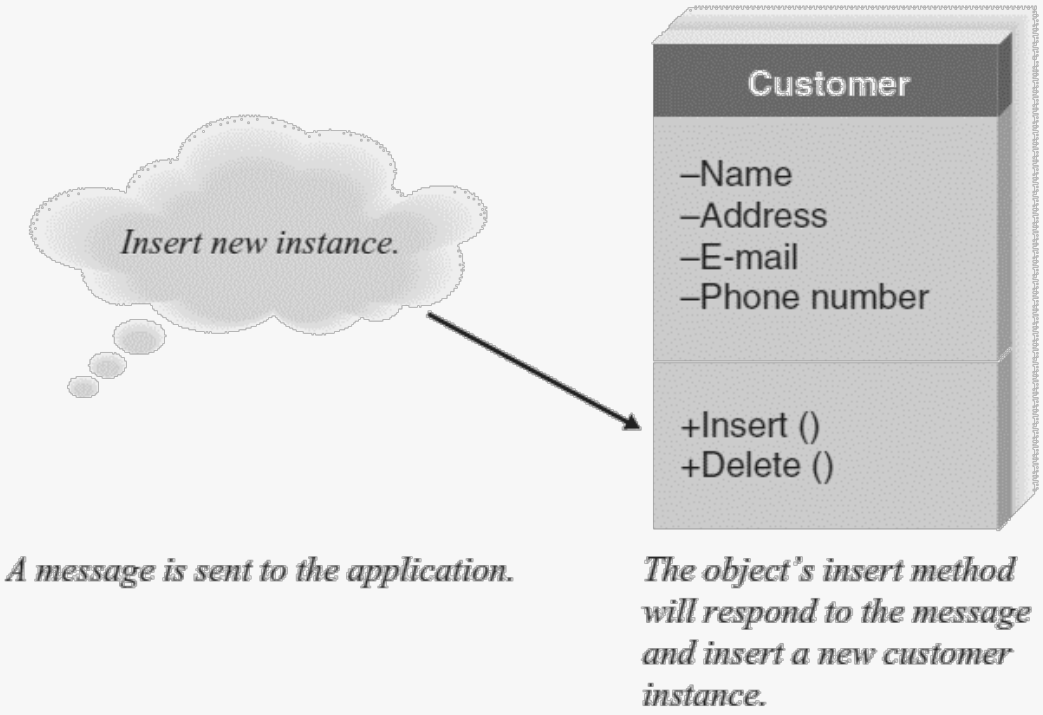


(source: Dennis, Alan, Barbara Haley Wixom, and Roberta M. Roth. *Systems analysis and design*. John wiley & sons.)

Methods and Messages

- **Methods implement an object's behavior**
  - Analogous to a function or a procedure
- **Messages are sent to trigger methods**
  - Procedure call from one object to the other

클래스 안에 있는 함수 -> 메서드



(source: Dennis, Alan, Barbara Haley Wixom, and Roberta M. Roth. Systems analysis and design. John wiley & sons.)

Encapsulation and Information Hiding

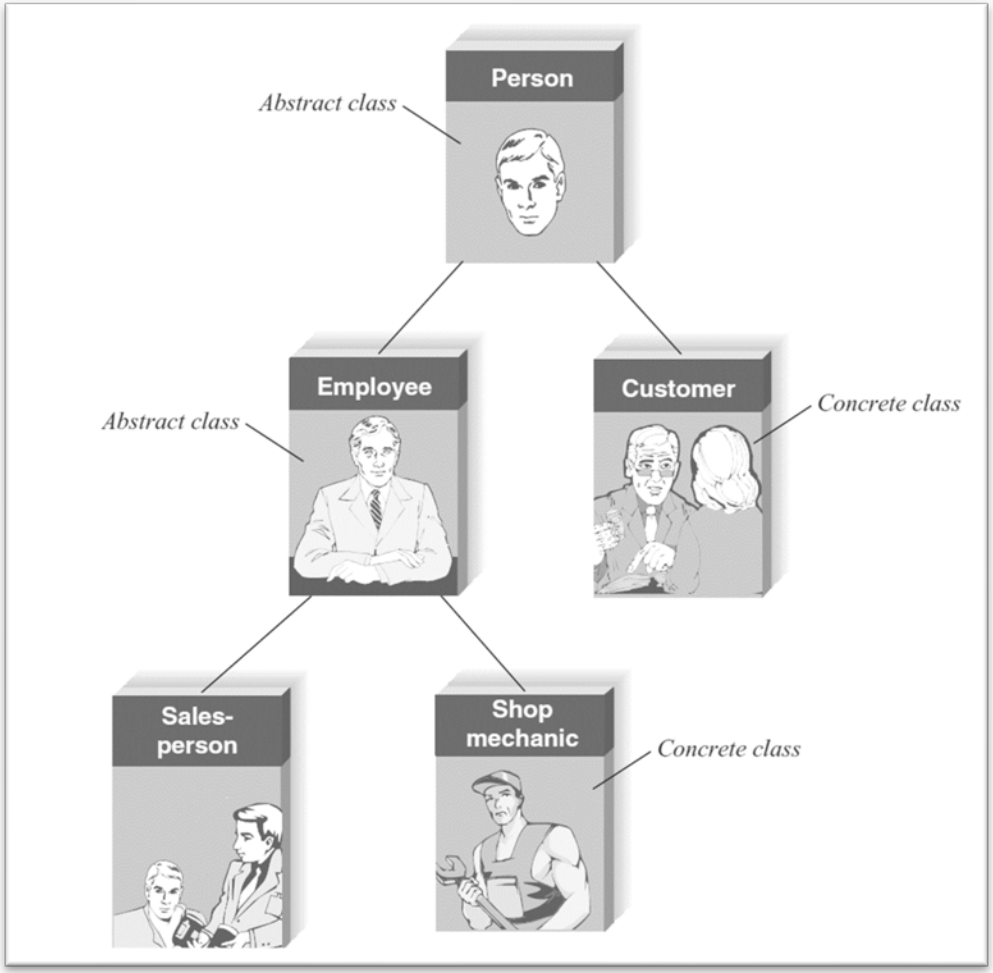
- **Encapsulation** 캡슐화
  - combination of data and process into an entity
- **Information Hiding** 정보 은닉
  - Only the information required to use a software module is published to the user
- **Reusable Key** 재사용 키
  - Use an object by calling methods

Inheritance

상속, 파생

❖ Inheritance

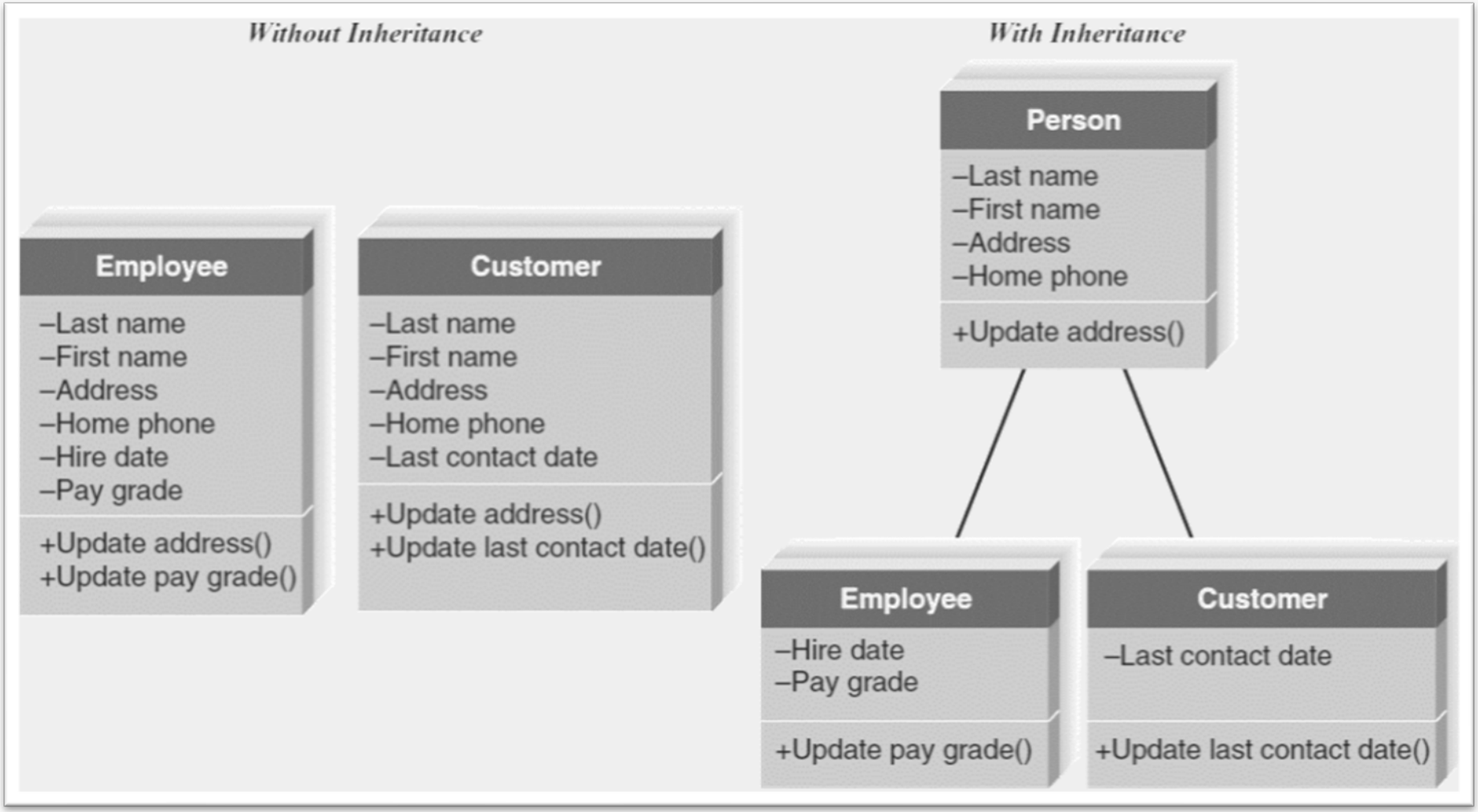
- Super classes or general classes are at the top of a hierarchy of classes
- Subclasses or specific classes are at the bottom
- Subclasses inherit attributes and methods from classes higher in the hierarchy



(source: Dennis, Alan, Barbara Haley Wixom, and Roberta M. Roth. Systems analysis and design. John wiley & sons.)

Inheritance: example

상속: 예제



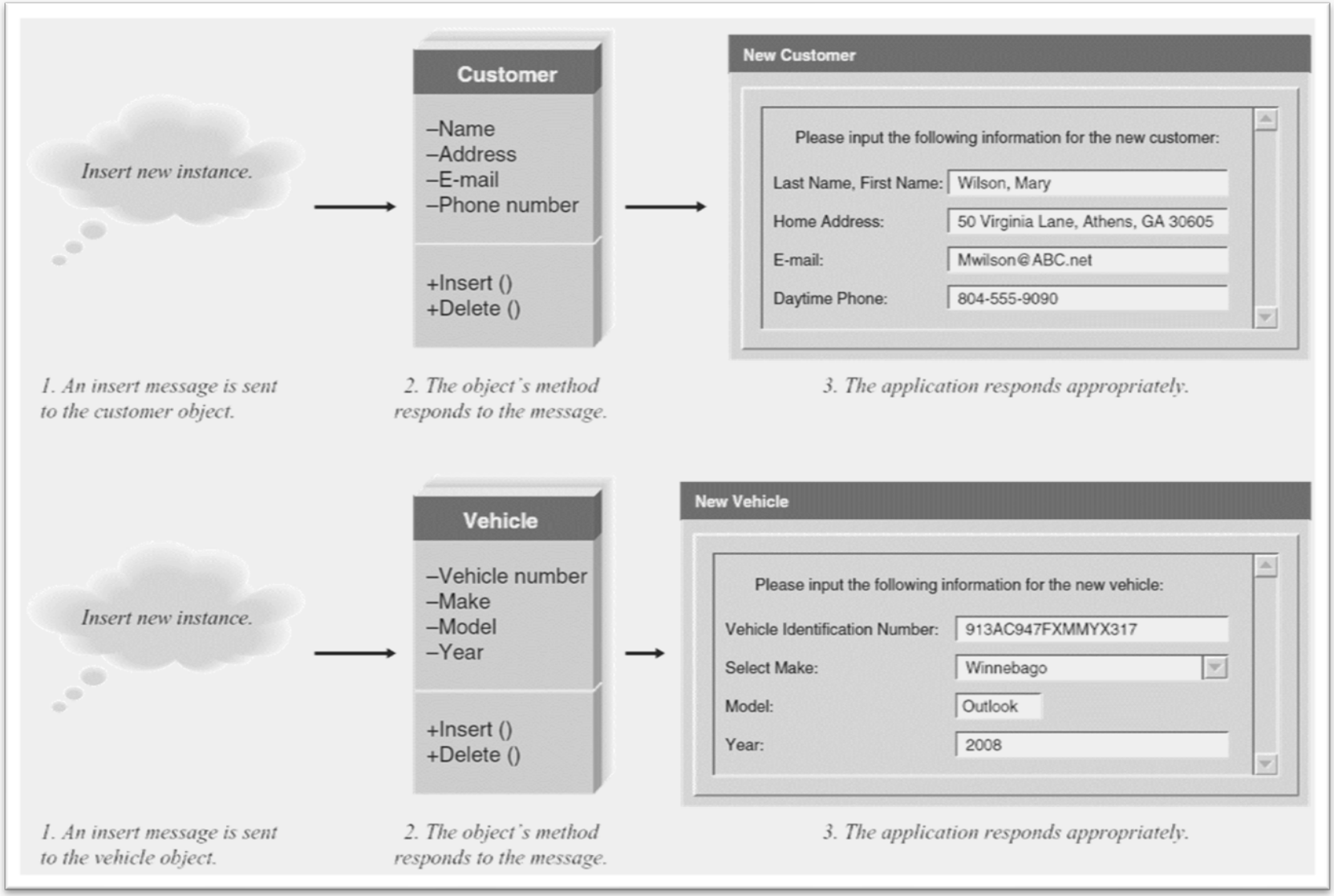
(source: Dennis, Alan, Barbara Haley Wixom, and Roberta M. Roth. Systems analysis and design. John wiley & sons.)

Polymorphism and Dynamic Binding

- **Polymorphism** 多形
  - A message can be interpreted differently by different classes of objects
- **Dynamic Binding**
  - Sometimes called late binding
  - Delays typing or choosing a method for an object until run-time
- **cf) Static Binding**
  - Type of object determined at compile time

상황에 따라서 다양하게 해석되는 성질

Polymorphism & Encapsulation

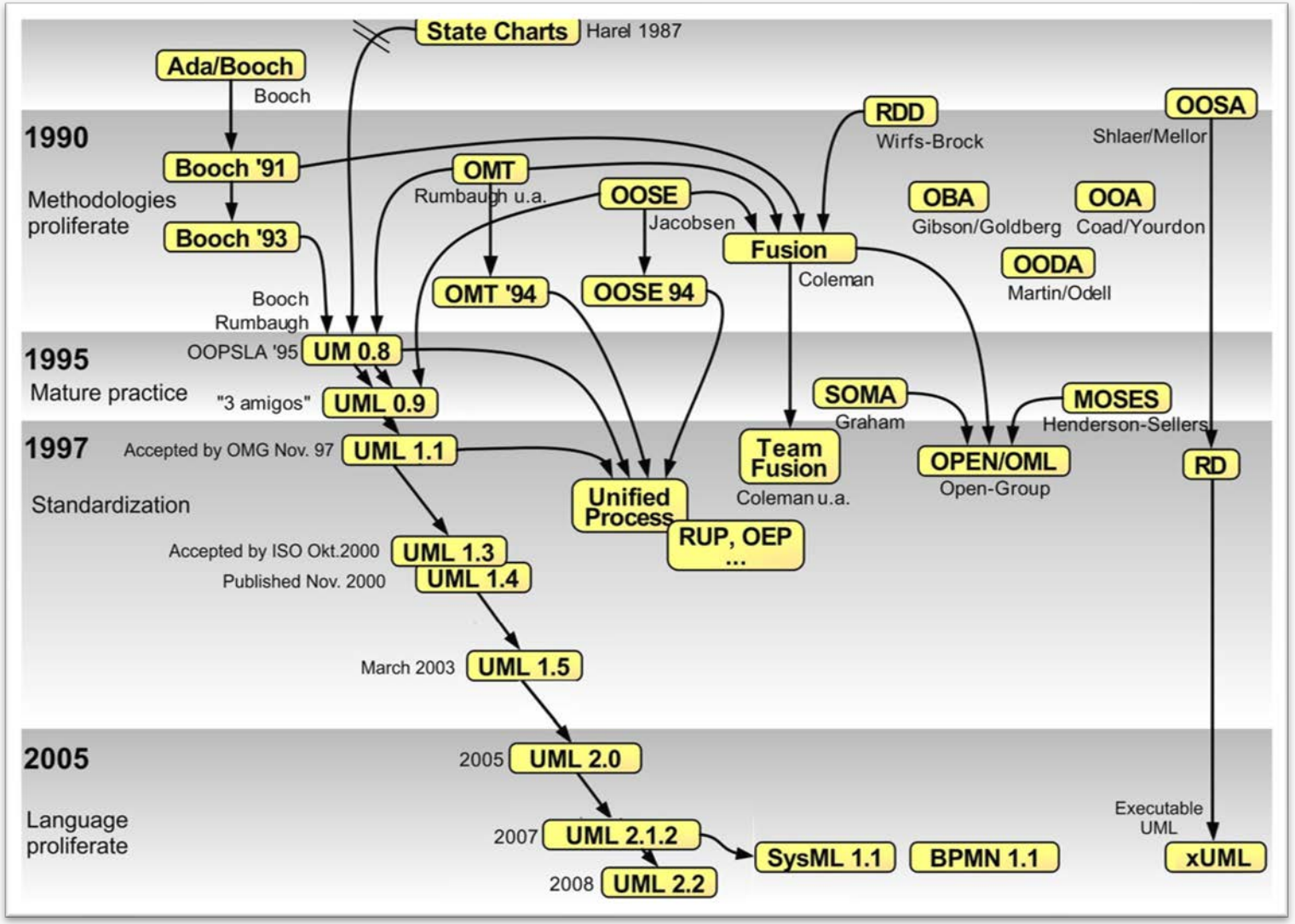


The Unified Modeling Language, Version 2.x

- **Industry standard mechanisms for visualizing, specifying, constructing, and documenting software systems.**
- **History of UML**
  - Started the work on UML in 1994
  - UML 1.0 adopted by OMG in 1997
  - UML 1.4 : widely used version to Year 2005
  - UML official version 2.0, at July 04, 2005.
  - UML 2.2 at Feb., 2009, UML 2.4.1 at Aug., 2011
  - UML 2.5 at Dec., 2017
- **Developed with**
  - Booch method by Grady Booch
  - OMT(Object Modeling Technique) by James Rumbaugh
  - OOSE(Object-Oriented Software Engineering) by Ivar Jacobson
- **UML Notation consists of**
  - Structure Diagrams
  - Behavior Diagrams
  - Extension Mechanisms

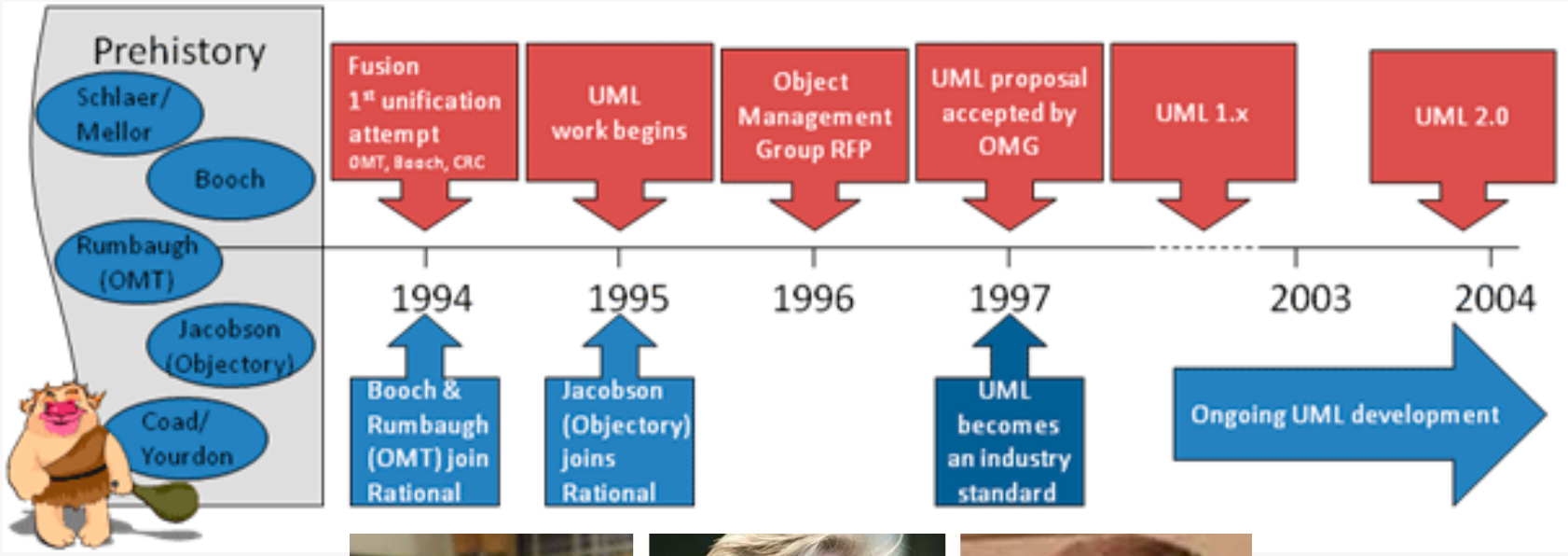
The Unified Modeling Language, Version 2.x

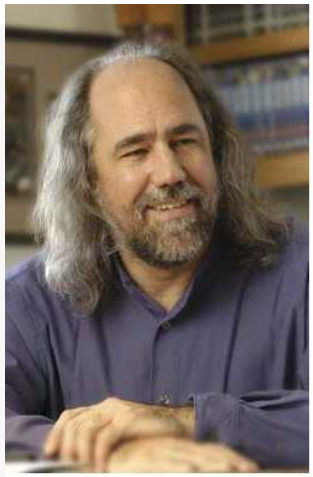
- History of object-oriented methods and notation



(source: https://en.wikipedia.org/wiki/Unified_Modeling_Language)

The Unified Modeling Language, Version 2.x





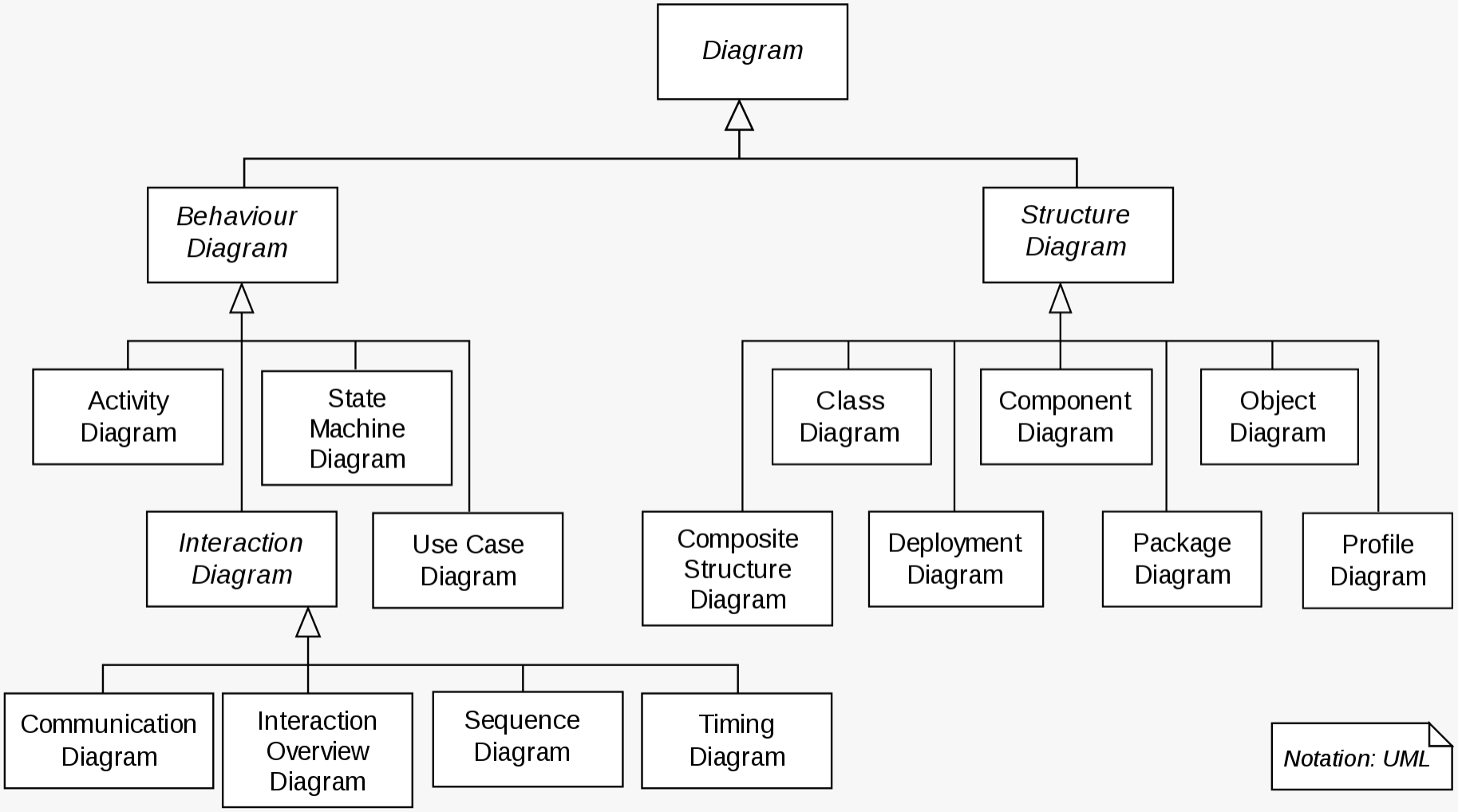
grady booch

ivar jacobson

james rumbaugh

(source: https://www.startertutorials.com/uml/introduction-to-uml.html)

The Unified Modeling Language, Version 2.x



(source: https://en.wikipedia.org/wiki/Unified_Modeling_Language)

Structure Diagram

❖ **Structure Diagrams include**

- Class diagram
- Object diagram
- Package diagram
- Deployment diagram
- Component diagram
- Composite diagram
- Profile diagram

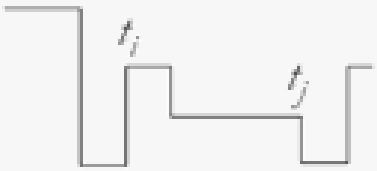
도표 그리는 방법

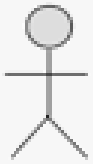
UML 2.0 Structure Diagram Summary

Diagram Name	Used to	Primary Phase
Structure Diagrams		
Class	Illustrate the relationships between classes modeled in the system.	Analysis, Design
Object	Illustrate the relationships between classes modeled in the system. Function when actual instances of the classes will better communicate the model.	Analysis, Design
Package	Group other UML elements together to form higher level constructs.	Analysis, Design, Implementation
Deployment	Show the physical architecture of the system. Can also be used to show software components being deployed onto the physical architecture.	Physical Design, Implementation
Component	Illustrate the physical relationships among the software components.	Physical Design, Implementation
Composite Structure	Illustrate the internal structure of a class-i.e., the relationships among the parts of a class.	Analysis, Design

Behavior Diagram

- **Behavior Diagrams include**
  - **Activity diagram**
  - **Interaction diagram**
    - Sequence diagram
    - Communication diagram
    - Interaction overview diagram
    - Timing diagram
  - **State Machine diagram**
  - **Use-Case diagram**





UML 2.0 Behavior Diagram Summary – (1)

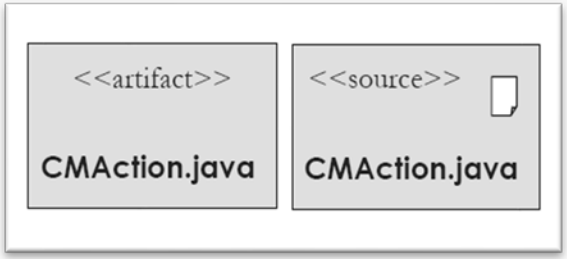
Diagram Name	Used to	Primary Phase
Behavioral Diagrams	클래스와 독립적인 비즈니스 워크플로우, 사용 사례 내 활동의 흐름 또는 메서드의 상세 설계를 설명한다.	
Activity	Illustrate business work flows independent of classes, the flow of activities in a use case, or detailed design of a method.	Analysis, Design
Sequence	사용 사례 내 객체의 동작을 모델링한다. 활동의 시간 기반 순서를 중점적으로 다룬다. Model the behavior of objects within a use case. Focuses on the time-based ordering of an activity.	Analysis, Design
Communication	사용 사례 내 객체의 동작을 모델링한다. 활동에서 협업하는 객체들 간의 의사소통에 중점을 둔다. Model the behavior of objects within a use case. Focuses on the communication among a set of collaborating objects of an activity.	Analysis, Design
Interaction Overview	Illustrate an overview of the flow of control of a process. 프로세스의 제어 흐름 개요를 설명한다.	Analysis, Design

UML 2.0 Behavior Diagram Summary – (2)

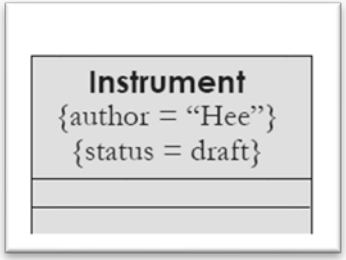
Diagram Name	Used to	Primary Phase
Behavioral Diagrams		
Timing	Illustrate the interaction that take place among a set of objects and the state changes that they go through along a time axis.	Analysis, Design
Behavioral State Machine	Examine the behavior of on class.	Analysis, Design
Protocol State Machine	Illustrate the dependencies among the different interfaces of a class.	Analysis, Design
Use Case	Capture business requirements for the system and to illustrate the interaction between the system and its environment.	Analysis

Extension Mechanisms

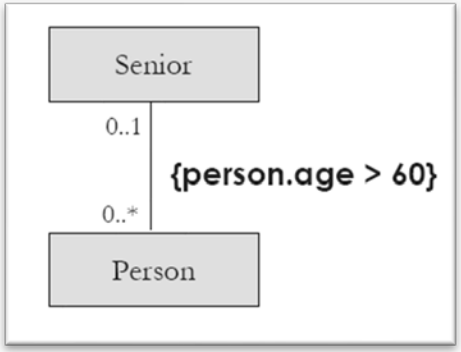
- **Stereotype**
  - A type of modeling element that extends the semantics of the UML
  - Shown as a text item enclosed within angle brackets(<< >>)



- **Tagged Values**
  - Add new properties to base elements

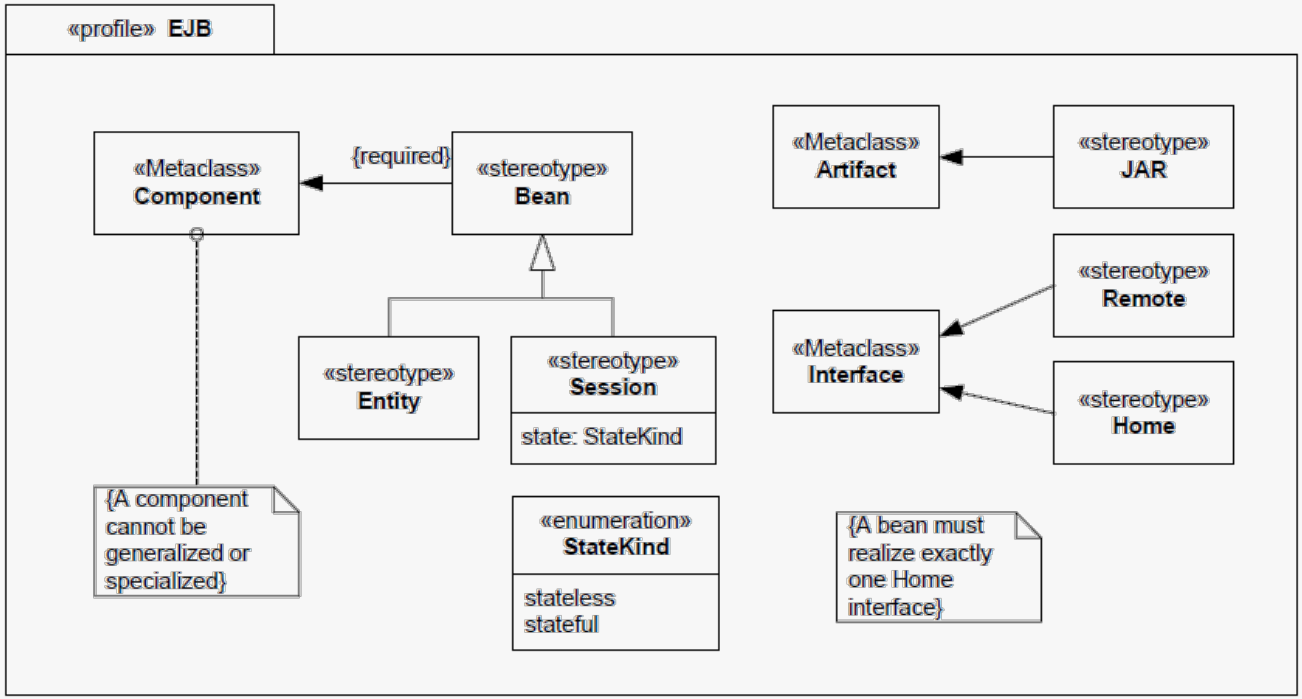


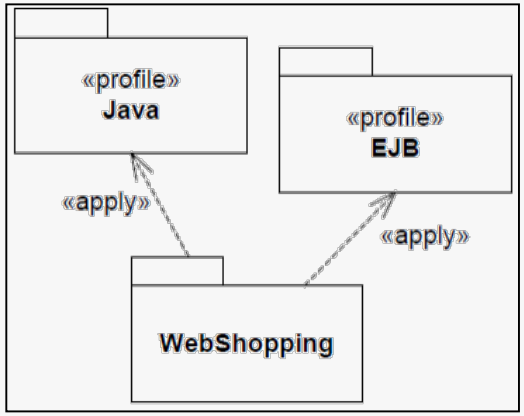
- **Constraints**
  - Place restrictions on use of model elements using OCL



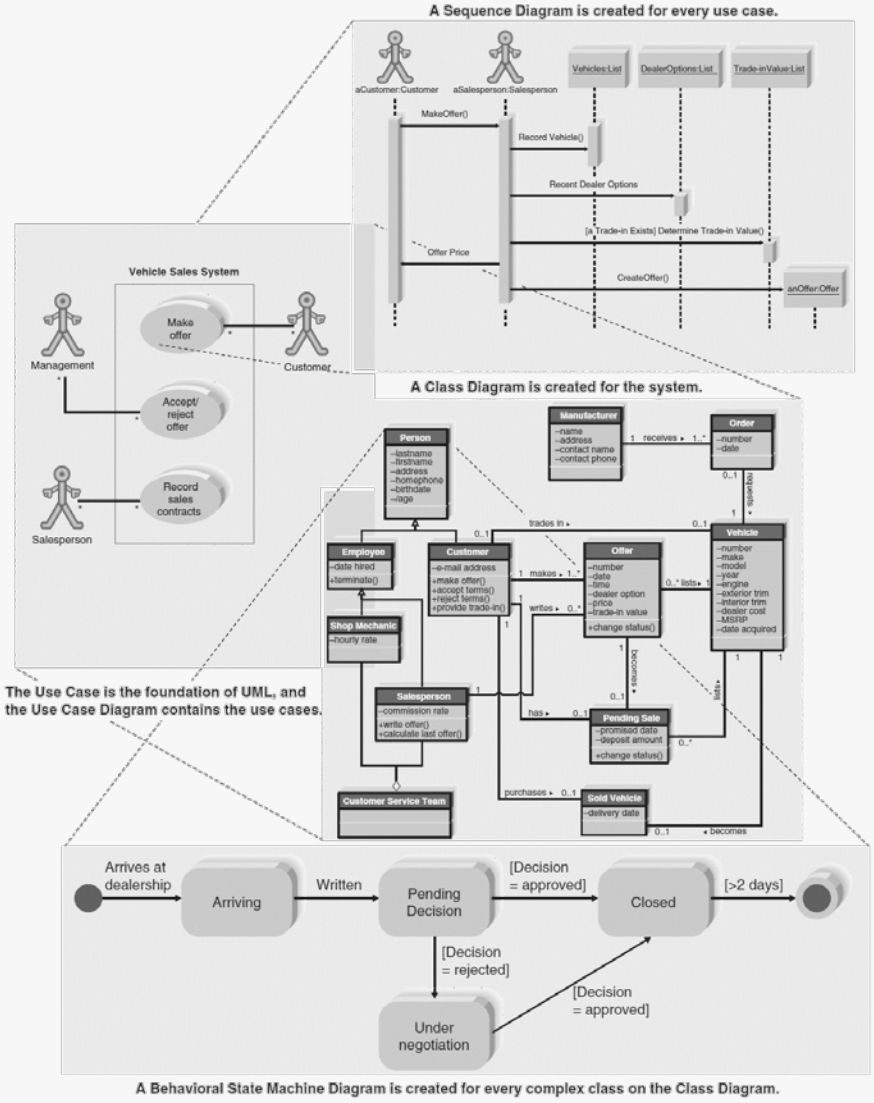
Extension Mechanisms

- **Profiles**
  - Group model elements that have been extended using stereotype, tagged values, and/or constraints into a package
  - A stereotyped package to manage sets of extension





The integration of UML diagrams



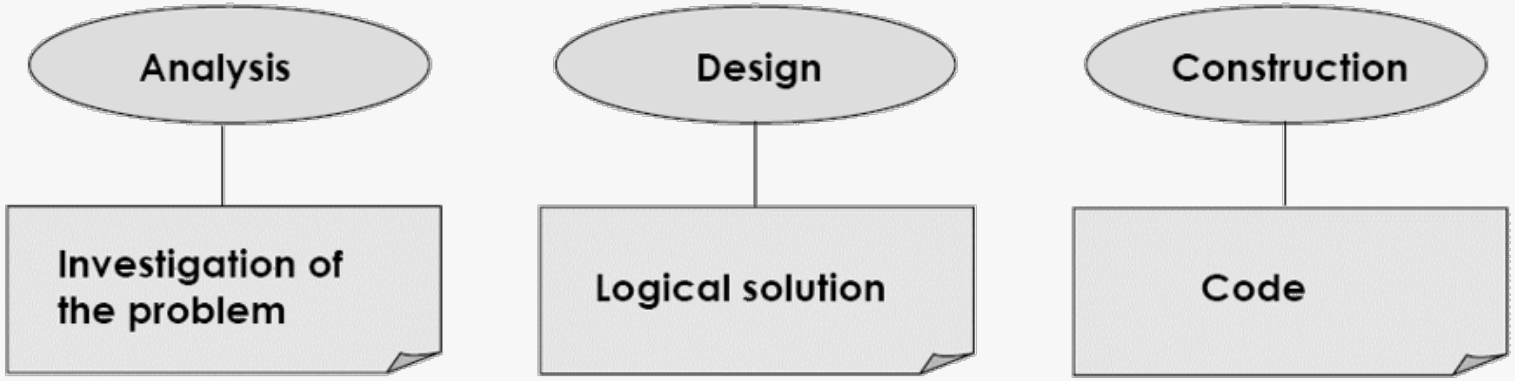
Object-Oriented Analysis and Design (1/3)

- **Analysis**
  - Emphasize an investigation of the problem rather than how a solution is defined

정의된 문제를 어떻게 해결하는가 보다는 연구에 강조한다.

- **Design**
  - Emphasize a conceptual solution, how the system fulfills the requirements

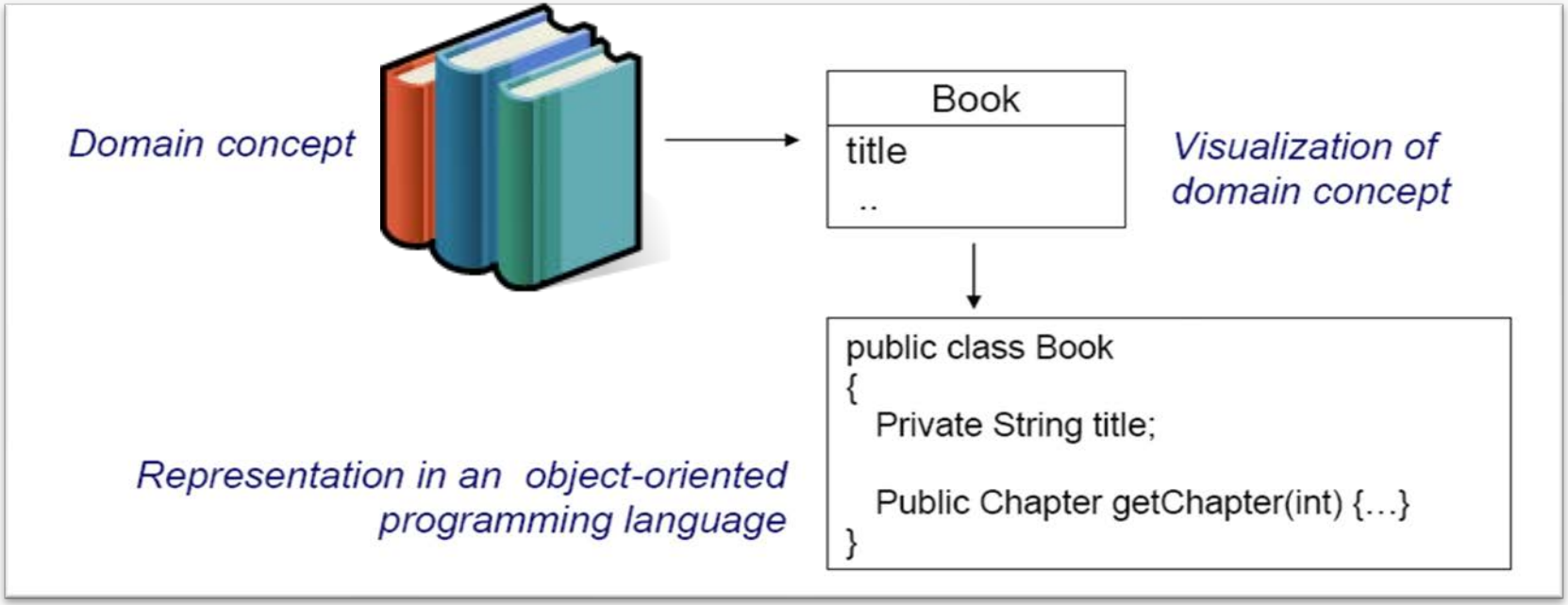
어떻게 요구사항을 시스템에 채울지 개념적 해결책을 강조한다.



Object-Oriented Analysis and Design (2/3)

- **Object-Oriented Analysis**
  - Finding and Describing the objects (or concepts)

- **Object-Oriented Design**
  - Defining software objects and how they collaborate to fulfill the requirements



Object-Oriented Analysis and Design (3/3)

❖ **OOAD approaches**

- **Use-case driven approach**
  - Use case are the primary modeling tool to define the behavior of the system
- **Architecture Centric** ㄴ 주요한
  - Underlying software architecture derives the specification, construction, and documentation of the system
  - Functional, static, dynamic architectural views of a system
- **Iterative and Incremental**
  - Development undergoes continuous testing and refinement throughout the life of the project ㄴ 견디다, 겪다, 받다
- **The Unified Process (UP)**
  - Development process that map out when and how to use the various UML techniques for OOAD
  - Support use-case driven, architecture-centric, and iterative and incremental approach

Benefits of the Object Approach

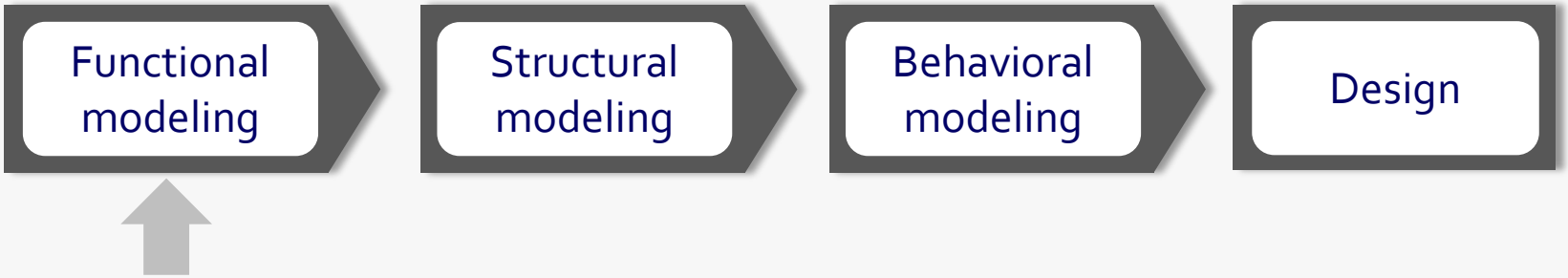
Concepts	Supports	Leads to
Classes, objects, methods, and messages	• A more realistic way for people think about their business • Highly cohesive (결합력 있는) units that contain both data and processes	• Better communication between user and analyst or developer • Reusable objects • Benefits from having a highly cohesive system(See cohesion in Chapter 10)
Encapsulation and information hiding	• Loosely coupled units	• Reusable objects (응집력) • Fewer ripple effects (파급 효과) from changes within an object or in the system itself • Benefits from having a loosely coupled system design(See coupling in Chapter 10)
Inheritance	• Allows us to use classes as standard templates from which other classes can be built	• Less redundancy (중복성) • Faster creation of new classes • Standards and consistency (일관성) within and across development efforts • Ease in supporting exceptions

Benefits of the Object Approach

Concepts	Supports	Leads to
Polymorphism	• Minimal messaging that is interpreted by objects themselves	• Simpler programming of events • Ease in replacing or changing objects in a system • Fewer ripple effects (파급 효과) from changes within an object or in the system itself
Use case driven	• Allows users and analysts to focus on how a user will interact with the system to perform a single activity	• Better understanding and gathering of user needs • Better communication between user and analyst
Architecture centric and functional, static, and dynamic views	• Viewing the evolving system from multiple points of view	• Better understanding and modeling of user needs • More complete depiction of information system
Iterative and incremental development	• Continuous testing and refinement of the evolving system	• Meeting real needs of users • Higher quality systems

Quick Tour OOAD by Example

Example: Dice Game



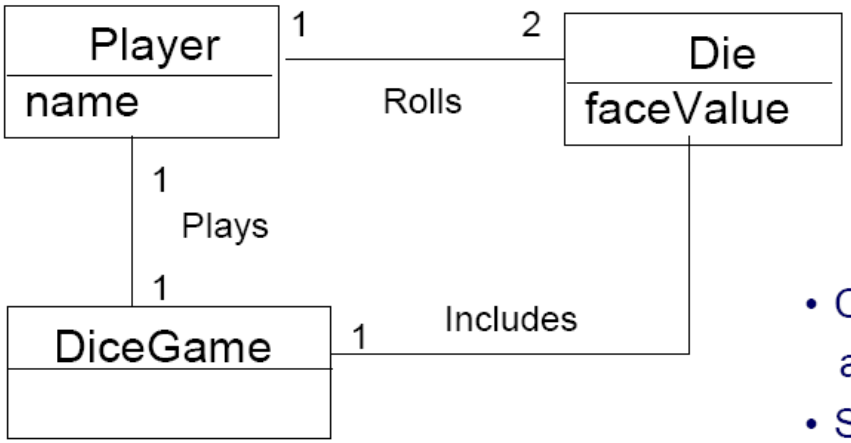
Use Case : Play a Dice Game

A player picks up and rolls the dice. If the dice face value total seven, they win; otherwise, they lose

Quick Tour OOAD by Example

Example: Dice Game

Functional modeling → Structural modeling → Behavioral modeling → Design

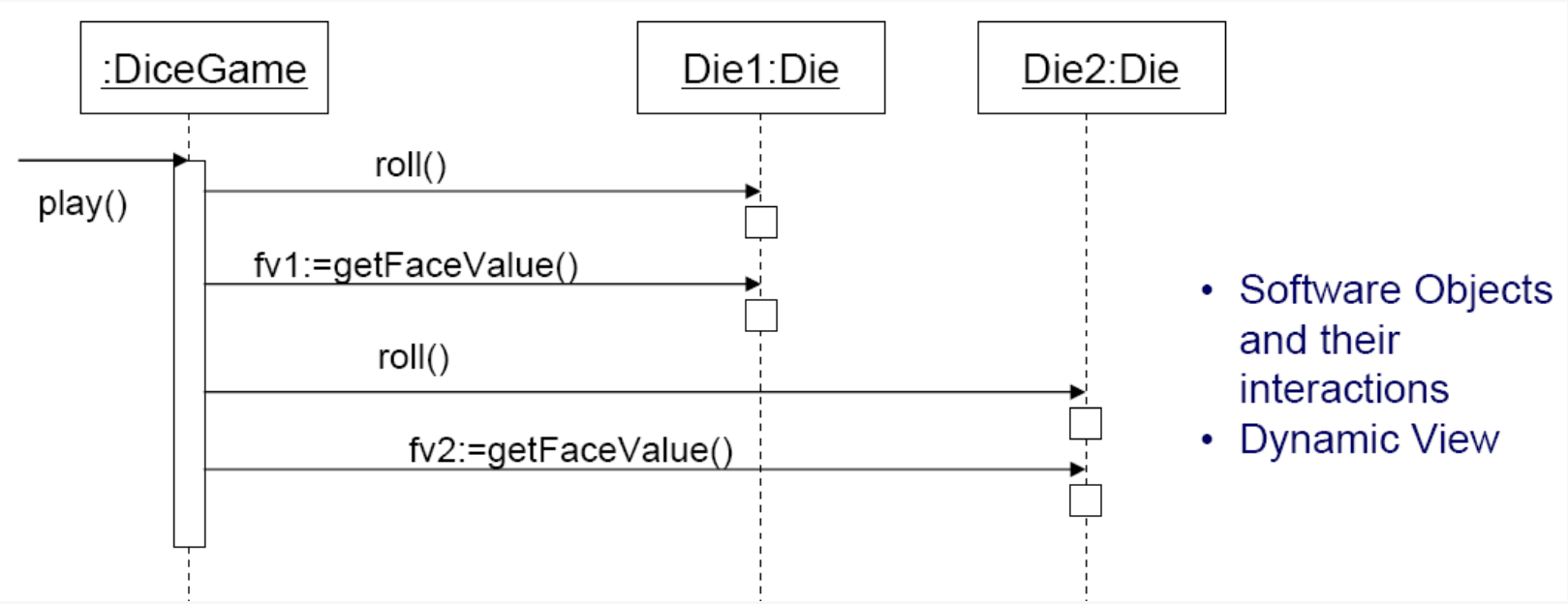


- Concepts, attributes, and associations
- Static view

Quick Tour OOAD by Example

Example: Dice Game

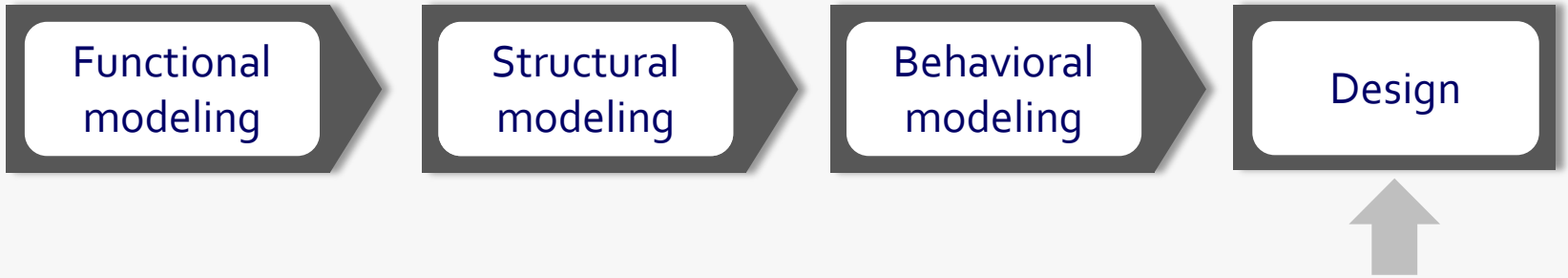
Functional modeling → Structural modeling → Behavioral modeling → Design

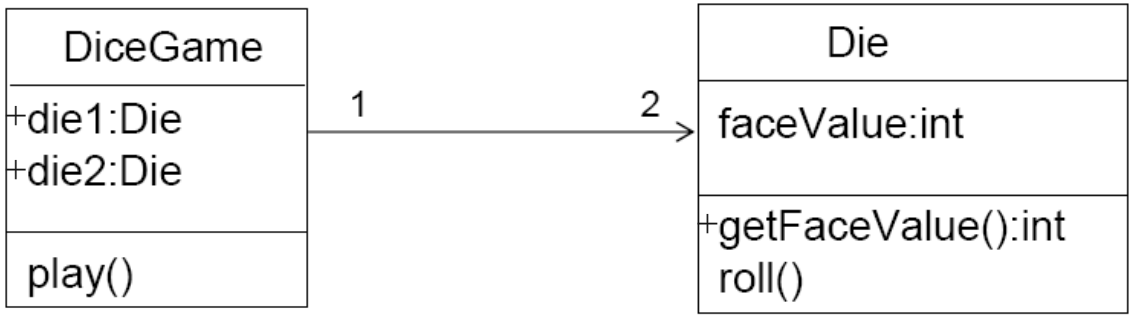


- Software Objects and their interactions
- Dynamic View

Quick Tour OOAD by Example

Example: Dice Game



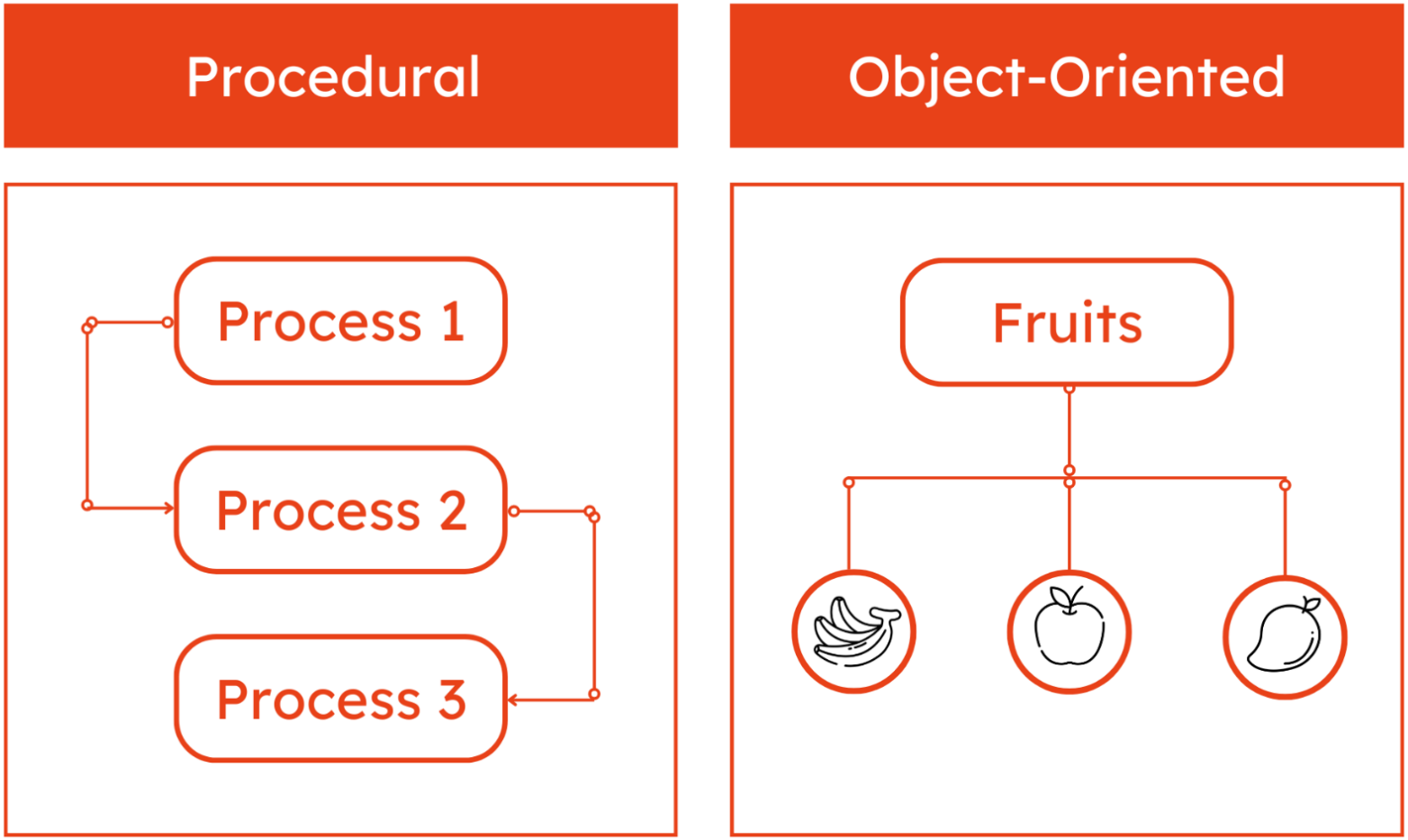


- Class and method design
- Physical design

Before the end…

Now, you can understand following Figure?

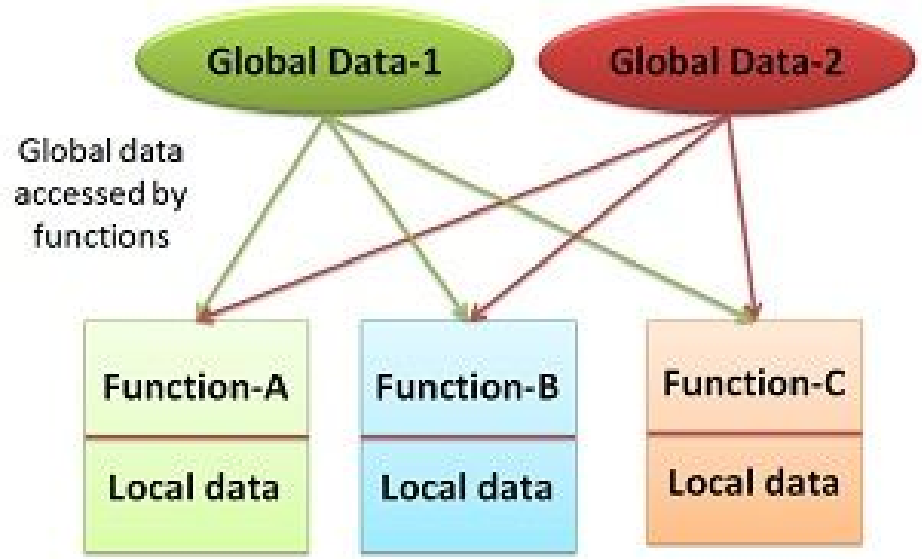
Procedural programming (C) VS Object-Oriented (Java, Python)

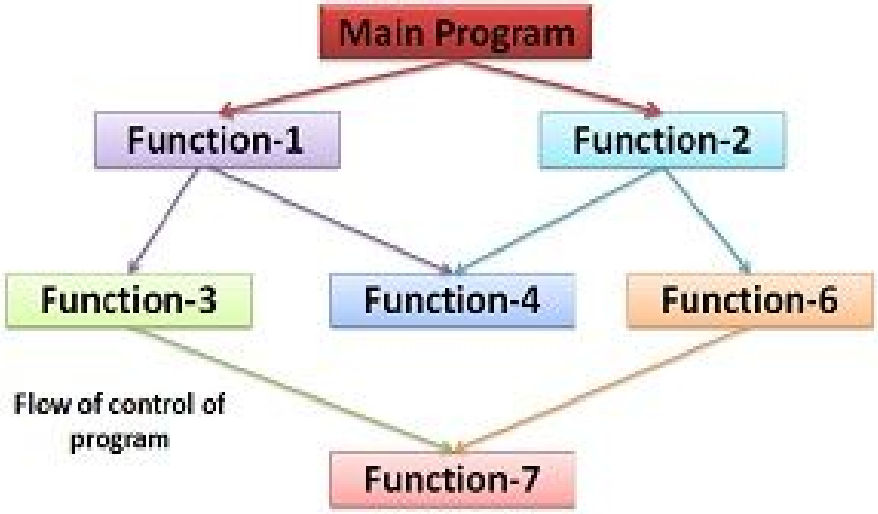


Before we start..

Now, you can understand following Figure?

Procedural programming (C) VS Object-Oriented (Java, Python)





Structure of procedure oriented program

Summary and Discussion

- Basic characteristics of an object-orientation.
- Unified Modeling Language, briefly.
- Unified Process.

- Which programming language supports object-oriented concepts ?

- Should fourteen diagrams be developed without considering the software and/or project characteristics ?

- IF YOU WANT MORE EXAMPLE
  - https://levelup.gitconnected.com/explain-by-example-oop-24fe5d6c978