충북대학교
SW중심대학사업단

CH10. Class & Method Design

School of Computer Science
Prof. Euijong Lee

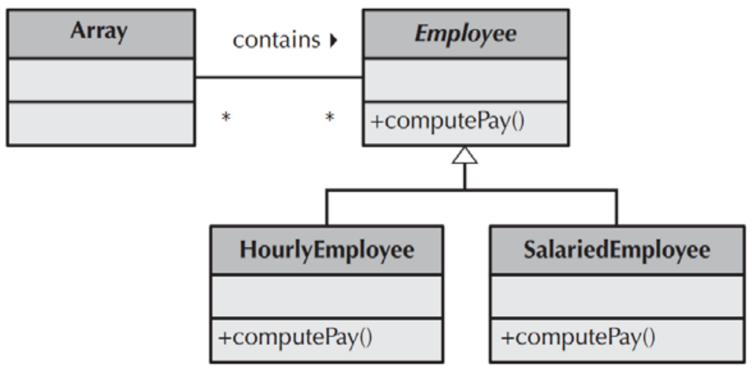
Objectives

- Be able to specify, restructure, and optimize objects design
- Be able to identify the reuse of predefined classes
- Be able to specify constraints and contracts
- Be able to create a method specification

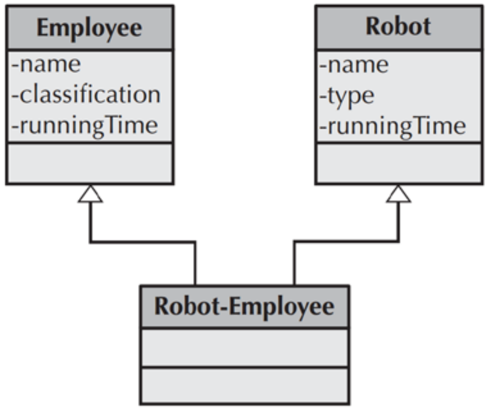
Review of the Basic Characteristics of Object Oriented

틀 실체를 가진것 기능 메시지

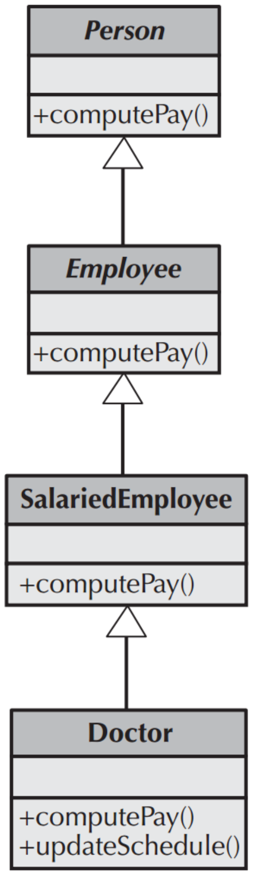
- ❖ Classes, Objects, Methods, and Messages
- ❖ Encapsulation and information hiding
- ❖ Polymorphism and dynamic binding
- ❖ Inheritance



<Example of polymorphism>



<Inheritance Conflicts with multiple Inheritance >



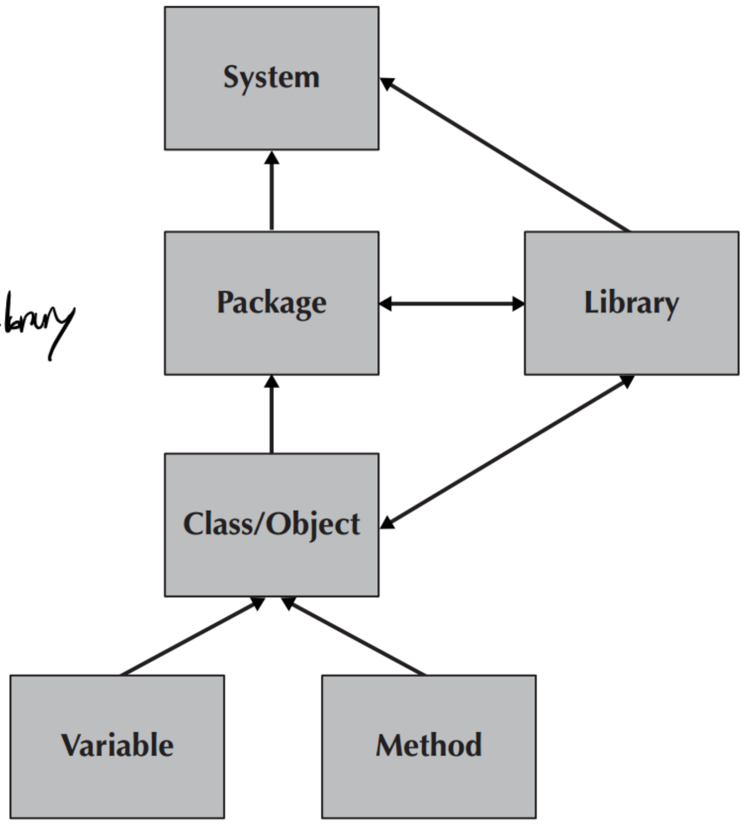
<Example of Redefinition and Inheritance Conflict >

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

Level of Object-Oriented Systems

❖ **In an object-oriented system, changes can take place at different levels of abstraction, and it includes**

- Variable
- Method
- Class / object
- Package = Class + Object
- Library = Package 확장 (일반적인 기능 제공)
- Application / System level = package + Library



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

Design Criteria

결합도

❖ **A set of metrics to evaluate the design Variable**

- **Coupling**—refers to the degree of the closeness of the relationship between classes

  클래스끼리 얼마나 가까운가? 약하게 결합되어야 좋다.

  응집도

- **Cohesion**—refers to the degree to which attributes and methods of a class support a single object

  높을수록 좋다.

- **Connascence**—refers to the degree of interdependency between objects

  ↳ Coupling + Cohesion 얼마나 상관관계가 있는가?

Coupling

❖ **Close coupling means that changes in one part of the design may require changes in another part**

❖ **Types** 상호 작용 결합도
- **Interaction coupling** measured through message passing
- **Inheritance coupling** deals with the inheritance hierarchy of classes

상속 관계 결합도 상속 계층

❖ **Minimize interaction coupling by restricting messages (Law of Demeter)** → 객체구조 → 다른 객체 알려주지 X

❖ **Minimize inheritance coupling by using inheritance to support only generalization/specialization and the principle of substitutability**

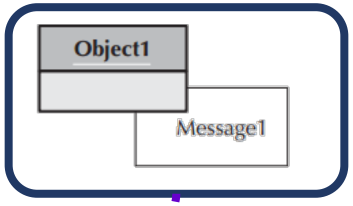
Law of Demeter

데메테르 법칙

❖ **The law states that an object should send messages**

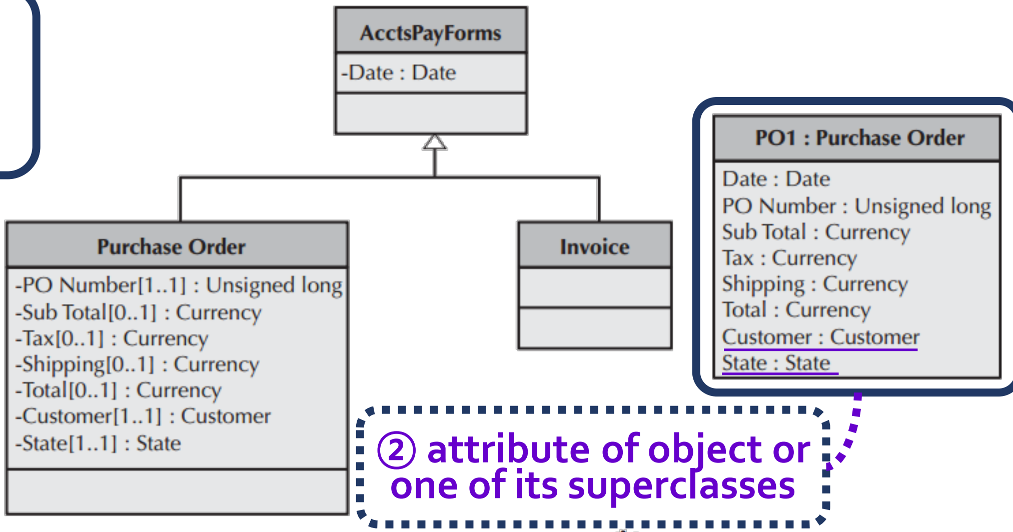
1. Itself
2. An object that is contained in an attribute of object or one of its super-classes
3. An object that is passed as a parameter to the method
4. An object that is created by the method
5. An object that is stored in a global variable

Example: Interaction Coupling



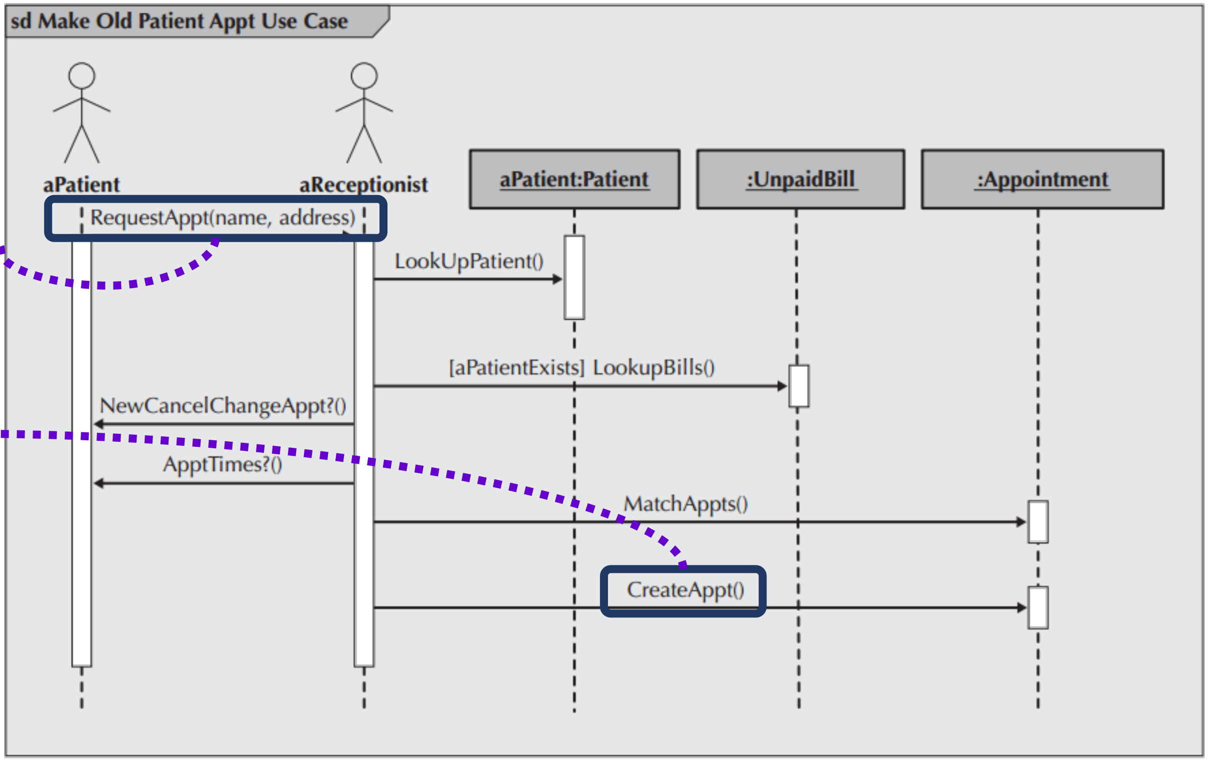
① itself

(a)



② attribute of object or one of its superclasses

(b)



③ An object that is passed as a parameter to the method

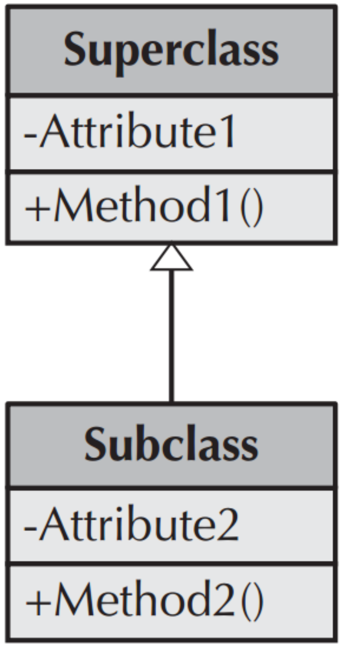
④ An object that is created by the method

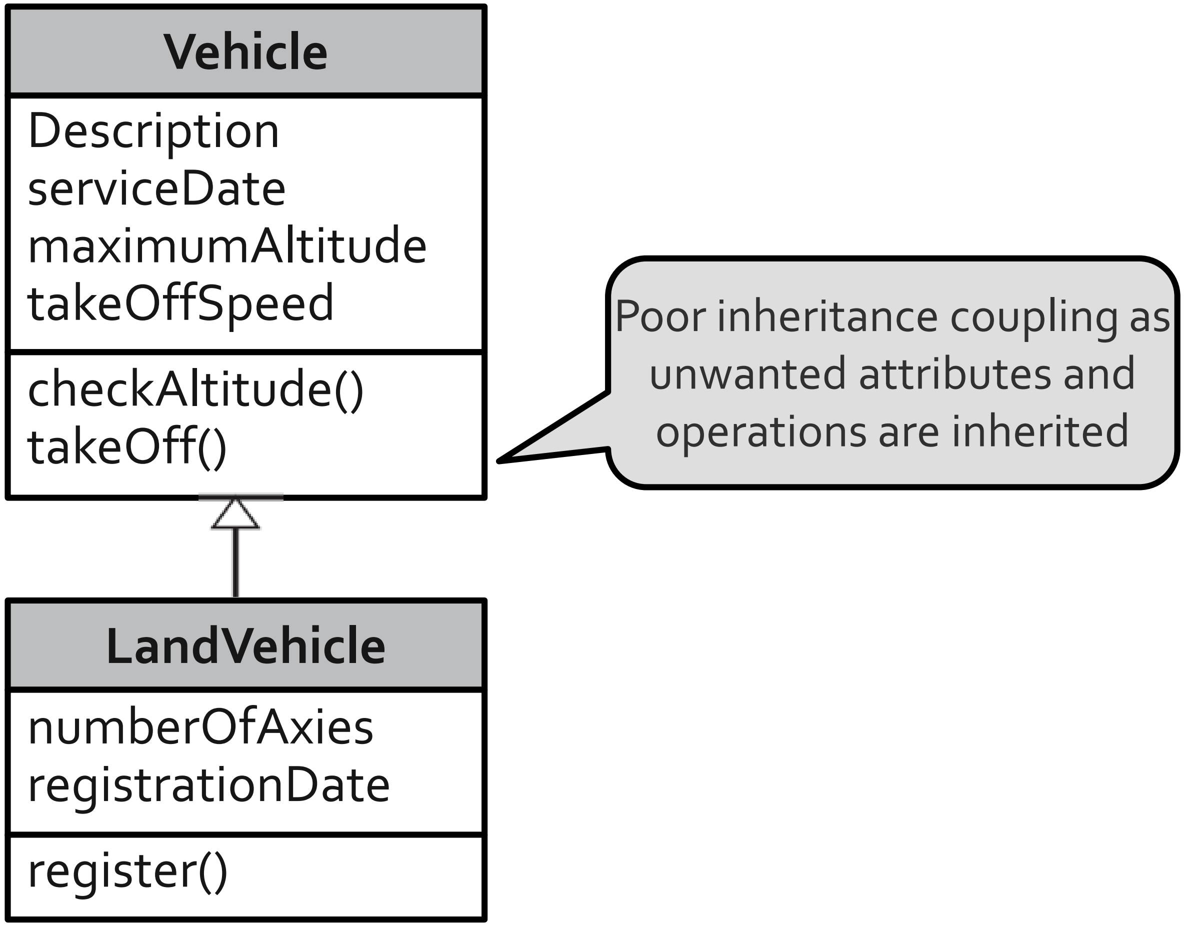
(source: Dennis, Alan, Barbara Haley Wixom, and Roberta M. Roth. Systems analysis and design. John wiley & sons.)

(c)

Example: Inheritance Coupling

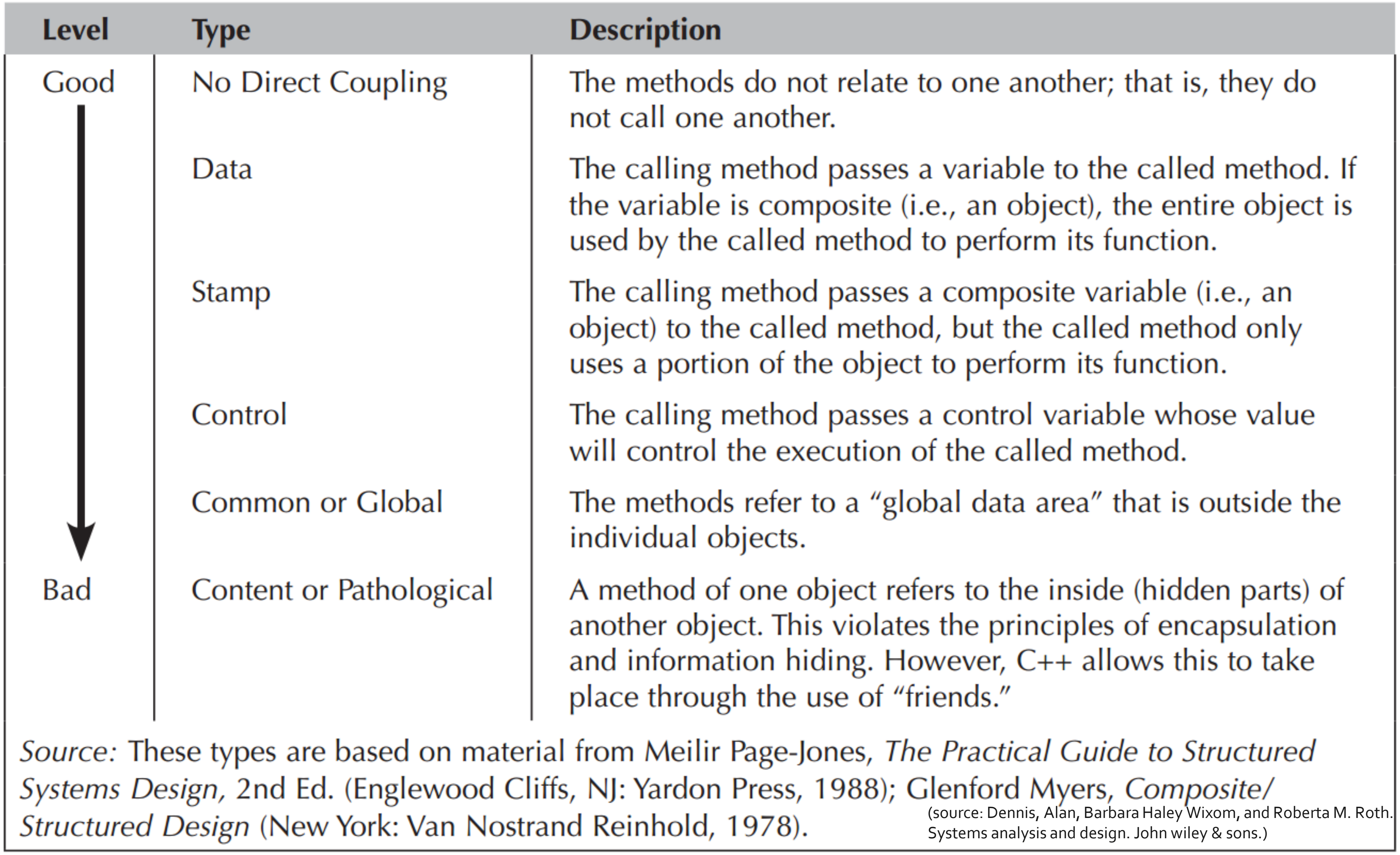
- Inheritance Coupling implies, deals with how tightly coupled the classes are in an inheritance hierarchy





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

Types of Interaction Coupling



Level	Type	Description
Good	No Direct Coupling	The methods do not relate to one another; that is, they do not call one another.
	Data	The calling method passes a variable to the called method. If the variable is composite (i.e., an object), the entire object is used by the called method to perform its function.
	Stamp	The calling method passes a composite variable (i.e., an object) to the called method, but the called method only uses a portion of the object to perform its function.
	Control	The calling method passes a control variable whose value will control the execution of the called method.
	Common or Global	The methods refer to a "global data area" that is outside the individual objects.
Bad	Content or Pathological	A method of one object refers to the inside (hidden parts) of another object. This violates the principles of encapsulation and information hiding. However, C++ allows this to take place through the use of "friends."

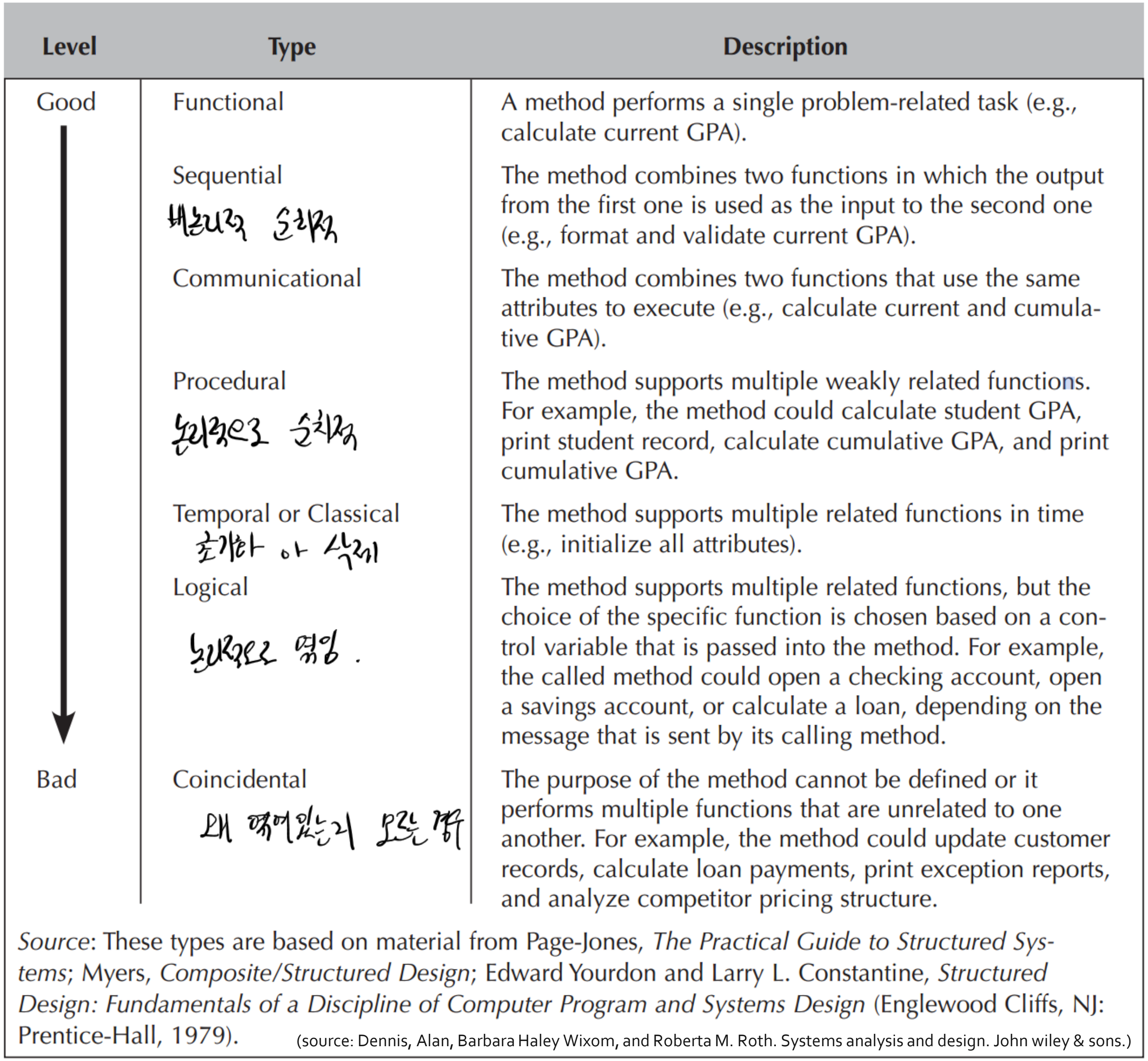
Source: These types are based on material from Meilir Page-Jones, *The Practical Guide to Structured Systems Design*, 2nd Ed. (Englewood Cliffs, NJ: Yardon Press, 1988); Glenford Myers, *Composite/Structured Design* (New York: Van Nostrand Reinhold, 1978).

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

Cohesion

- ❖ **A cohesive class, object or method refers to a single thing**
- ❖ **Types**
  - **Method cohesion**
    - Does a method perform more than one operation?
    - Performing more than one operation is more difficult to understand and implement
  - **Class cohesion**
    - Do the attributes and methods represent a single object?
    - Classes should not mix class roles, domains or objects
  - **Generalization / Specialization cohesion**
    - Classes in a hierarchy should show “a-kind-of” relationship, not associations or aggregations

Types of Method Cohesion

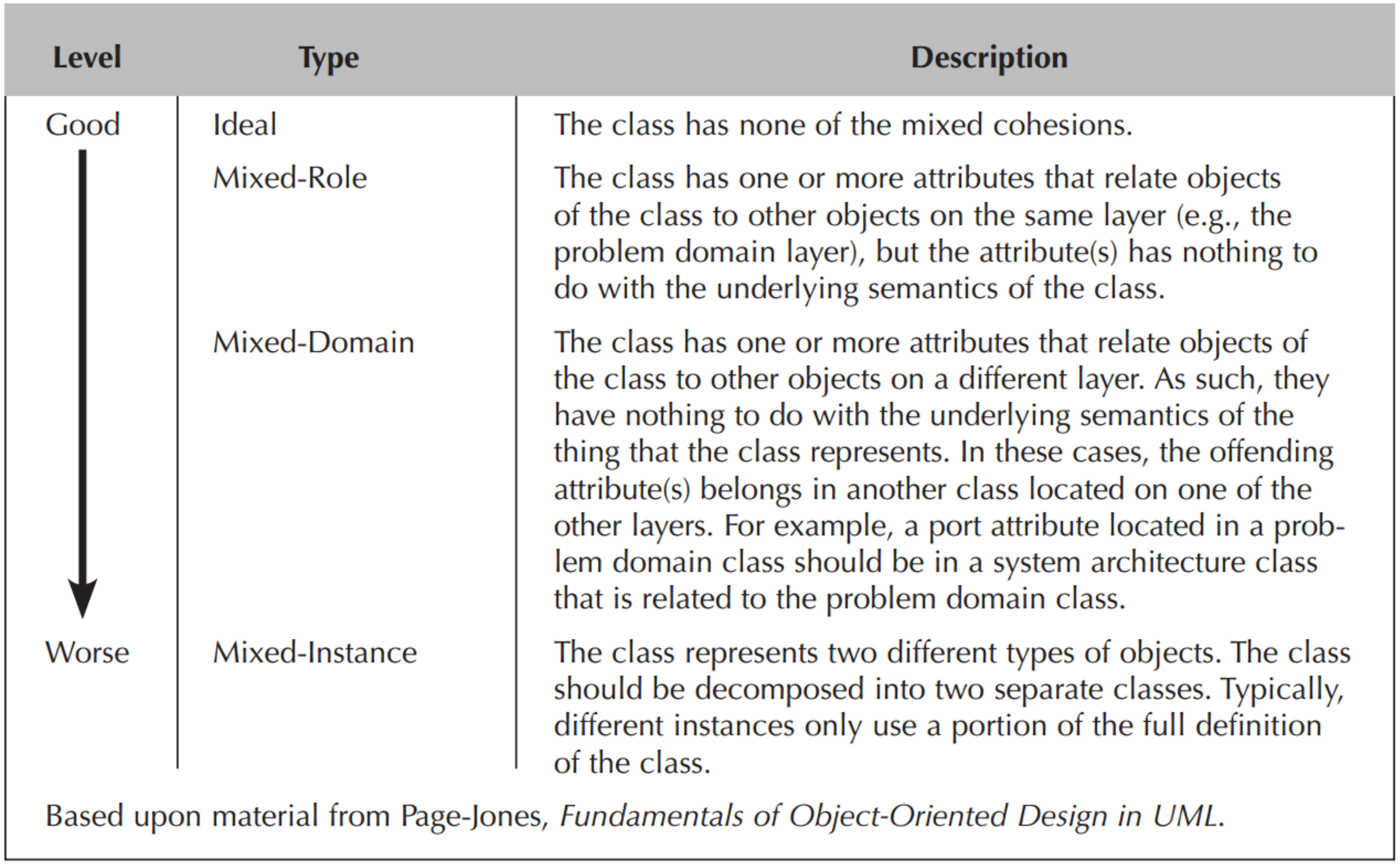


Level	Type	Description
Good	Functional	A method performs a single problem-related task (e.g., calculate current GPA).
↓	Sequential 배타적 순차적	The method combines two functions in which the output from the first one is used as the input to the second one (e.g., format and validate current GPA).
	Communicational	The method combines two functions that use the same attributes to execute (e.g., calculate current and cumulative GPA).
	Procedural 논리적으로 순차적	The method supports multiple weakly related functions. For example, the method could calculate student GPA, print student record, calculate cumulative GPA, and print cumulative GPA.
	Temporal or Classical 초기화 or 삭제	The method supports multiple related functions in time (e.g., initialize all attributes).
	Logical 논리적으로 엮임.	The method supports multiple related functions, but the choice of the specific function is chosen based on a control variable that is passed into the method. For example, the called method could open a checking account, open a savings account, or calculate a loan, depending on the message that is sent by its calling method.
Bad	Coincidental 왜 엮여있는지 모르는 경우	The purpose of the method cannot be defined or it performs multiple functions that are unrelated to one another. For example, the method could update customer records, calculate loan payments, print exception reports, and analyze competitor pricing structure.

Source: These types are based on material from Page-Jones, *The Practical Guide to Structured Systems*; Myers, *Composite/Structured Design*; Edward Yourdon and Larry L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design* (Englewood Cliffs, NJ: Prentice-Hall, 1979).

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

Types of Class Cohesion



Level	Type	Description
Good	Ideal	The class has none of the mixed cohesions.
↓	Mixed-Role	The class has one or more attributes that relate objects of the class to other objects on the same layer (e.g., the problem domain layer), but the attribute(s) has nothing to do with the underlying semantics of the class.
↓	Mixed-Domain	The class has one or more attributes that relate objects of the class to other objects on a different layer. As such, they have nothing to do with the underlying semantics of the thing that the class represents. In these cases, the offending attribute(s) belongs in another class located on one of the other layers. For example, a port attribute located in a problem domain class should be in a system architecture class that is related to the problem domain class.
Worse	Mixed-Instance	The class represents two different types of objects. The class should be decomposed into two separate classes. Typically, different instances only use a portion of the full definition of the class.

Based upon material from Page-Jones, *Fundamentals of Object-Oriented Design in UML*.

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

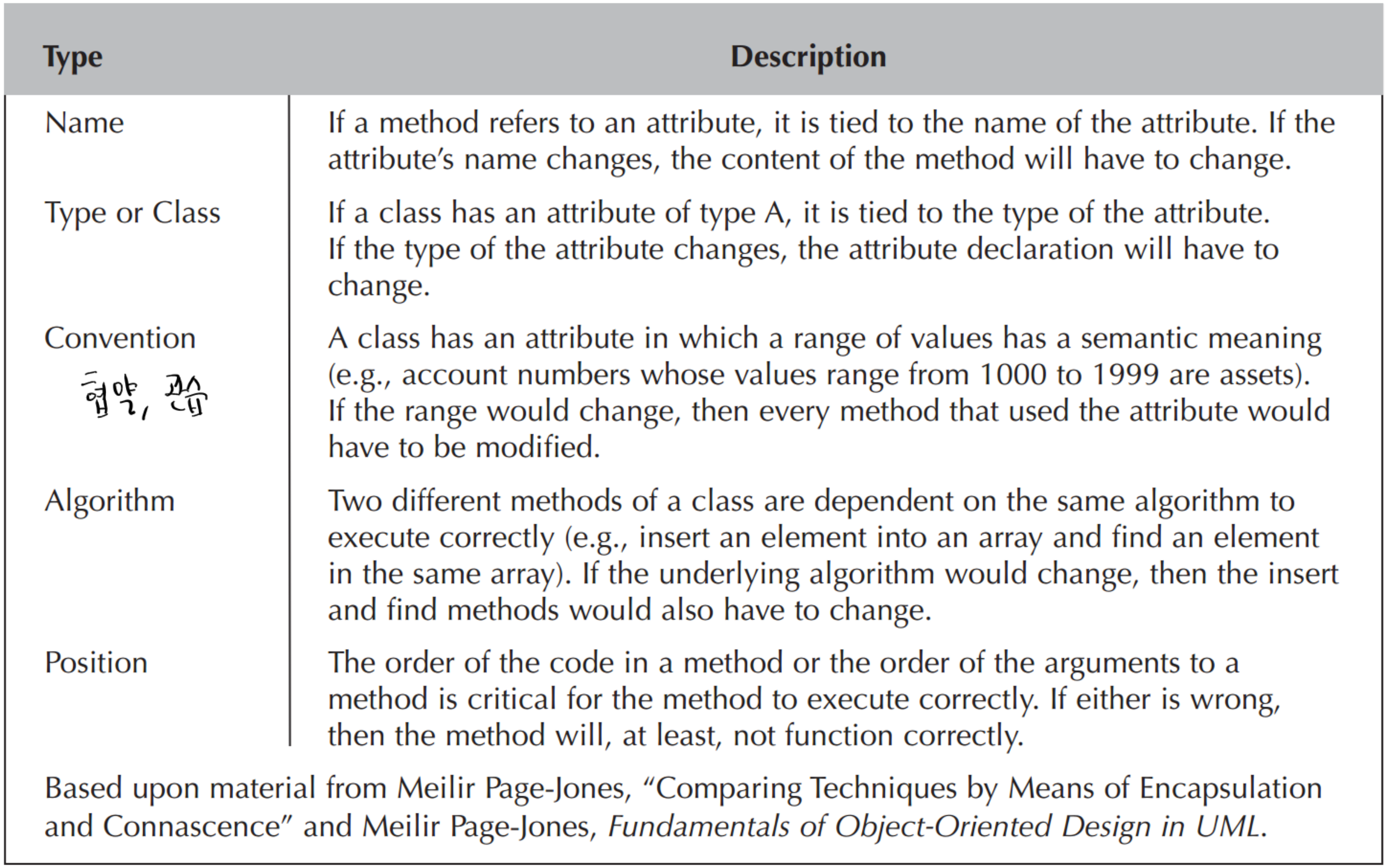
Connascence

- ❖ **Classes are so interdependent that a change in one necessitates a change in the other**

- ❖ **Good programming practice should:**
  - **Minimize overall connascence; however, when combined with encapsulation boundaries, you should:**
    - Minimize across encapsulation boundaries (less interdependence between or among classes)
    - Maximize within encapsulation boundary (greater interdependence within a class)
  - **A sub-class should never directly access any hidden attribute or method of a super class**

Types of Class Connascence

attribute : 속성



Type	Description
Name	If a method refers to an attribute, it is tied to the name of the attribute. If the attribute's name changes, the content of the method will have to change.
Type or Class	If a class has an attribute of type A, it is tied to the type of the attribute. If the type of the attribute changes, the attribute declaration will have to change.
Convention 협약, 관습	A class has an attribute in which a range of values has a semantic meaning (e.g., account numbers whose values range from 1000 to 1999 are assets). If the range would change, then every method that used the attribute would have to be modified.
Algorithm	Two different methods of a class are dependent on the same algorithm to execute correctly (e.g., insert an element into an array and find an element in the same array). If the underlying algorithm would change, then the insert and find methods would also have to change.
Position	The order of the code in a method or the order of the arguments to a method is critical for the method to execute correctly. If either is wrong, then the method will, at least, not function correctly.

Based upon material from Meilir Page-Jones, "Comparing Techniques by Means of Encapsulation and Connascence" and Meilir Page-Jones, *Fundamentals of Object-Oriented Design in UML*.

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

Object Design Activities

Additional specification

Identifying opportunities for reuse

Restructuring the design

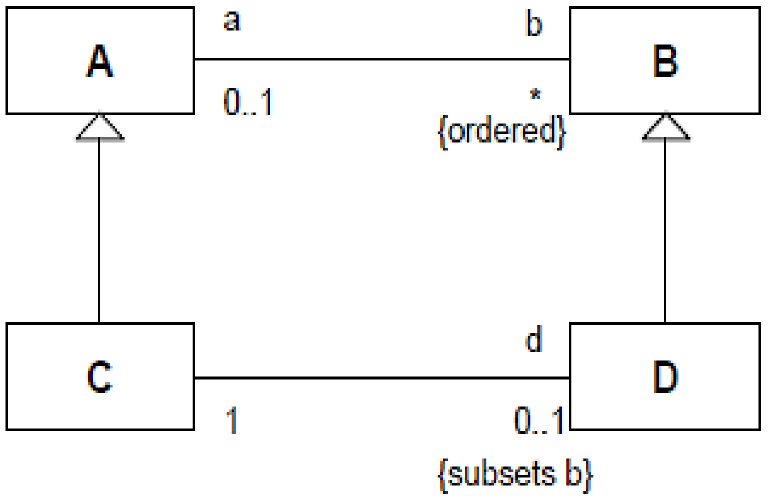
Optimizing the design

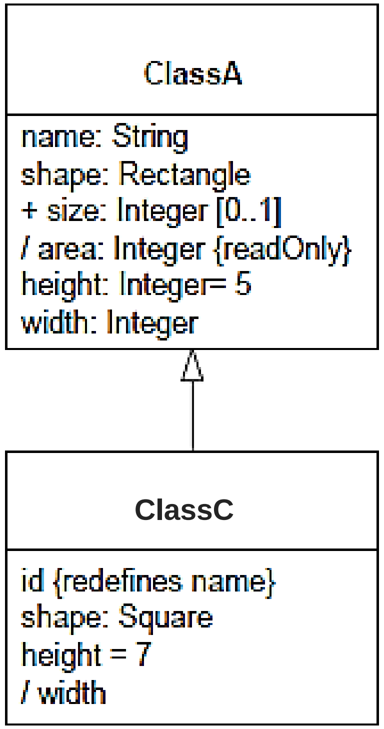
Mapping problem domain classes to implementation language

Additional Specification

❖ **Review the current set of models**

1. Ensure the classes are both necessary and sufficient for the problem
2. Finalize the visibility of the attributes and methods of each class
3. Determine the signature of every method of each class
4. Define constraints to be preserved by objects





Identifying Opportunities for reuse

- Opportunities for Reuse
  - **Design patterns**
    - Useful grouping of collaborating classes that provide a solution to a commonly occurring problem
  - **Components**
    - A self-contained, encapsulated pieces of software that can be plugged into a system to provide a specific set of the required functionalities
  - **Libraries**
    - A set of implemented classes, in general purpose
  - **Frameworks**
    - A set of implemented classes that can be used as a basis for implementing an application (domain-specific application, e.g., Spring framework)
    - containing applied techniques /methods, concept and principles

Restructuring the Design

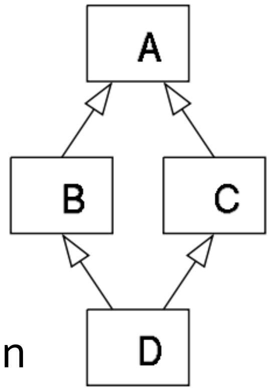
- **Factoring**
  - Separate aspects of a method or class into a new method or class
- **Normalization**
  - Identifies potential classes missing from the design
- **Challenge inheritance relationships to ensure they only support a generalization/specialization semantics**
  - Kind-of Semantics

Optimizing the Design

- To create more efficient design
  - **Review access paths between objects**
    - a message from one object to another may have a long path to traverse
  - **Review each attribute of each class**
    - which methods and which attributes between classes
  - **Review fan-out of each method**
    - the number of message sent by a method
  - **Examine execution order of statements**
    - rearrange some of statements to be more efficient
  - **Create derived activities**
    - to avoid re-computation in several time

Map Domain Classes to Implementation Lang.

- **Single-Inheritance Language**
  - Convert relationships to association relationships
  - Flatten inheritance hierarchy by copying attributes and methods of additional superclass(es)

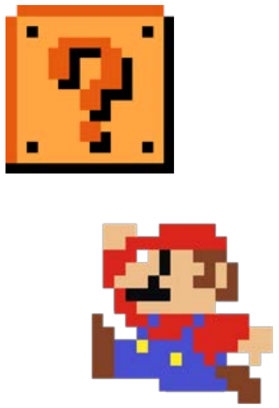


- Object-Based Language
  - Factor out all uses of inheritance from the problem domain class design
- Traditional Language (procedural)
  - Stay away from this !
  - Factor out inheritance from the design
  - Factor out all uses of polymorphism, dynamic binding, encapsulation, etc.

Your Turn – Activity

❖ **Medical appointment system**

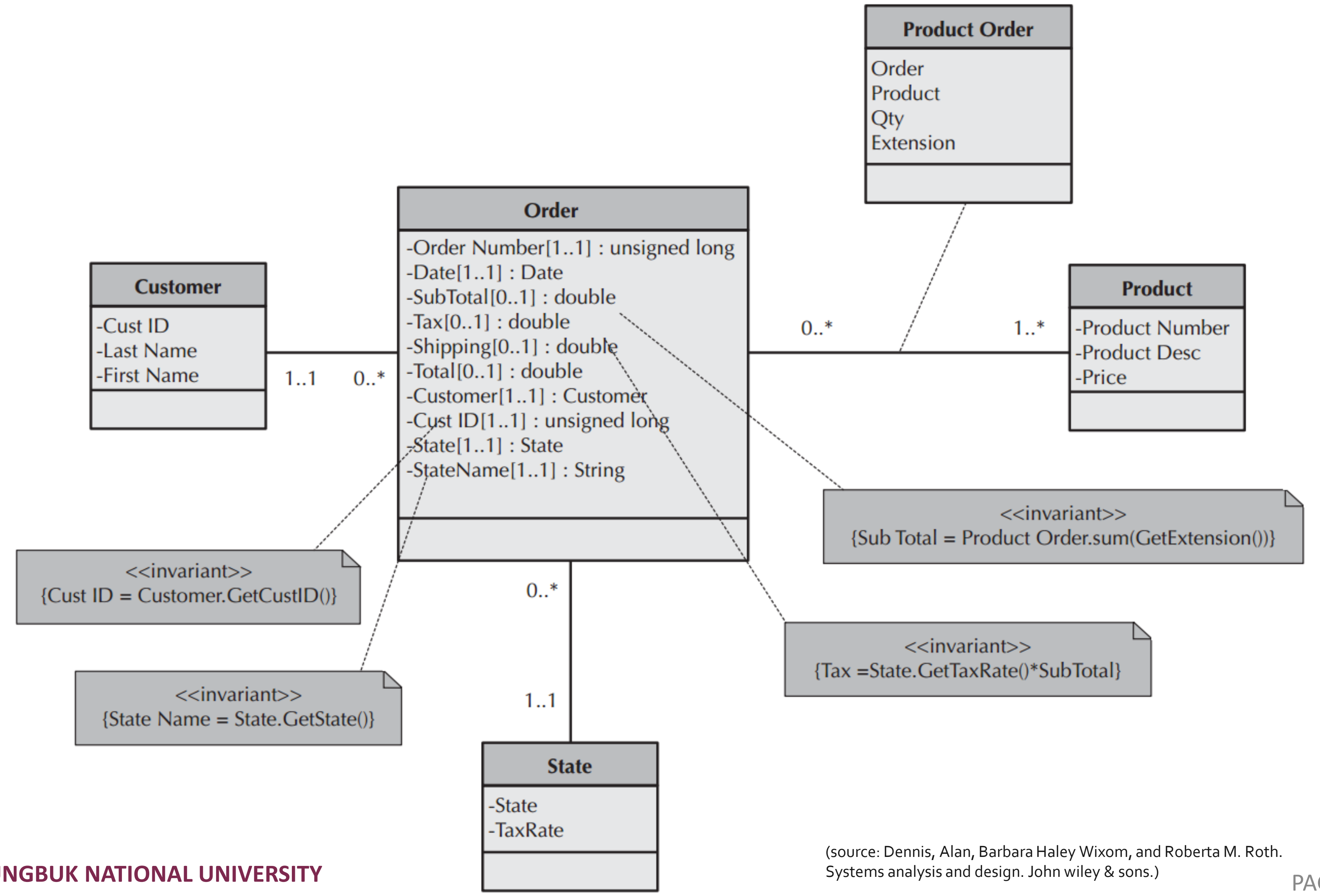
- Assume that you now know that the system must be implemented in Visual Basic 6, which does not support implementation inheritance.
- As such, redraw the class diagram by factoring out the use of inheritance in the design with applying the above rules.



Constraints

- ❖ Constraints
  - **Semantic condition on an element**
- ❖ Types of Constraints
  - **Pre-Conditions**
    - A constraint to be met to allow a method to execute
  - **Post-condition**
    - A constraint to be met after a method executes
  - **Invariants**
    - Constraints that must be true for all instances of a class

Invariants

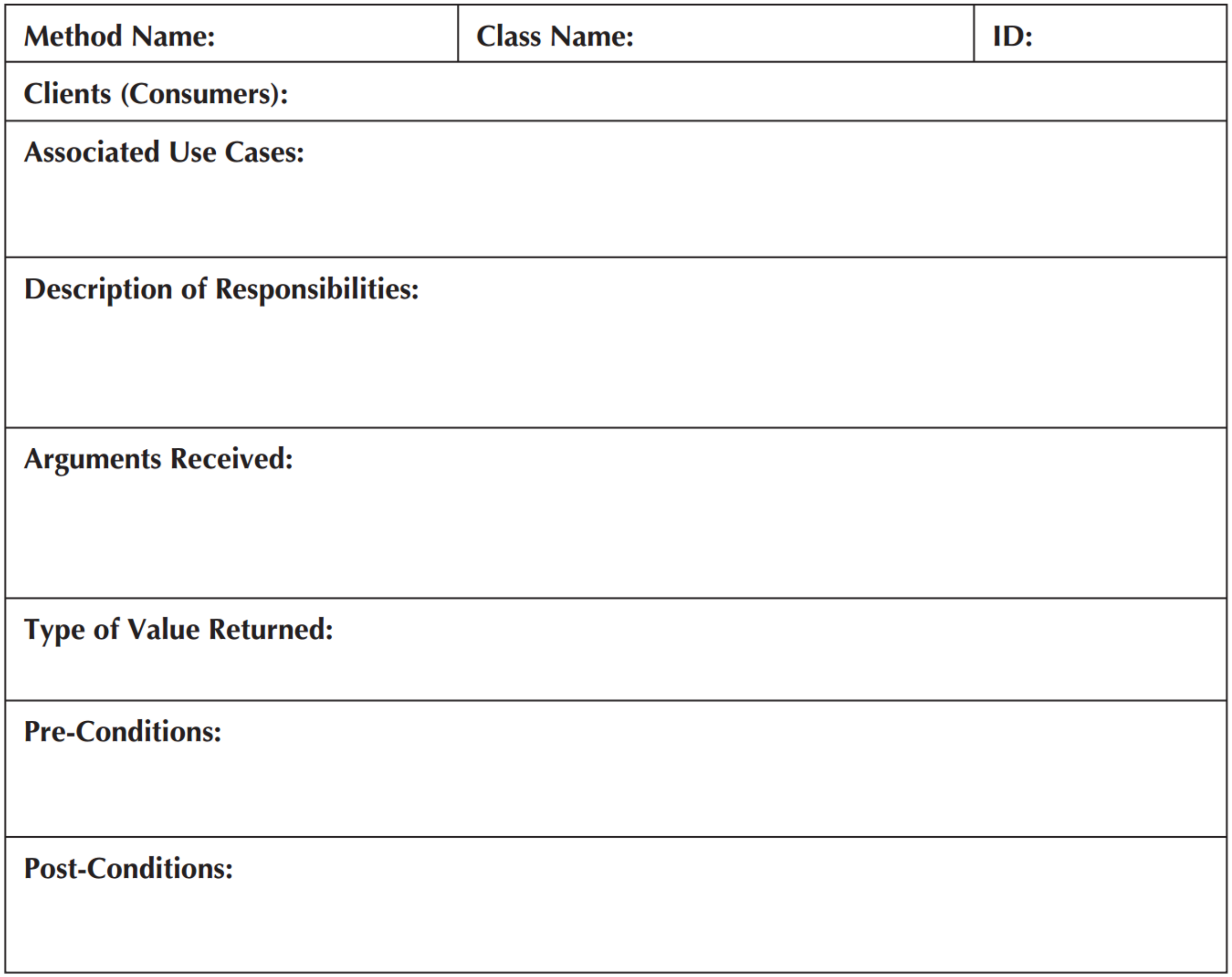


(source: Dennis, Alan, Barbara Haley Wixom, and Roberta M. Roth. Systems analysis and design. John wiley & sons.)

Contracts

- **Document the message passing that take place between objects**
- **One for each interaction**
  - created for each method that can receive message from other objects
- **Composed of the information required for the developer**
  - what message can be sent to the server object and
  - what the client can expect in return
- **Associated with a specific method and a specific class**

Sample Contract Format



Method Name:	**Class Name:**	**ID:**
Clients (Consumers):		
Associated Use Cases:		
Description of Responsibilities:		
Arguments Received:		
Type of Value Returned:		
Pre-Conditions:		
Post-Conditions:		

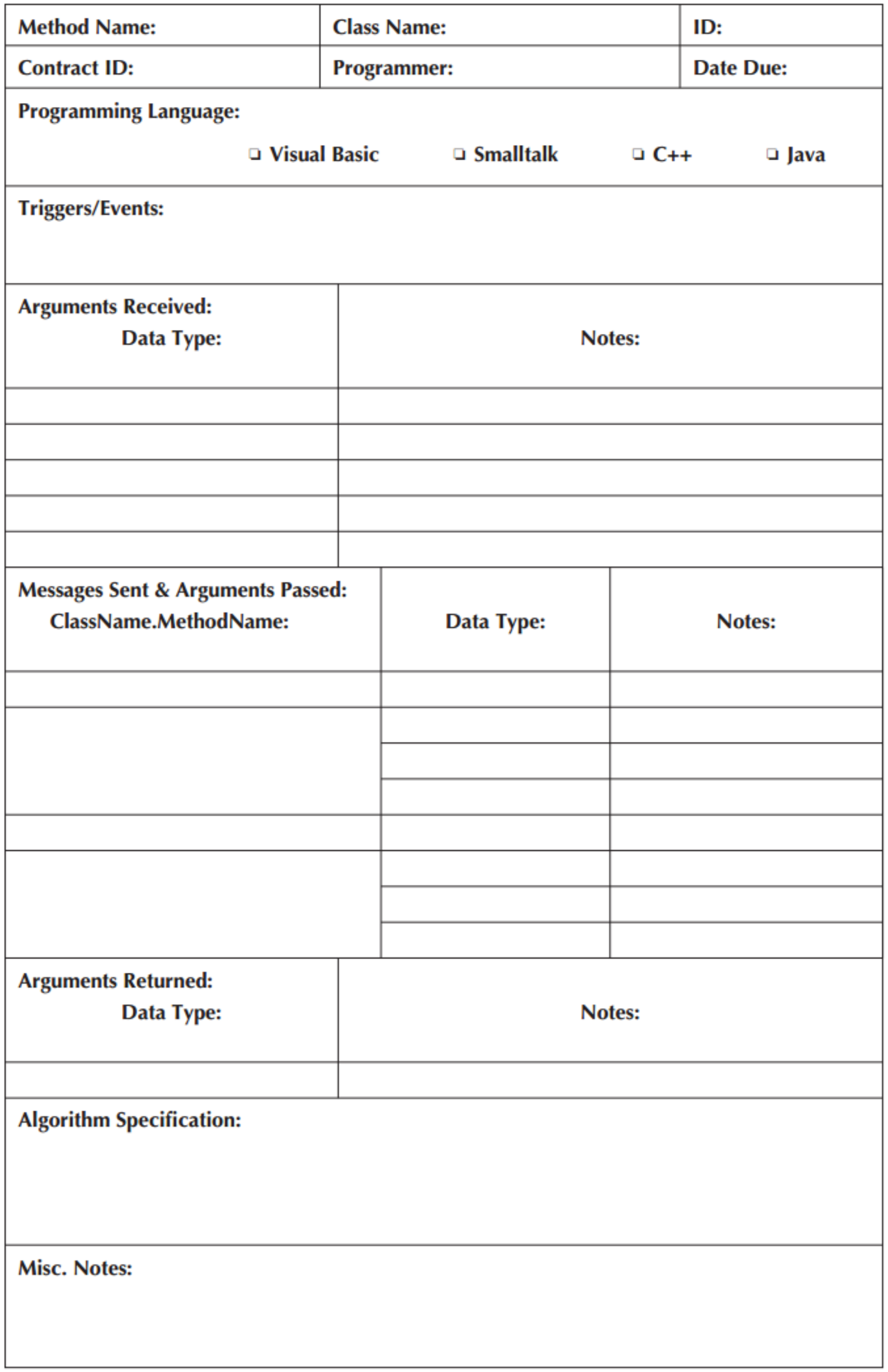
(source: Dennis, Alan, Barbara Haley Wixom, and Roberta M. Roth. Systems analysis and design. John wiley & sons.)

Method Specification

- ❖ **Written document that include explicit instructions on how to write the code to implement the method**

- ❖ **For components for method specification**
  - **General information**
    - method name, ID, Contract ID, Programming Language
  - **Events**
    - list the events that trigger the method
  - **Message passing**
    - what arguments are being passed into, from, and returned by the method
  - **Algorithm specification**
    - Structured English
    - Pseudocode
    - UML activity diagram, etc.

Method Specification Format



Method Name:	Class Name:	ID:
Contract ID:	Programmer:	Date Due:

Programming Language:

❑ Visual Basic ❑ Smalltalk ❑ C++ ❑ Java

Triggers/Events:

Arguments Received: Data Type:	Notes:

Messages Sent & Arguments Passed: ClassName.MethodName:	Data Type:	Notes:

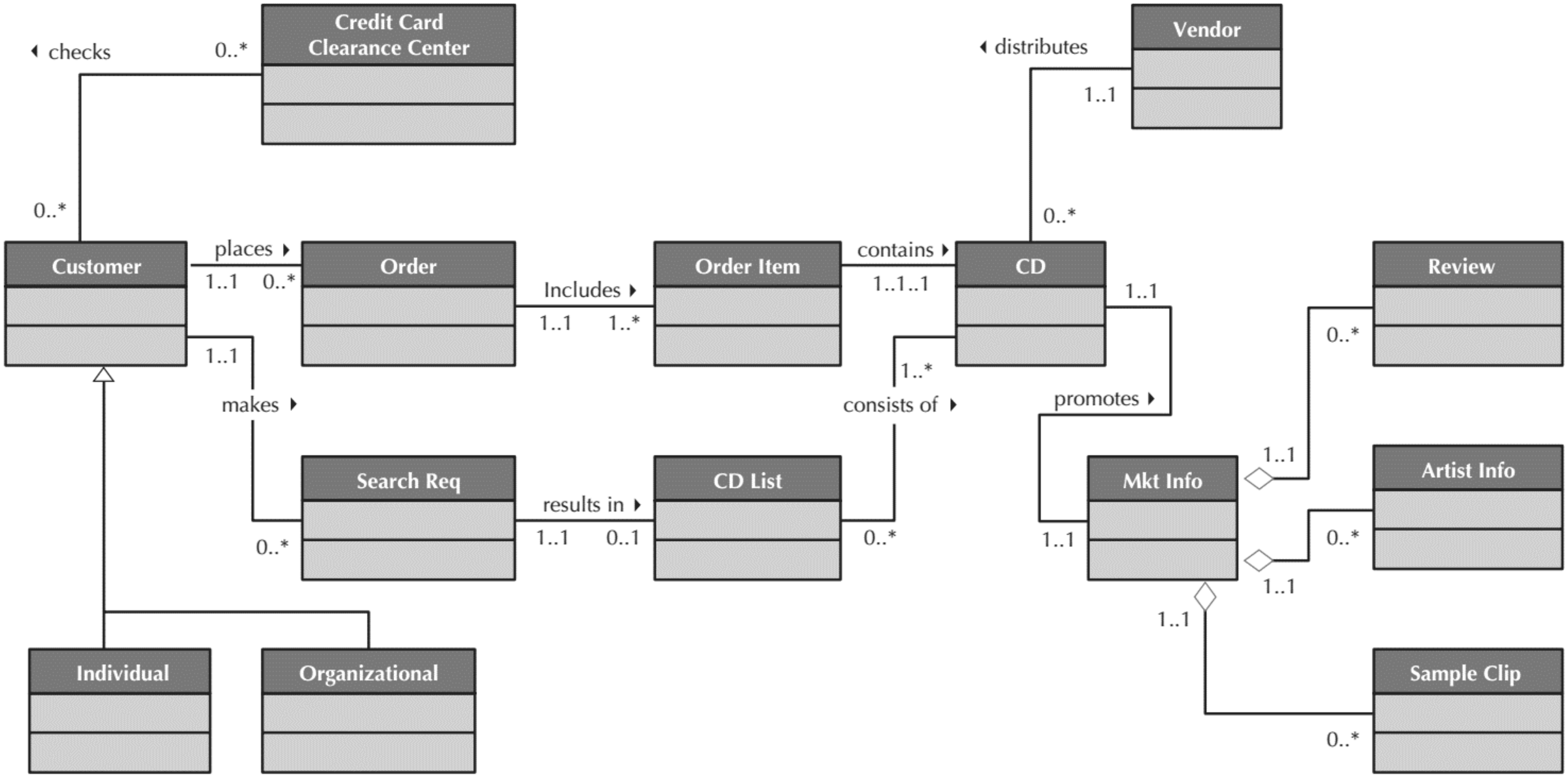
Arguments Returned: Data Type:	Notes:

Algorithm Specification:

Misc. Notes:

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

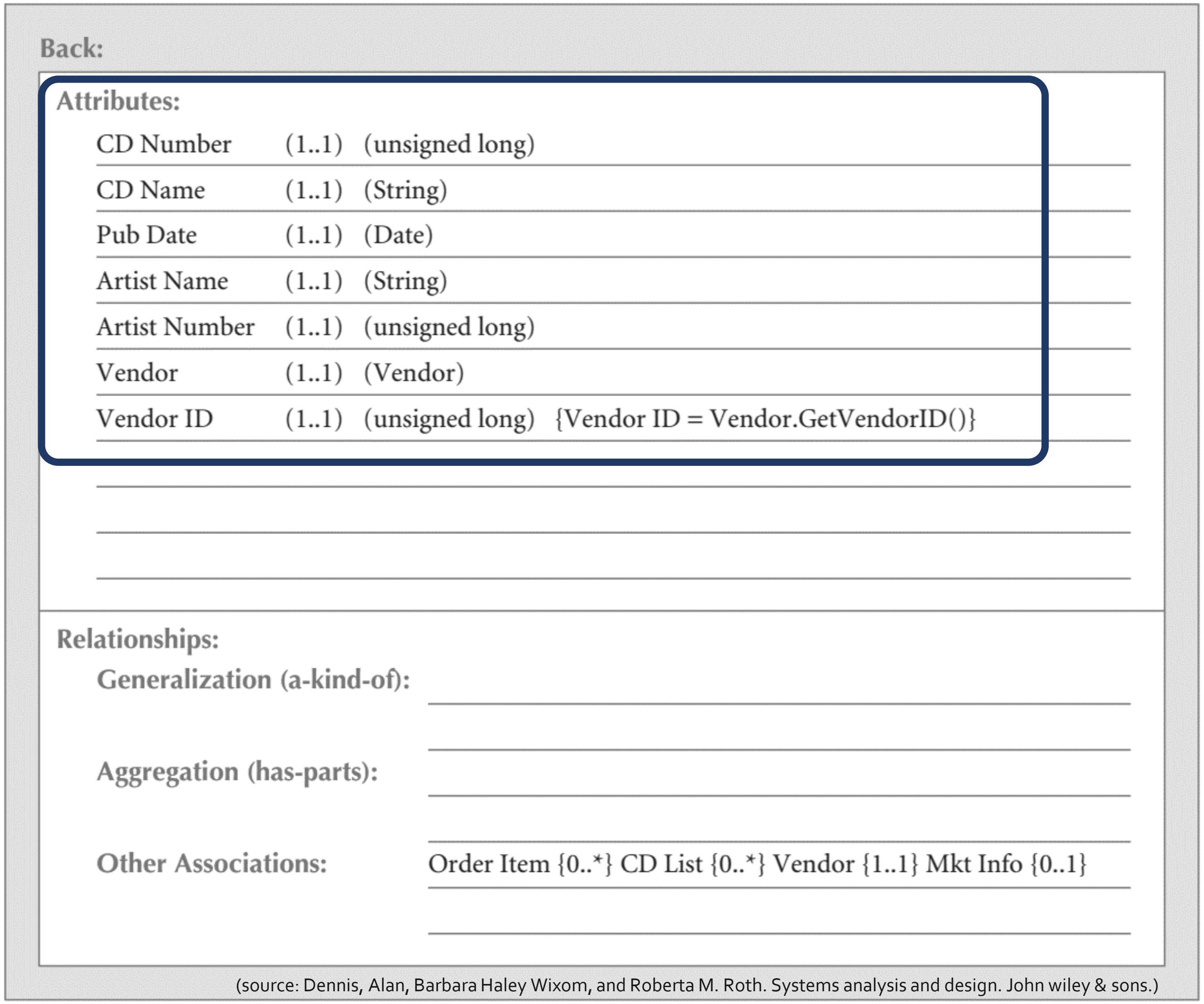
Case Study : CD Selection Company



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

Case Study : CD Selection Company

Back of CRC Card



Back:

Attributes:

CD Number	(1..1)	(unsigned long)	
CD Name	(1..1)	(String)	
Pub Date	(1..1)	(Date)	
Artist Name	(1..1)	(String)	
Artist Number	(1..1)	(unsigned long)	
Vendor	(1..1)	(Vendor)	
Vendor ID	(1..1)	(unsigned long)	{Vendor ID = Vendor.GetVendorID()}

Relationships:

Generalization (a-kind-of):

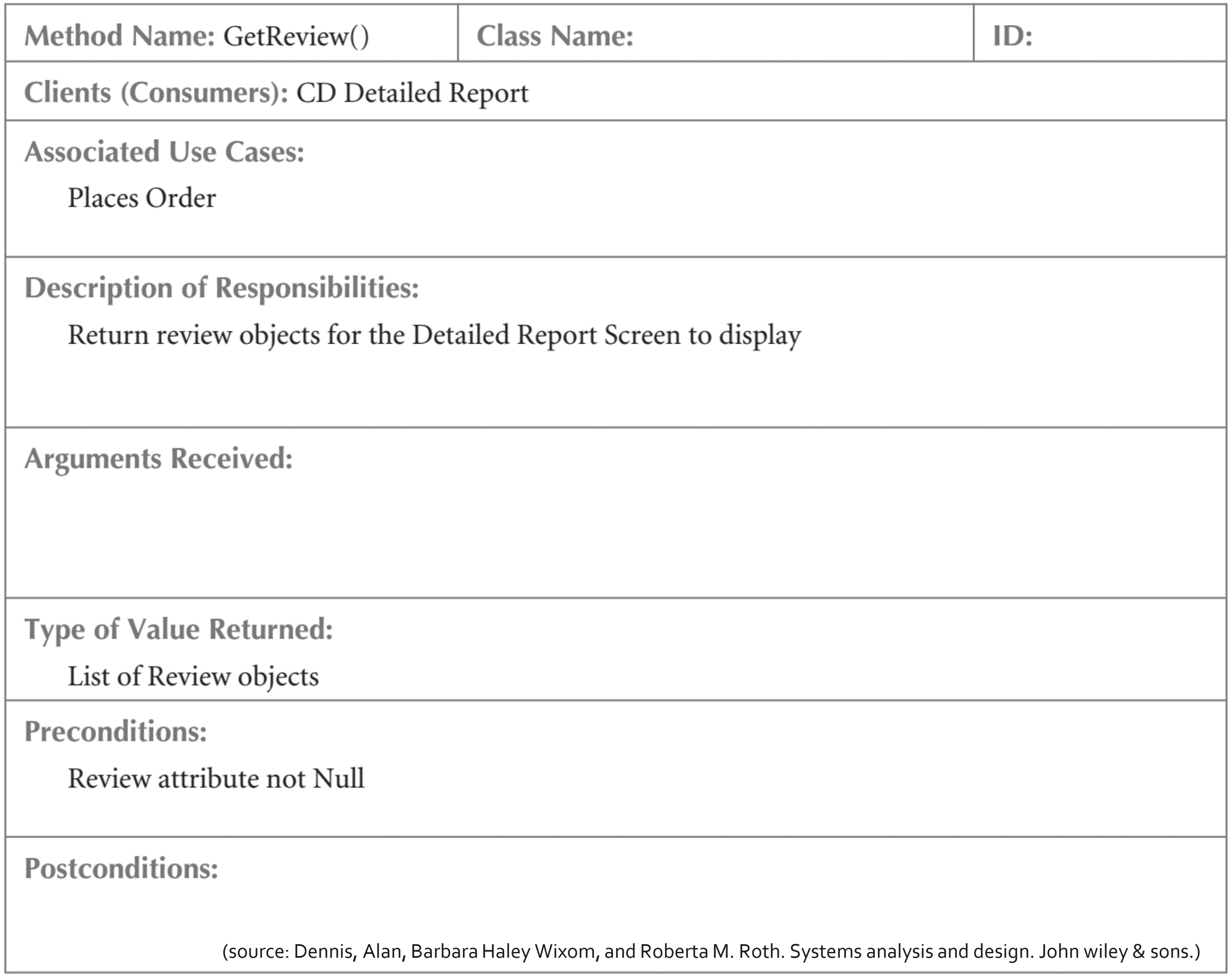
Aggregation (has-parts):

Other Associations: Order Item {0..*} CD List {0..*} Vendor {1..1} Mkt Info {0..1}

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

Case Study : CD Selection Company

Contract of the method “GetReview( )”

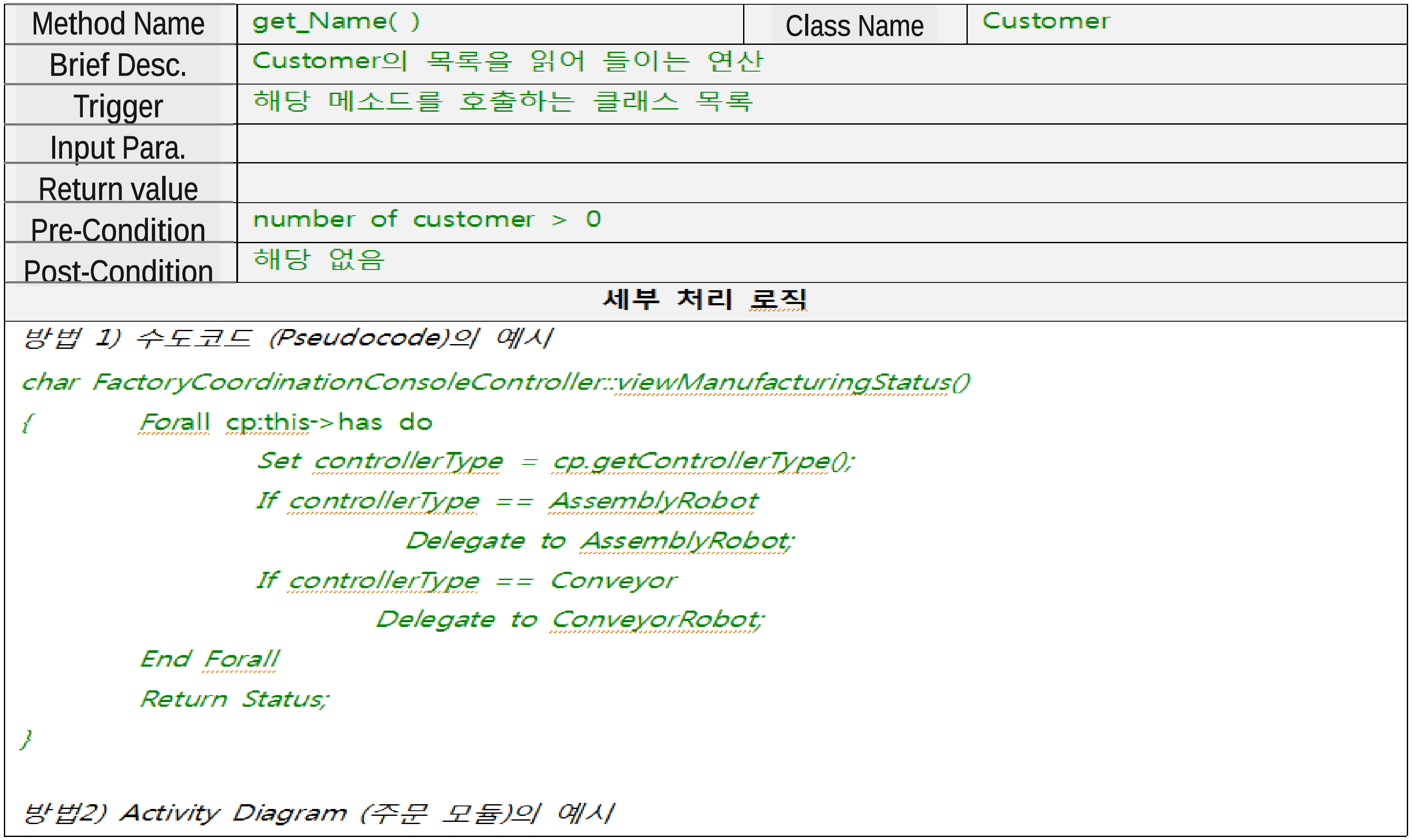


Method Name: GetReview()	**Class Name:**	**ID:**
Clients (Consumers): CD Detailed Report		
Associated Use Cases: Places Order		
Description of Responsibilities: Return review objects for the Detailed Report Screen to display		
Arguments Received:		
Type of Value Returned: List of Review objects		
Preconditions: Review attribute not Null		
Postconditions:		

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

Case Study : CD Selection Company

Method Specification



Method Name	get_Name( )	Class Name	Customer
Brief Desc.	Customer의 목록을 읽어 들이는 연산		
Trigger	해당 메소드를 호출하는 클래스 목록		
Input Para.			
Return value			
Pre-Condition	number of customer > 0		
Post-Condition	해당 없음		

세부 처리 로직

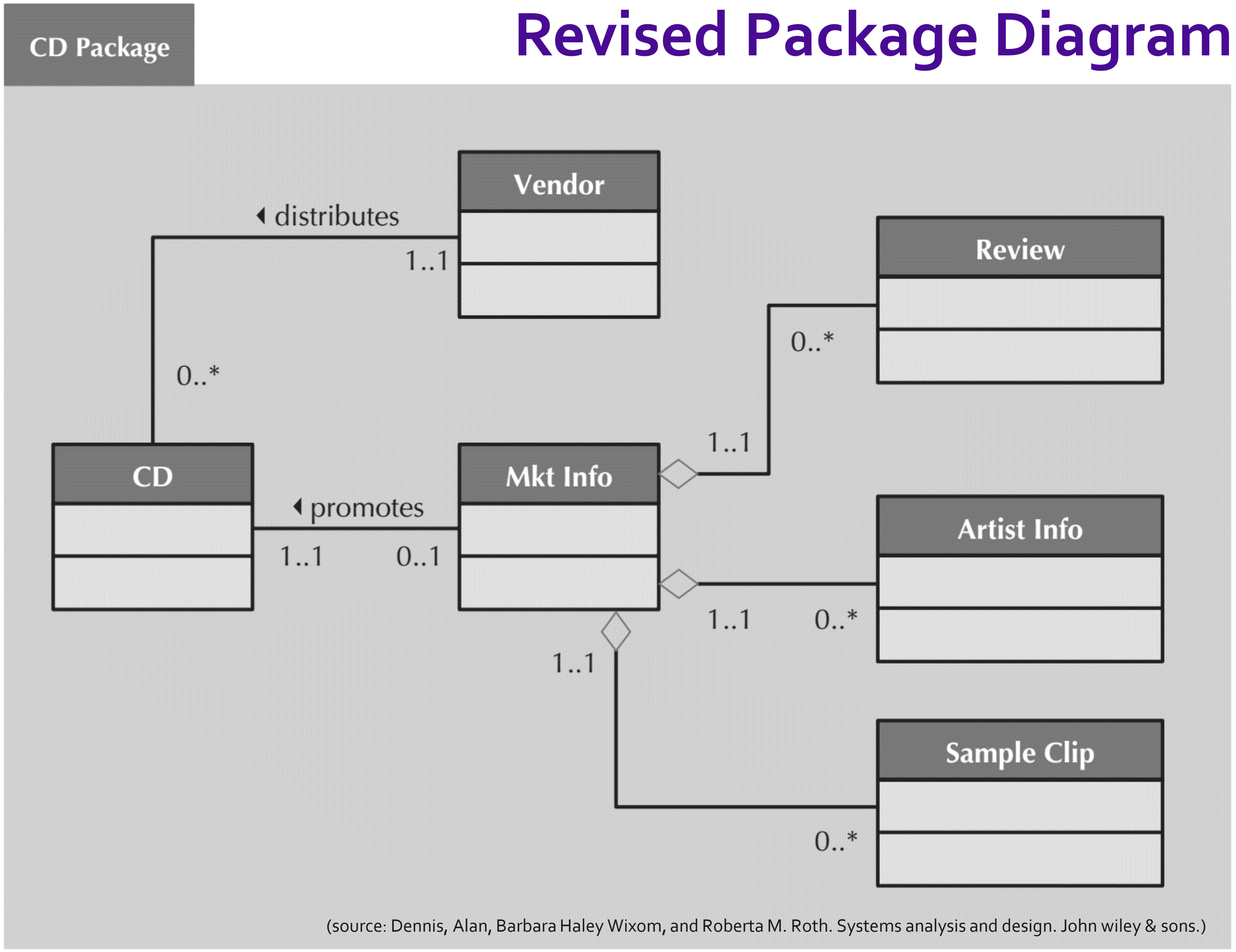
방법 1) 수도코드 (Pseudocode)의 예시

```
char FactoryCoordinationConsoleController::viewManufacturingStatus()
{       Forall cp:this->has do
                Set controllerType = cp.getControllerType();
                If controllerType == AssemblyRobot
                        Delegate to AssemblyRobot;
                If controllerType == Conveyor
                        Delegate to ConveyorRobot;
        End Forall
        Return Status;
}
```

방법2) Activity Diagram (주문 모듈)의 예시

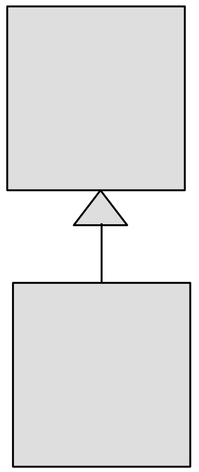
Case Study : CD Selection Company

Revised Package Diagram



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

Summary and Discussion



- ❖ **Design Criteria : Coupling and Cohesion**
- ❖ **In Design Activities,**
  - Additional Specification
  - Opportunities for reuse
  - Restructuring / Optimizing and Mapping Implementation Lang.
  - Constraints and Contracts
  - Method Specification
- ❖ **What is an inheritance conflict ? How does an inheritance conflict affect the design ?**
- ❖ **Explain the difference of overriding and overloading ?**

  Write an example code in C++ or Java for overriding

  and Overloading ?

```
Class Polygon{
   int getArea();
}
Class Circle {
   int getArea();

}

Polygon po;
Circle ci;

Figure = &po;
Figure -> getArea();
Figure = &ci;
Figure -> getArea();
```