

# UML 2.x와 모델링 도구의 활용



# 목차

---

1. UML 개요
2. 구조 다이어그램
3. 상호작용 다이어그램
4. 행위 다이어그램



## 1. UML 개요

---

- UML 정의
- UML의 3가지 목적
- UML 2.x 다이어그램



# 시작에 앞서...

## ✓ 모델링 도구(EA) 다운로드 및 설치

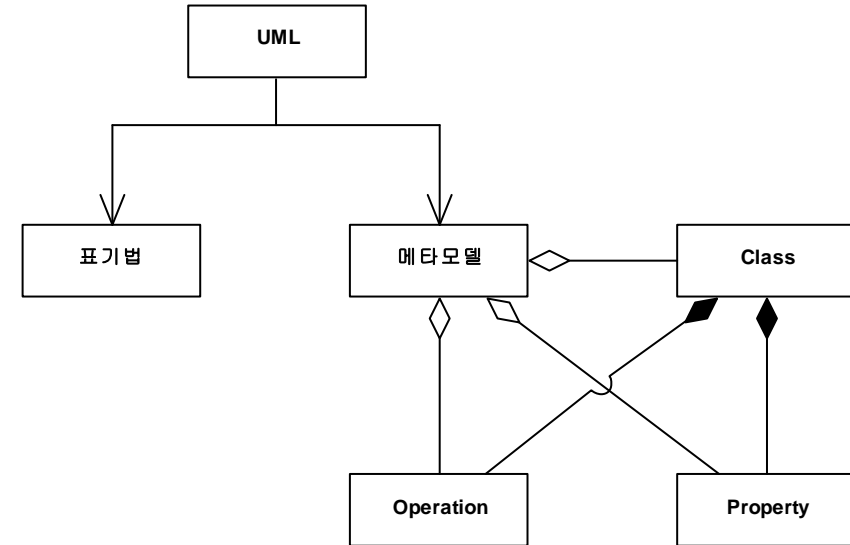
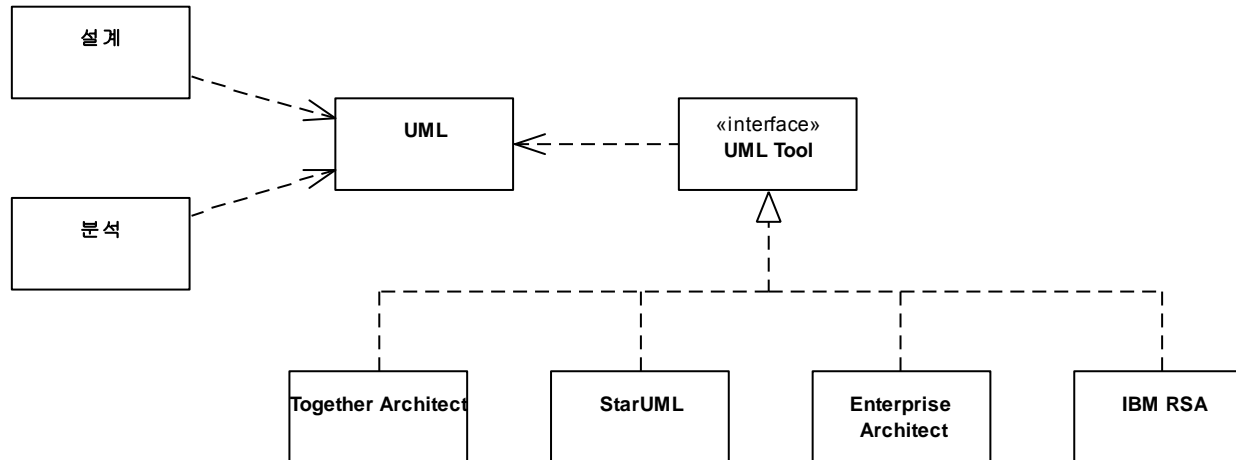
- URL : <http://www.sparxsystems.com/>
- Enterprise Architect Trial



# UML 정의

## ✓ Unified Modeling Language

- 통합된 모델링 언어
- 모델링을 위한 표준 표기법
  - 객체지향 분석/설계나 방법론이 아닌 단지 표기법
- 표기법을 이용하여 모델을 메타모델로 표현
  - 표기법 : 도식적인 표현을 의미
  - 메타모델은 언어를 표현하기 위한 상위의 언어
  - 모델의 필수 요소와 문법, 구조를 정의
  - 다이어그램은 메타모델을 표현한 것



# UML의 3가지 목적

---

## ✓ 스케치(sketch)

- 시스템의 일부 측면에 대한 설명
- 의사소통(communication)에 초점
- 단순한 도구(tool)를 사용

## ✓ 청사진(blueprint)

- 완전성(completeness)에 초점
- 코딩을 위한 설계의 세부적인 결정이 끝난 상태
- 정교한(sophisticated) 도구(tool)를 사용

## ✓ 프로그래밍 언어(programming language)

- UML 다이어그램이 실행 가능한 코드로 컴파일
- UML = 소스코드
- 매우 정교한 도구가 필요

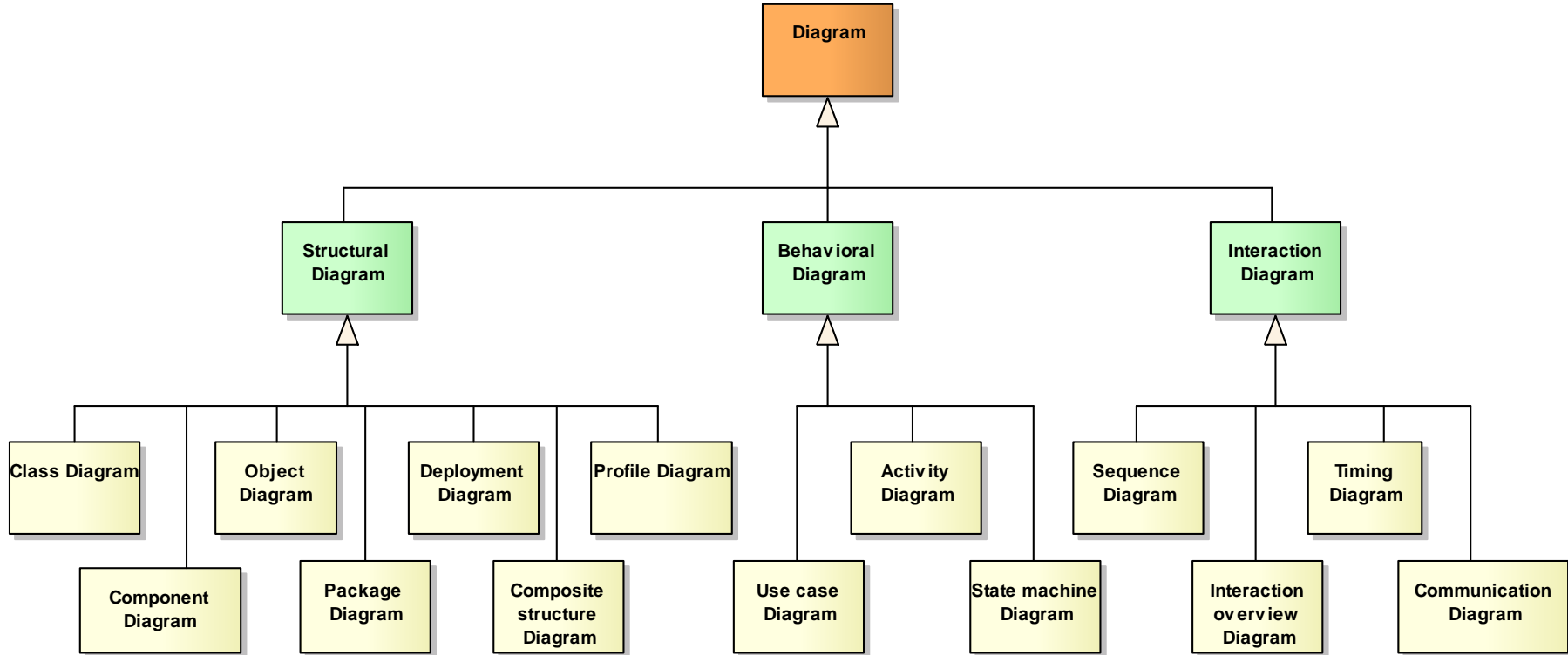
# UML 2.x 다이어그램

✓ UML 2.x 의 14가지 다이어그램

분류	다이어그램	목적	버전
구조 다이어그램 (Structural Diagram)	Class	클래스, 특징, 그리고 관계	UML 1
	Object	인스턴스와 그들간의 관계	(비공식) UML 1
	Composite structure	실행시간에서의 내부 구조	UML 2
	Deployment	HW 플랫폼, SW산출물, 시스템의 런타임 아키텍처를 보여줌	UML 1
	Component	컴포넌트의 구조와 연결	UML 1
	Package	모델 요소를 구성하고 그들간의 종속 관계를 보여줌	(비공식) UML 1
	Profile	UML의 확장메커니즘을 설명	UML 2
행위 다이어그램 (Behavioral Diagram)	Activity	행위의 데이터 흐름 및 제어 흐름을 보여줌	UML 1
	Use case	사용자와 시스템간의 상호 작용하는 방법	UML 1
	State machine	이벤트에 의한 객체의 생명주기 동안의 상태변화	UML 1
상호작용 다이어그램 (Interaction Diagram)	Interaction overview	액티비티 다이어그램과 시퀀스 다이어그램의 혼합	UML 2
	Sequence	흐름에 중점을 둔 객체간의 상호작용	UML 1
	Communication	링크에 중점을 둔 객체간의 상호작용	UML 1
	Timing	타이밍에 중점을 둔 객체간의 상호작용	UML 2

# UML 2.x 다이어그램

✓ UML 2.x 의 14가지 다이어그램







## 2. 구조 다이어그램

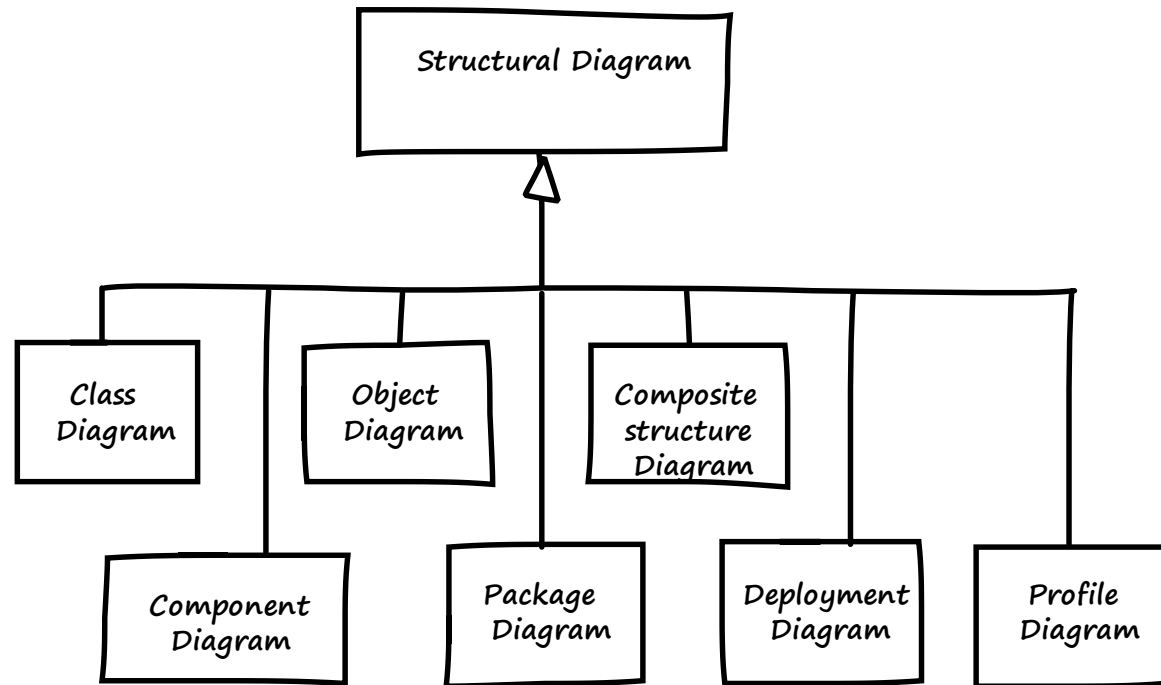
---

- 구조 다이어그램 개요
- 클래스 다이어그램

# 구조 다이어그램 개요

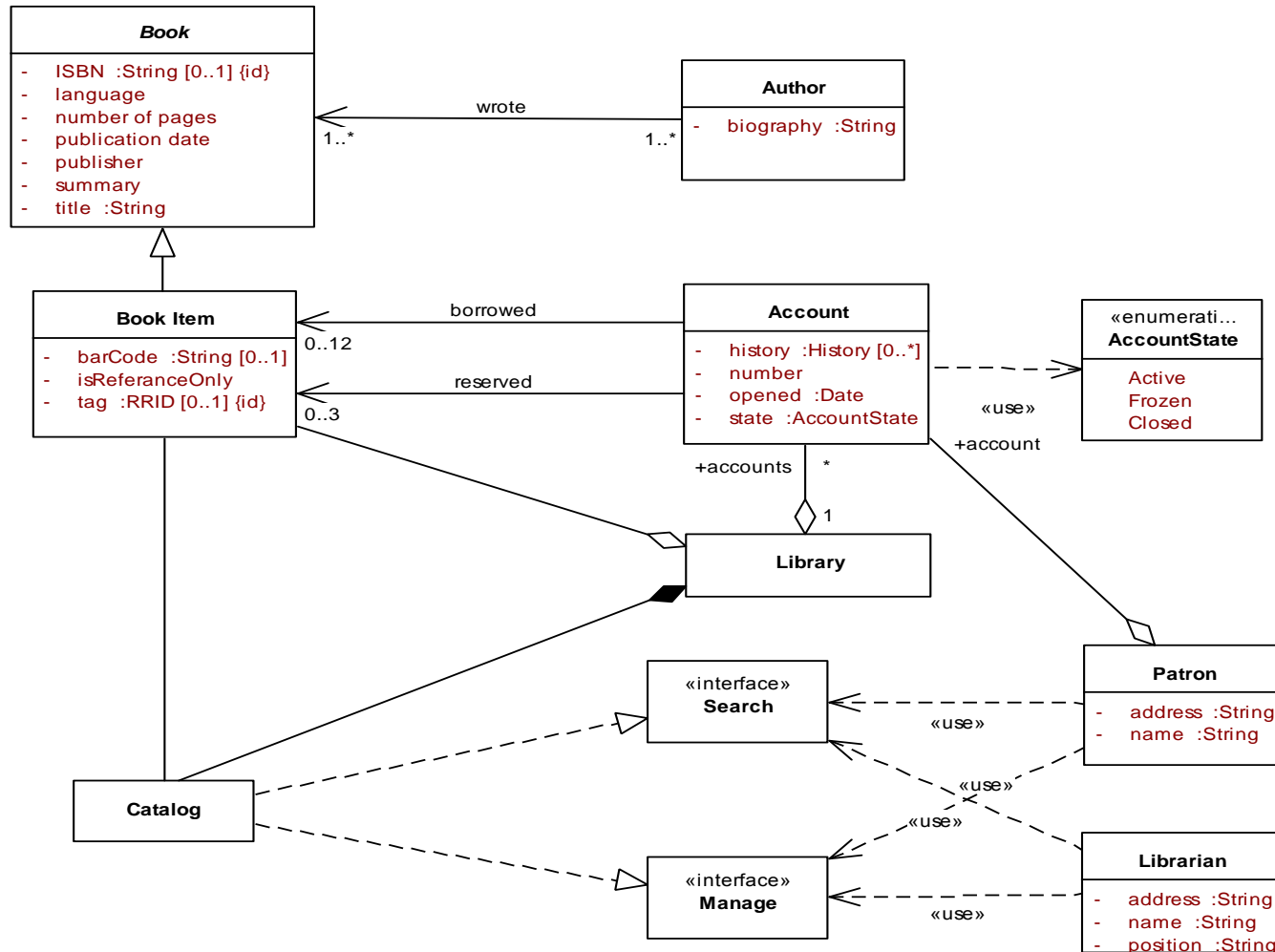
## ✓ 구조 다이어그램(Structural Diagram)

- 시스템의 구조적인 측면을 표현하기 위해 사용하며, 시스템을 구성하는 요소 및 그들간의 관계를 표현한다.
- UML 2.x 의 구조 다이어그램
  - Class diagram
  - Object diagram
  - Composite structure diagram
  - Deployment diagram
  - Component diagram
  - Package diagram
  - Profile diagram



# 클래스 다이어그램 *Class Diagram*

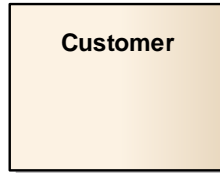
✓ 클래스 다이어그램은 클래스와 인터페이스의 특성 및 제약조건, 관계(연관,상속,의존 등)를 표현한다.



# 클래스 Class [1/3]

## ✓ 클래스 표기법

- 클래스의 이름, 속성, 오퍼레이션을 나타내는 세 개의 구획으로 나뉨
- 클래스명을 제외한 나머지는 선택적 요소이므로 상세화 수준에 따라 생략 가능함



클래스의 이름만 표현

```
class Customer {  
    int age;  
    String name;  
    // ...  
}
```

## ✓ 분석 단계의 상세화

- 분석 수준의 속성과 오퍼레이션을 추가로 표현함

SearchService
- engine :SearchEngine - query :SearchResult
+ search()

속성, 오퍼레이션까지 표현

```
class SearchService {  
    SearchEngine engine;  
    SearchRequest request;  
  
    SearchResult search(SearchRequest query) {  
        //TODO  
        return null;  
    }  
}
```



## ✓ 구현 단계의 상세화

- Static operation과 같이 구현 단계에서 볼 수 있는 상세한 정보까지 표현함

SearchService
- config :Configuration - engine :SearchEngine
+ createEngine() :SearchEngine + search(SearchRequest) :SearchResult

리턴타입, 파라미터, 정적 오퍼레이션 등 표현

```
class SearchService {  
    SearchEngine engine;  
    Configuration config;  
  
    private static SearchEngine createEngine() {  
        //TODO  
        return null;  
    }  
  
    SearchResult search(SearchRequest query) {  
        //TODO  
        return null;  
    }  
}
```

# 추상 클래스와 인터페이스 *Abstract Class & Interface*

## ✓ 추상 클래스 표기법

- 추상 클래스는 클래스와 동일하게 사각형으로 표현하고 클래스의 이름만 이탤릭체로 표현함



*SearchRequest*

```
abstract class SearchRequest {  
  
}
```

## ✓ 인터페이스 표기법

- 인터페이스는 사각형 안에 «interface» 키워드를 이름 앞에 표현함



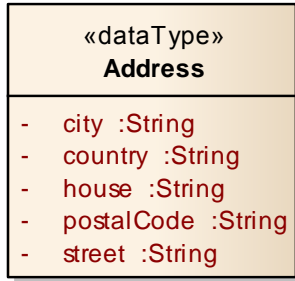
«interface»  
SearchService

```
interface SearchService {  
  
}
```

# 데이터 타입 *Data Type*

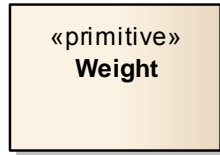
## ✓ 데이터 타입 표기법 (키워드사용)

- 데이터 타입은 구조화된 데이터 타입의 모델링을 위하여 attribute와 operation을 포함할 수 있다.



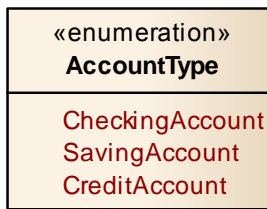
```
class Address {  
    String house;  
    String street;  
    String city;  
    String country;  
    String postalCode;  
}
```

- 원시 타입은 원자의 데이터 값을 표현한 데이터 타입이다.



```
Integer weight;
```

- Enumeration 타입은 사용자 정의 열거 literal 데이터 타입이다.

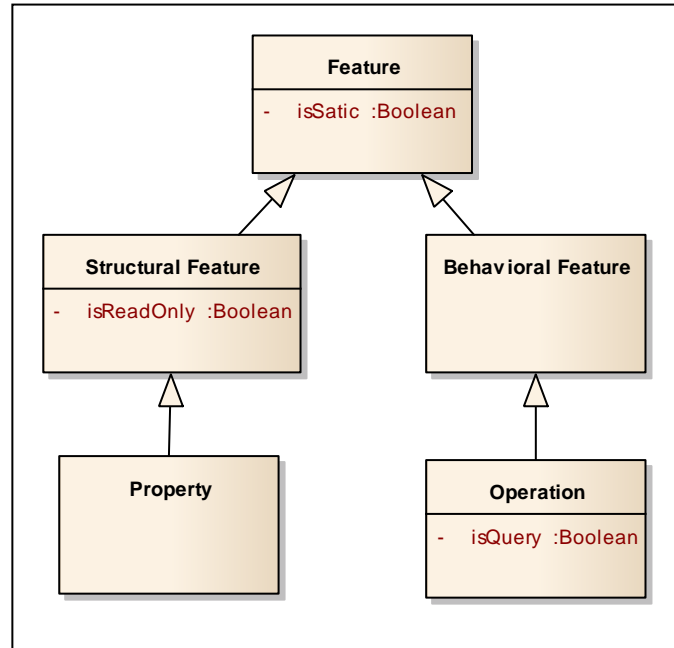


```
enum AccountType {  
    CheckingAccount, SavingAccount, CreditAccount;  
}
```

# 클래스의 특성 *Feature*

## ✓ 클래스를 구성하는 특성 (Feature)

- 구조적인 특성 : 속성
- 행위적인 특성 : 오퍼레이션



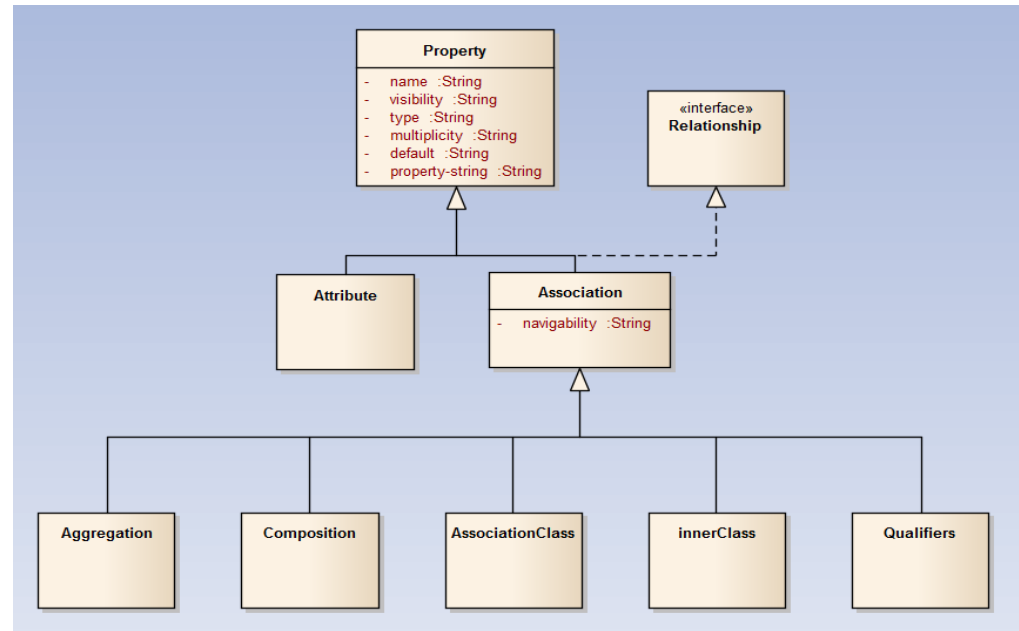
# 프로퍼티 Property [1/2]

## ✓ 개요

- 구조적인 Feature를 표현, 클래스의 구조적인 측면을 표현
- 속성(attribute)이나 연관(association)으로 표현

## ✓ 속성(Attributes)

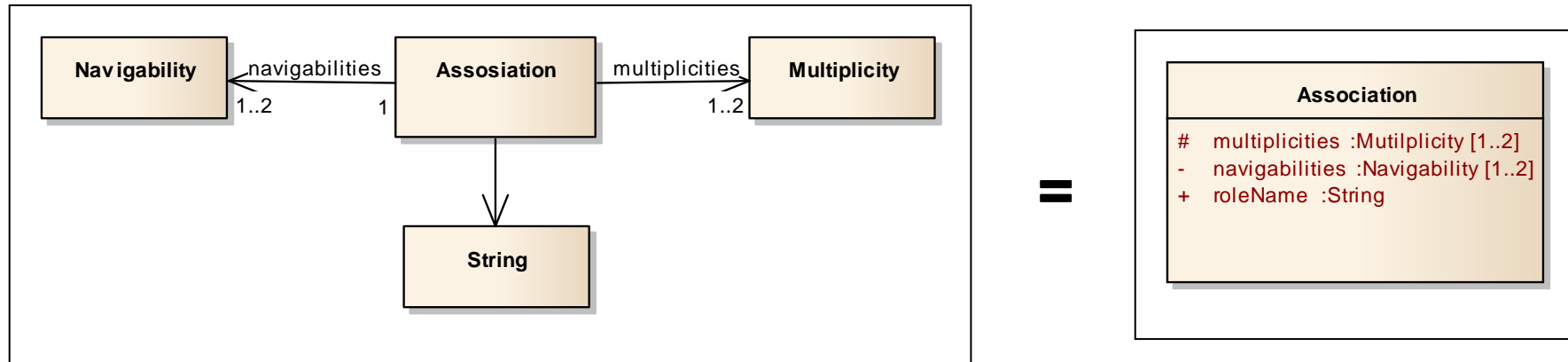
- 클래스 내부에 정의되어 프로퍼티를 설명
- 일반적으로 값 객체 또는 다른 객체와 관계가 없는 경우 사용
- 문법 : **visibility name : type multiplicity = default {property-string}**  
(예) - name : String[1] = "Untitled" {readonly}
  - visibility : 가시성
  - name : 속성의 이름
  - type : 속성의 종류
  - multiplicity : 개수
  - default : 초기 값
  - {property-string} : 추가 특성





### ✓ 연관(Associations)

- 일반적으로 참조 객체인 경우
- Role Name : 연관의 형태를 명확히 하기 위해 역할 이름을 가짐
- Navigability : 화살표를 가리키는 방향에 있는 클래스를 탐색하거나 질의할 수 있음
- Multiplicity : 하나의 인스턴스에 참여하는 다른 쪽의 인스턴스의 개수 표현



# 파생 프로퍼티 *Derived Property*

## ✓ 개요

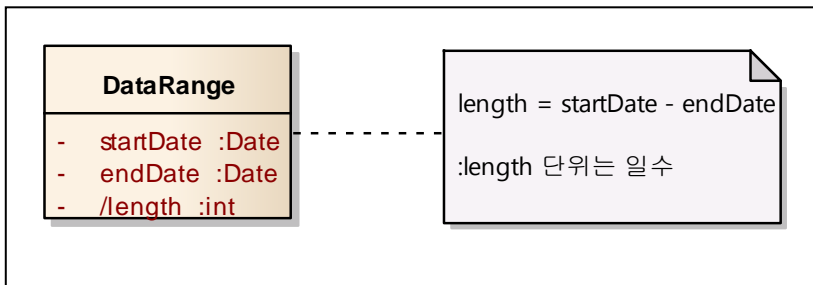
- 다른 값에 의해 계산 가능한 프로퍼티

## ✓ 두 가지 해석 방법

- 계산된 값과 저장된 값을 구분하기 위해서 사용됨
  - start와 end는 저장된 값(stored value)
  - length는 계산된 값(calculated value)
  - 일반적인 용도지만 Date Range 내부 구조의 지나친 노출
- 값들 사이의 제약을 가리키는 것으로 사용됨
  - 3개의 값 사이에 제약사항이 존재함을 표현
  - 3개의 값 중 어떤 것이 파생인지는 중요하지 않음

## ✓ 표기법

- 연관 이름에 '/' 를 추가



파생 프로퍼티에 접근할 때 마다 계산되어지기 때문에 효율성이 떨어질 수 있다

- ✓ 개요
  - 클래스(또는 인터페이스)의 속성과 오퍼레이션을 들여다 볼 수 있는 범위
- ✓ 표기법

표기법	의미
+	public
-	private
~	package
#	protected

SQLStatement
- clearWarnings() :void
+ executeQuery(String) :ResultSet
~ getQueryTimeout() :int
# isPoolable() :Boolean

# 개수 Multiplicity

✓ 개요

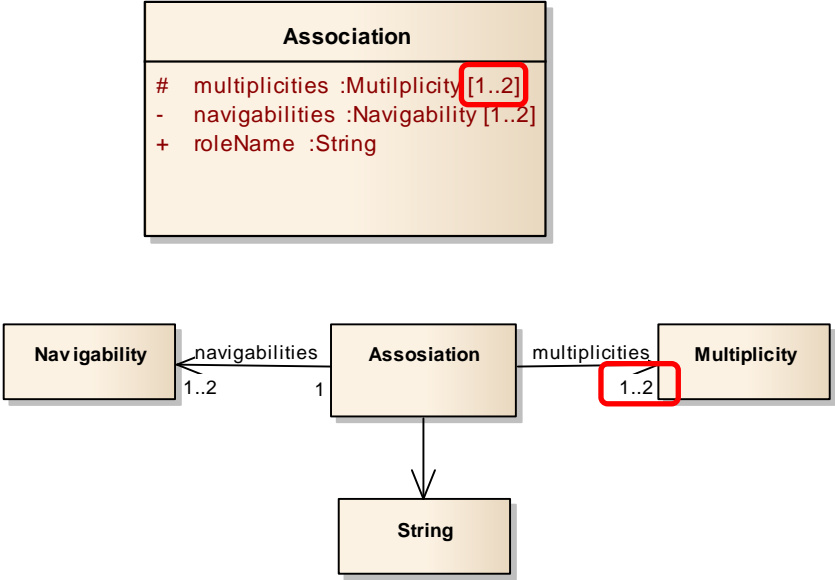
- 연관있는 두 클래스 사이에서 한 클래스의 객체와 관계를 가질 수 있는 다른 클래스의 객체 개수

✓ 속성에 대한 개수 표현

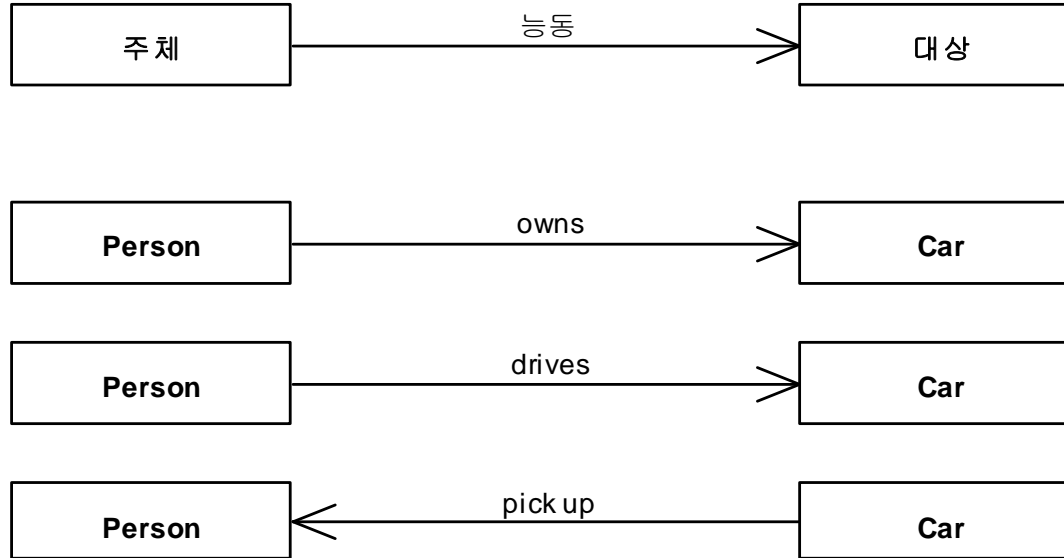
- attributeName: AttributeType [Multiplicity]  
(예) name: String [1..2] = "Michael"

✓ 표기법

표기법	의미
1	정확히 1
0..1	0 이거나, 혹은 1
*	무제한(0 포함)
1..*	적어도 하나 이상
2..6	2 부터 6 까지
2, 4	2 이거나, 혹은 4



## ✓ 연관 방향성의 이해





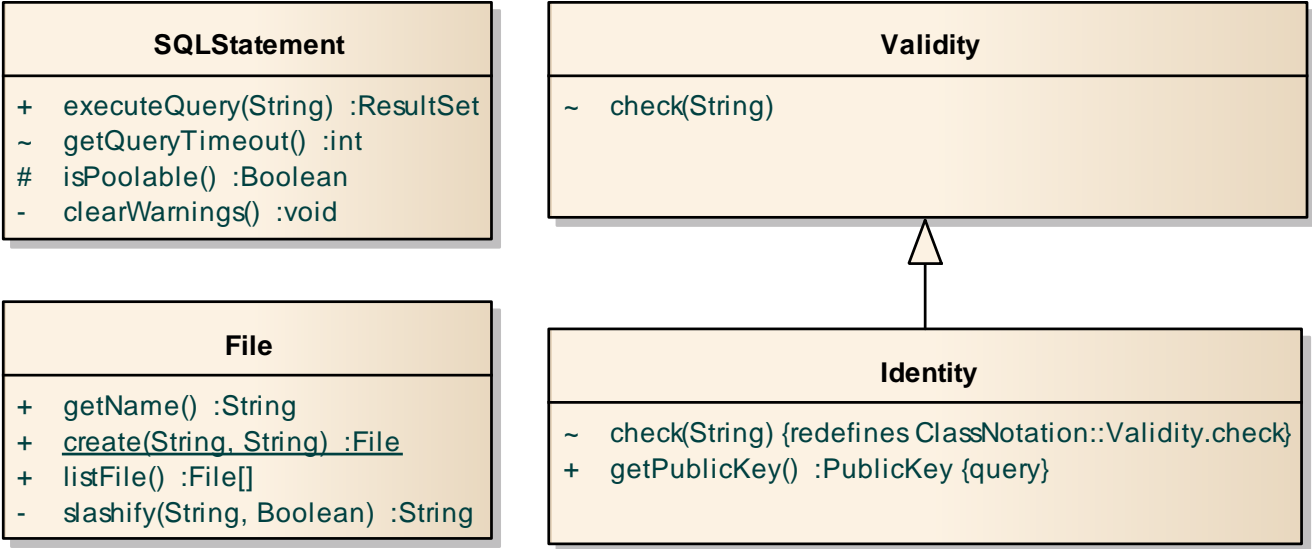
# 오퍼레이션 *Operation*

✓ 개요

- 클래스가 수행할 행위

✓ 표기법

- visibility name(parameter-list) : return-type {property-string}



오퍼레이션의 가시성, 파라미터, 프로퍼티 표기법

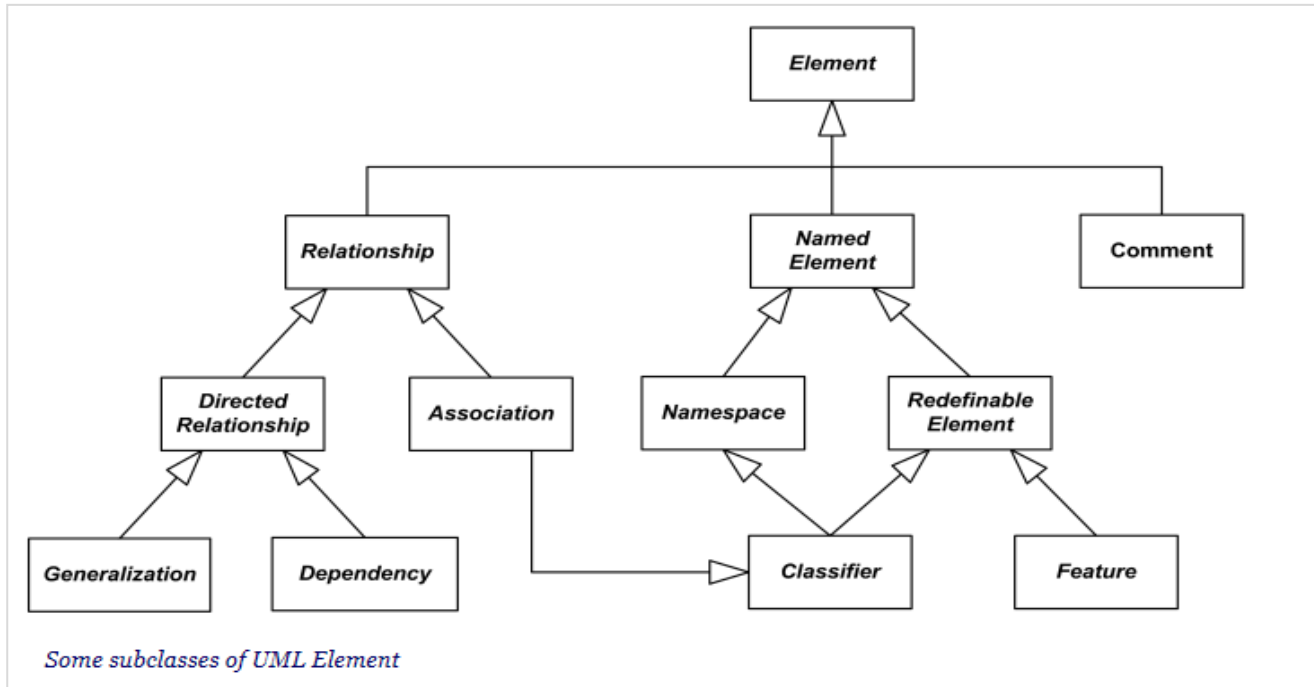
# 관계 Relationship

## ✓ 연관 : 지속적인 강한 관계

- 집합, 합성

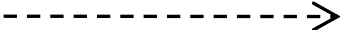







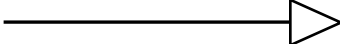
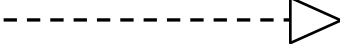
## ✓ Directed Relationship

- 의존 : 일시적인 약한 관계
- 상속 : target의 프로퍼티, 오퍼레이션을 포함하는 관계



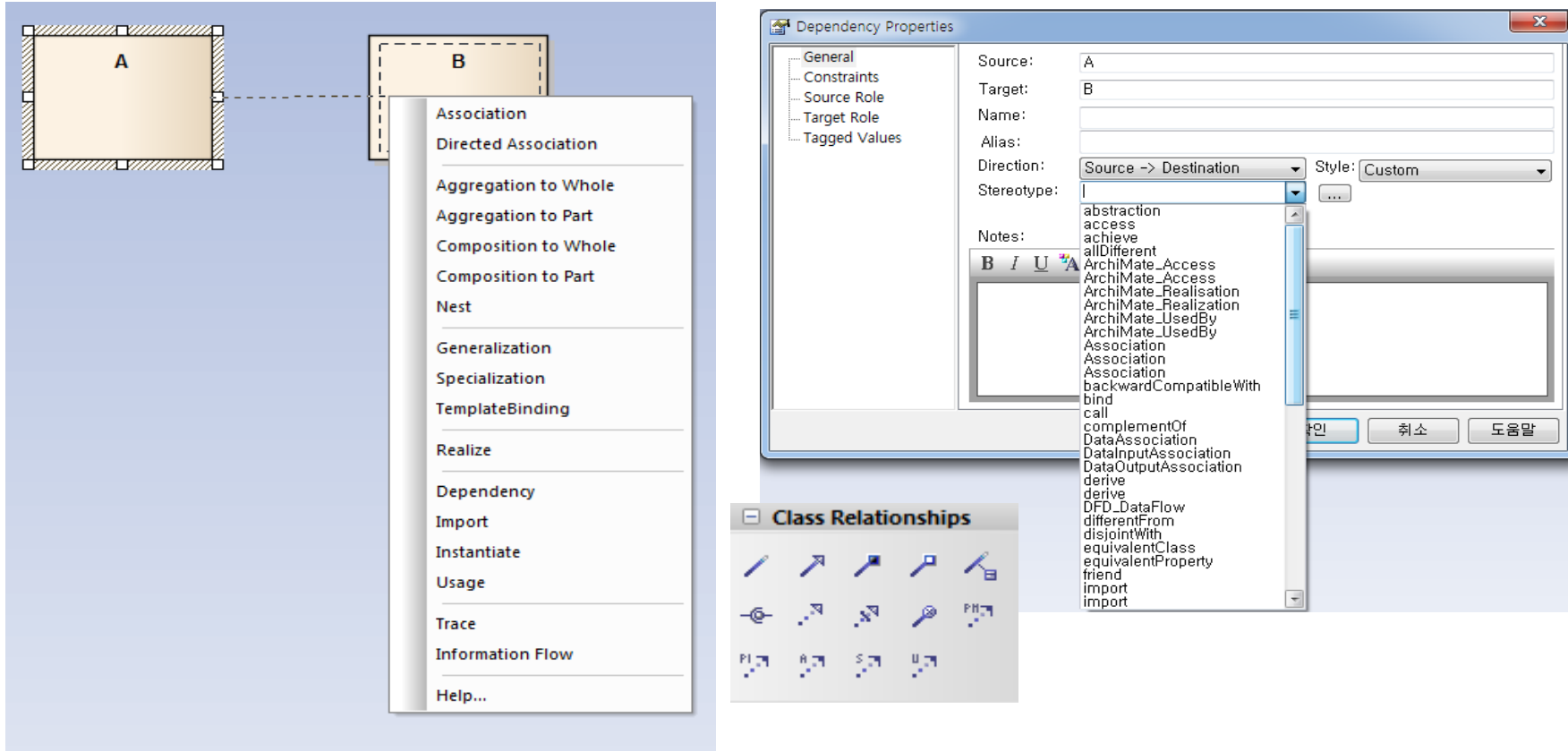
# 관계 Relationship

✓ 클래스 다이어그램에서 사용되는 관계의 유형은 아래와 같다.

유형	UML 표기		설명
	소스	타겟	
의존 Dependency			소스 요소가 타겟 요소의 변화에 영향을 받음
연관 Association			소스와 타겟 요소간의 연결
집합 Aggregation			타겟 요소는 소스 요소의 부분 (전체-부분)
합성 Composition			집합관계의 더욱 강력한 표현
포함 Containment			소스 요소가 타겟 요소를 포함
상속 Generalization			소스 요소는 보다 일반적인 타겟 요소를 구체화함
실현 Realization			소스 요소는 타겟 요소에 명시된 계약을 수행

# 관계 Relationship

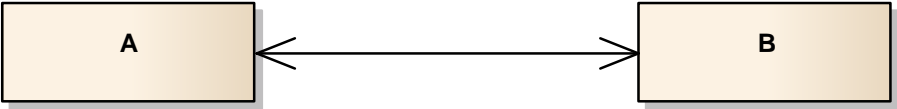
## ✓ 모델링 도구에서의 관계



모델링 도구는 매우 상세한 관계까지 정의할 수 있도록 지원해주며, 정의되지 않은 사용자 정의 타입도 가능하다. 이러한 메커니즘은 UML 확장 메커니즘과 유사하며 반비례적으로 프로그래밍 언어로서는 부적당해진다.

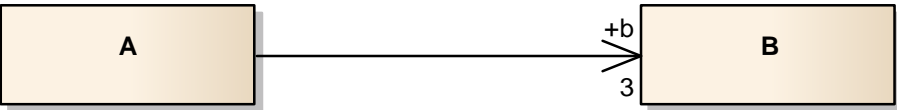
# 양방향 연관 vs 단방향 연관

## ✓ 양방향 연관



```
class A {
    B b;
}
class B {
    A a;
}
```

## ✓ 단방향 연관

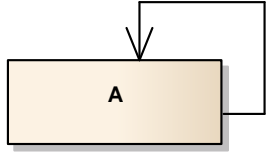


```
class A {
    B[] b = new B[3];
}
class B { }
```



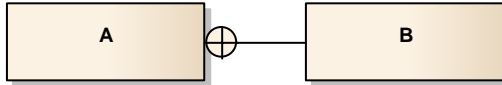
# 재귀적 연관과 내부 클래스

## ✓ 재귀적 연관관계



```
class A {  
    A a;  
}
```

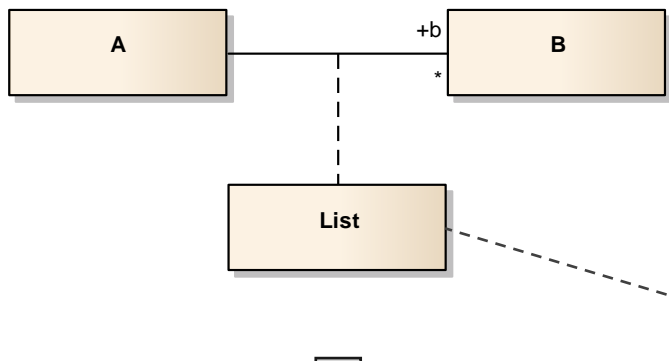
## ✓ 내부 클래스 (Inner class)



```
class A {  
    class B {}  
}
```

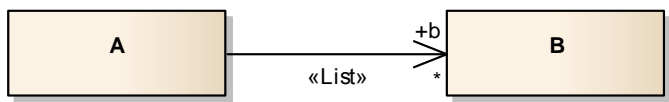
# 연관 클래스 Association class

## ✓ 연관 클래스



```
class A {  
    List<B> b;  
}  
  
class B {}
```

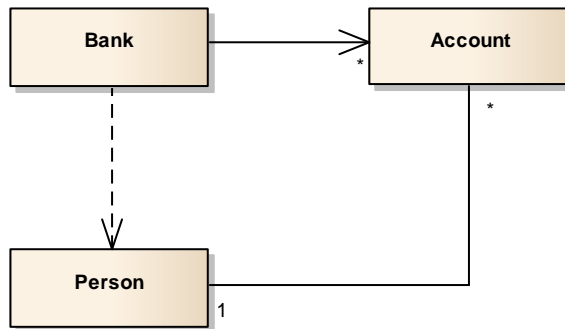
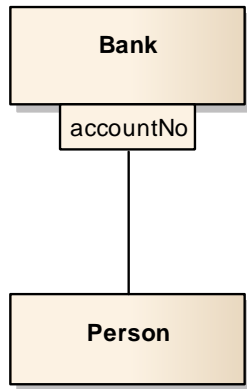
연관자체를 또 하나의 독립적인 클래스로 표현한다. 연관을 클래스로 표현하기 때문에 연관 자체도 다른 클래스와 마찬가지로 속성이나 오퍼레이션을 가질 수가 있다.



# 한정 연관 *Qualifiers*

## ✓ 한정 연관

- 식별정보(수식자)가 있는 연관을 표시하여 핵심이 되거나 부각시키고자 하는 구조 또는 행위를 표현함



```
class Bank {
    List<Account> accounts;

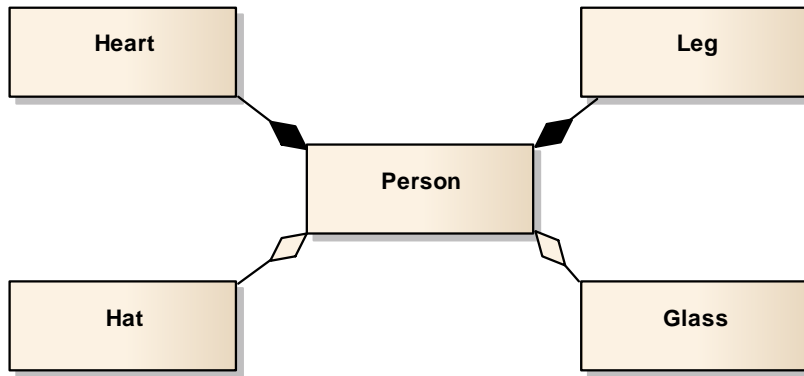
    public Person getPersonByAccountNo(int accountNo) {
        for(Account account : accounts) {
            // if account's number equals accountNo parameter then
            return account.person;
        }
        return null;
    }
}
```

```
class Account {
    Person person;
}
```

```
class Person {
    List<Account> accounts;
}
```

# Aggregation vs Composition

- ✓ 집합(Aggregation)과 합성(Composition)은 전체와 부분(whole-or-part-of)의 관계를 나타낸다.
  - 전체-부분의 관계, 얼마나 더 끈끈한가.
- ✓ 합성은 집합의 강한 형태로서,
  - 합성에서 하나의 객체는 어느 한 순간에 단지 한 객체만의 부분이 되며 다른 객체와 공유될 수 없다. (Sharing)
  - 합성에서 전체(whole) 객체는 부분(part) 객체의 생성과 소멸을 관리한다. (Life cycle)



```
class Person {  
    Heart heart;  
    Leg leg;  
    Hat hat;  
    Glass glass;  
}
```

```
class Heart {}  
class Leg{}  
class Hat{}  
class Glass{}
```

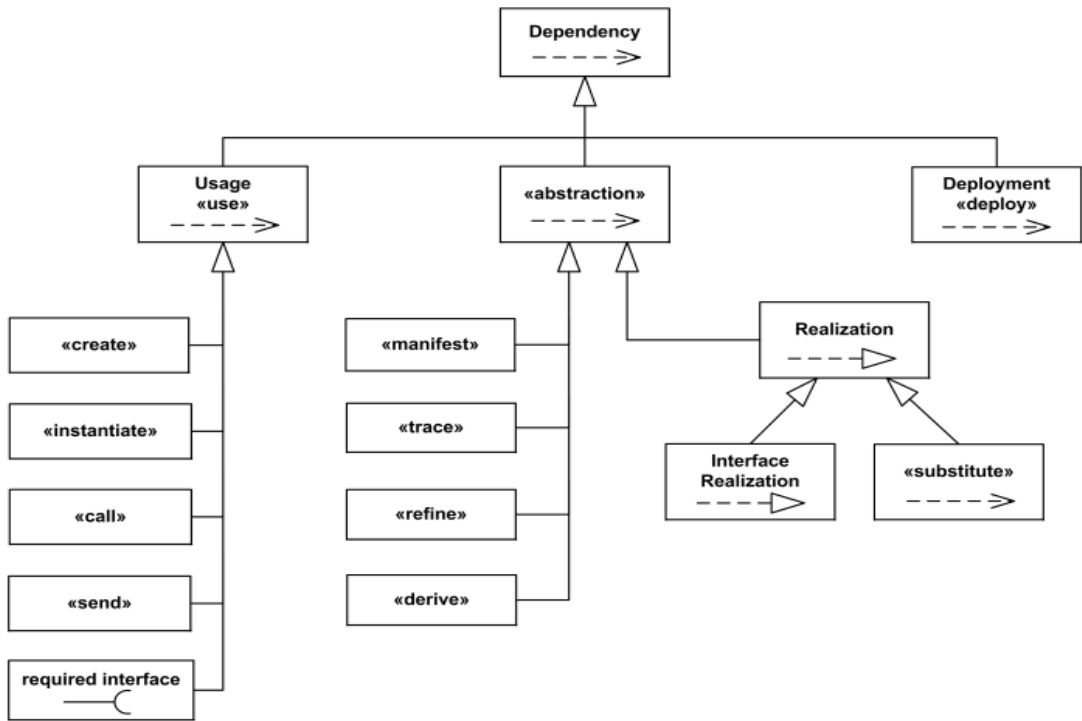
참고. UML 2.x에서 Aggregation 표기법은 일반 연관과 차이가 없기 때문에 사라짐

✓ 개요

- 한 요소의 변화가 다른 요소에 영향을 미칠 경우 의존성이 존재

✓ 표기법

- UML에서 점선 화살표는 의존성을 의미(키워드로 명확한 관계 정의)

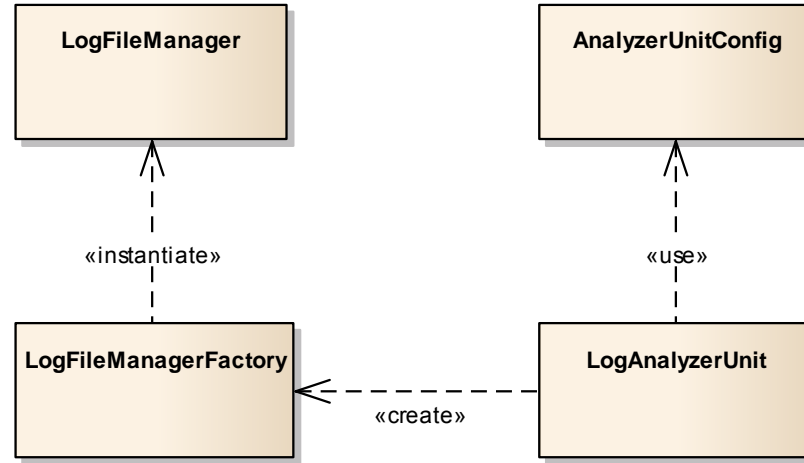


Dependency relationship overview diagram

## 의존성 Dependency [2/3]

### ✓ 키워드 예

- «instantiate»
  - target 인스턴스
- «use»
  - target을 필요로 함
- «create»
  - target 인스턴스 생성



```
class LogAnalyzerUnit {  
  
    private LogFileManager manager;  
  
    public LogAnalyzerUnit(AnalyzerUnitConfig config) {  
        manager = new LogFileManagerFactory().createLogFileManager(config);  
    }  
}
```

# 의존성 *Dependency* [3/3]

✓ 키워드를 몇 가지 더 알아보자

키워드	의미
«call»	소스가 타겟의 오퍼레이션을 호출한다
«create»	소스가 타겟의 인스턴스를 생성한다
«derive»	소스가 타겟으로부터 파생한다
«instantiate»	소스의 오퍼레이션이 타겟의 인스턴스를 생성한다.
«realize»	소스가 타겟에서 정의한 명세 또는 인터페이스를 구현한다
«trace»	클래스에 대한 요구 사항을 추적하거나, 한 모델의 변경사항이 다른 곳에 어떻게 영향을 주는지를 추적할 때 사용한다
«use»	구현을 위해서 소스가 타겟을 필요로 한다

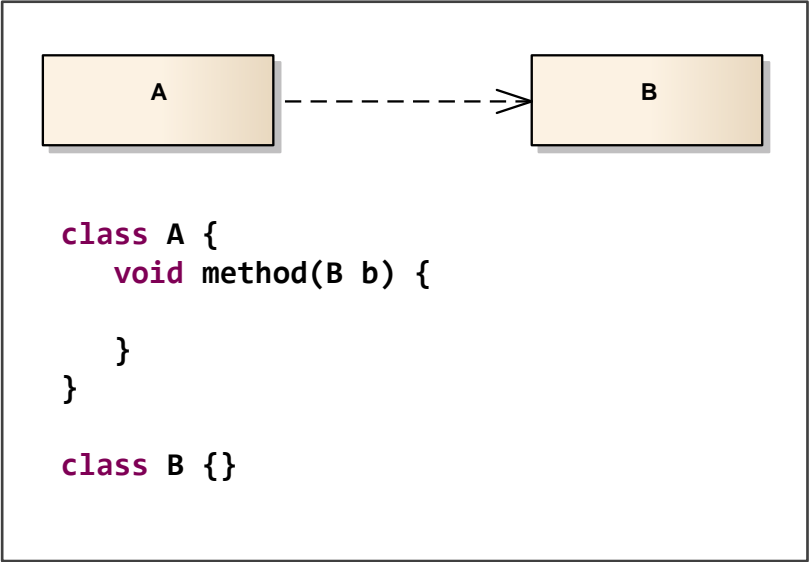


# 의존 vs 연관

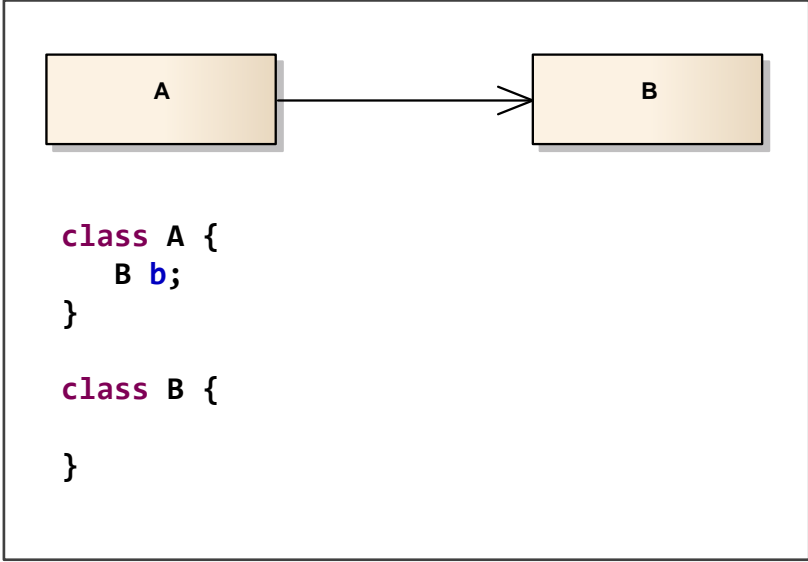
✓ 두 관계 모두 객체에 메시지를 전달하는 통로의 역할

	코드와의 관련성	관계 범위	방향성
연관	클래스 속성	클래스 범위	양 방향 가능
의존	연산의 인자, 지역객체, 전역객체	하나의 연산 범위	단 방향만 가능

<의존>



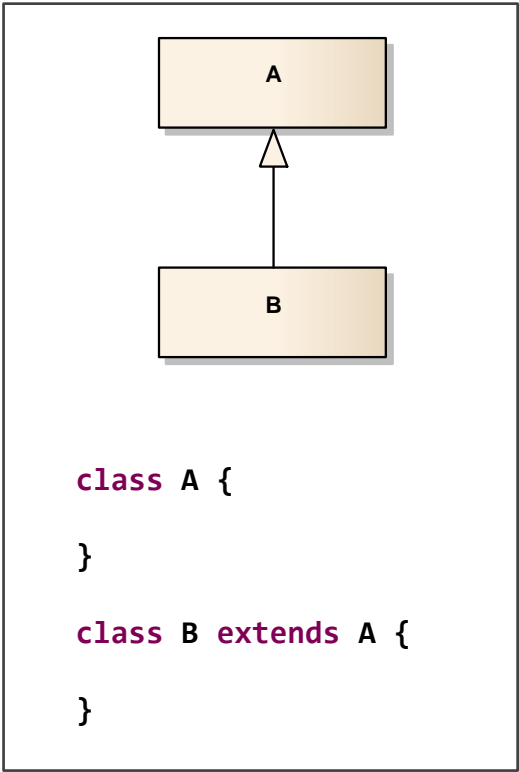
<연관>



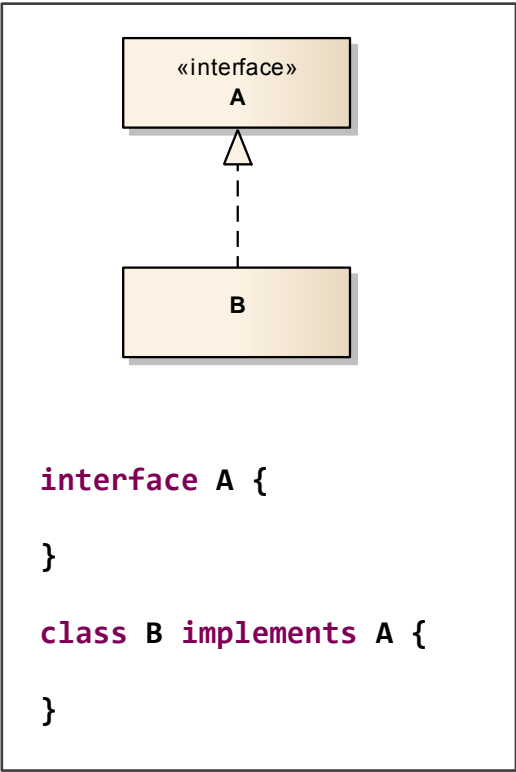
# 상속관계 Inheritance

✓ 일반화(Generalization) 와 실체화(Realization)

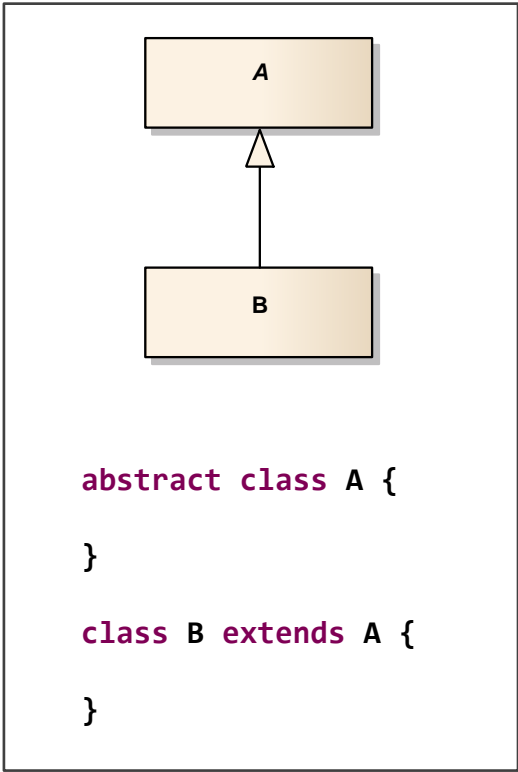
<일반화>



<실체화>



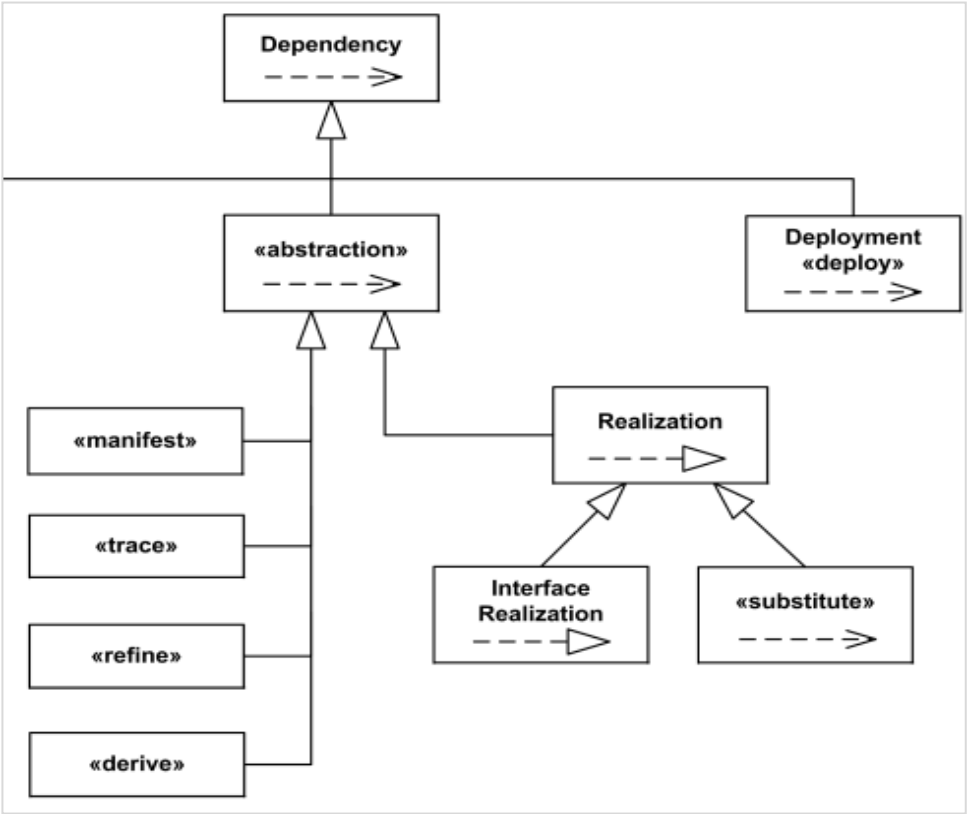
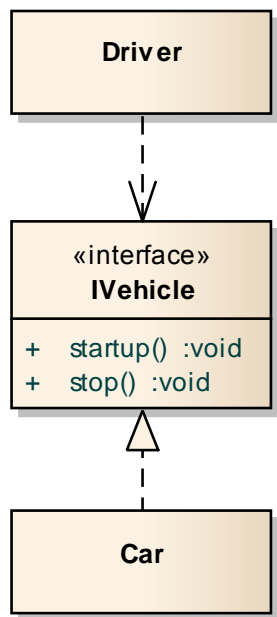
<일반화>



# 의존 vs 실체

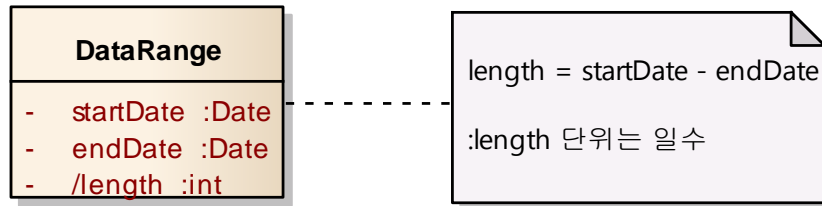
✓ 두 관계 모두 객체에 의존적이다

- 의존과 실체는 서로 다르게 보이지만, 사실 실체화(implements)는 target의 Interface에 굉장히 의존적이다.  
하지만 의존과 실체화는 전혀 다르다.



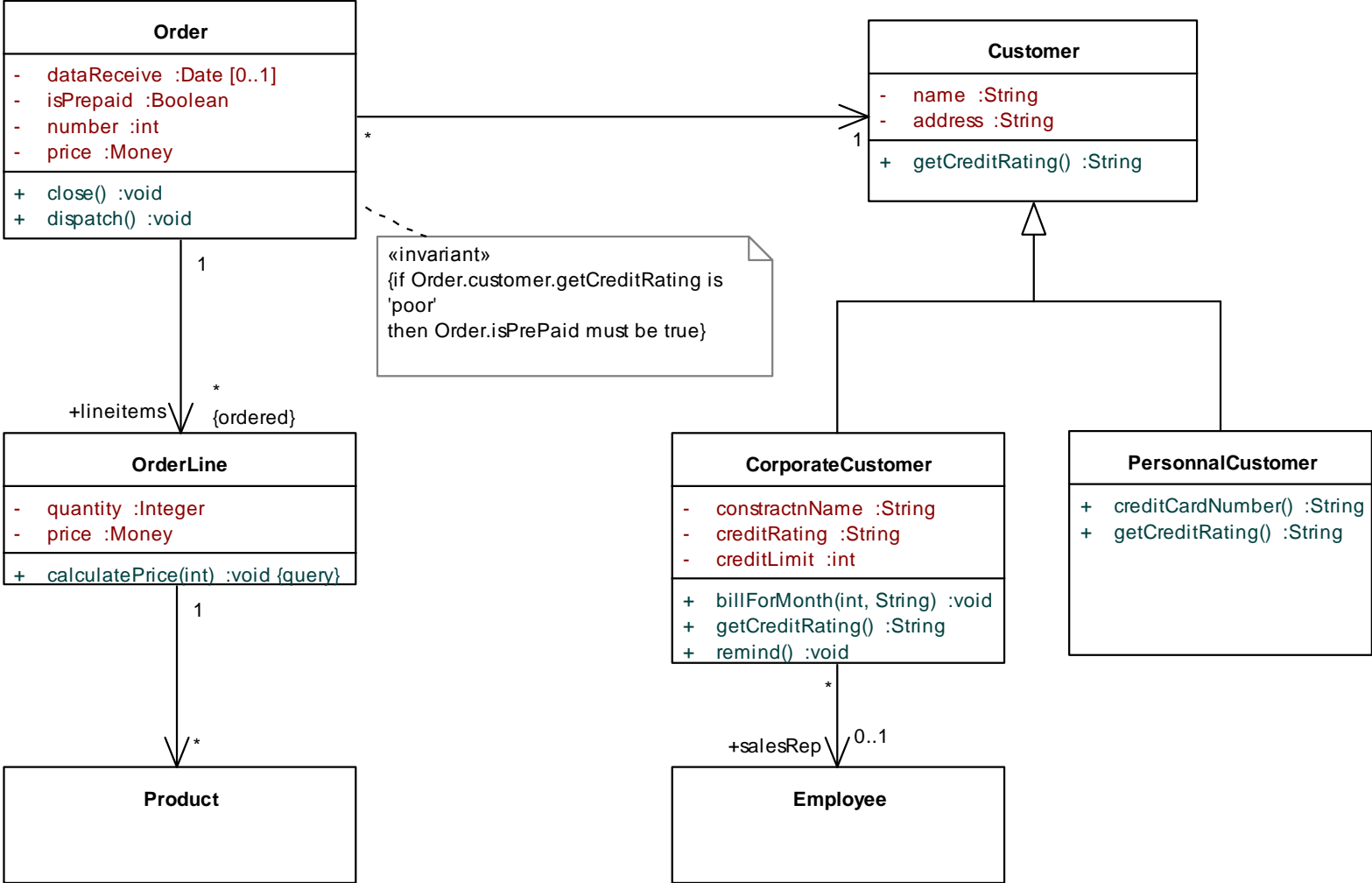
## ✓ 개요

- 노트는 다이어그램을 설명하기 위한 주석
- 다이어그램과 연결되거나 그 자체로 사용될 수 있음



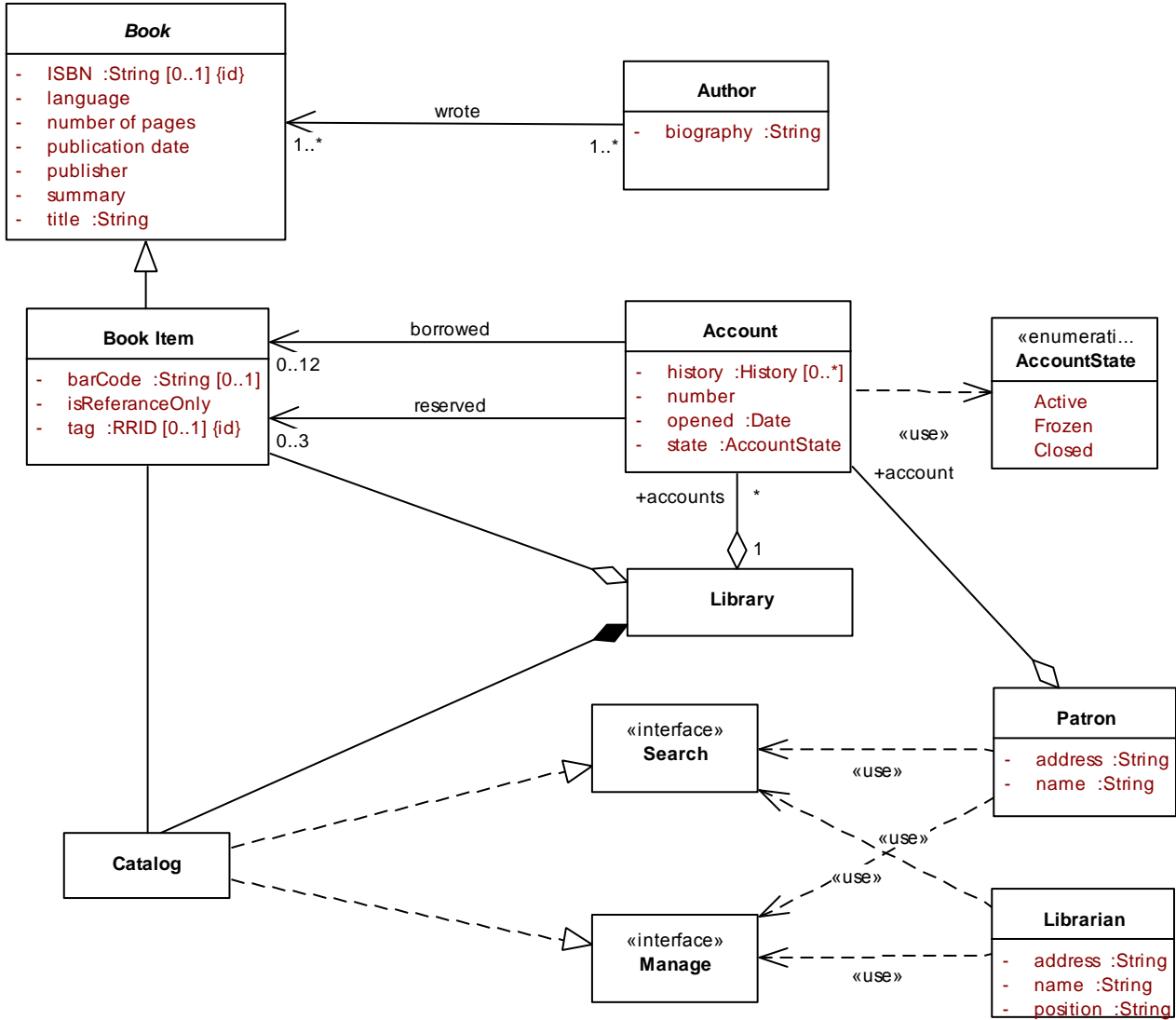
# 클래스 다이어그램 실습

## ✓ 상품 주문하기 도메인



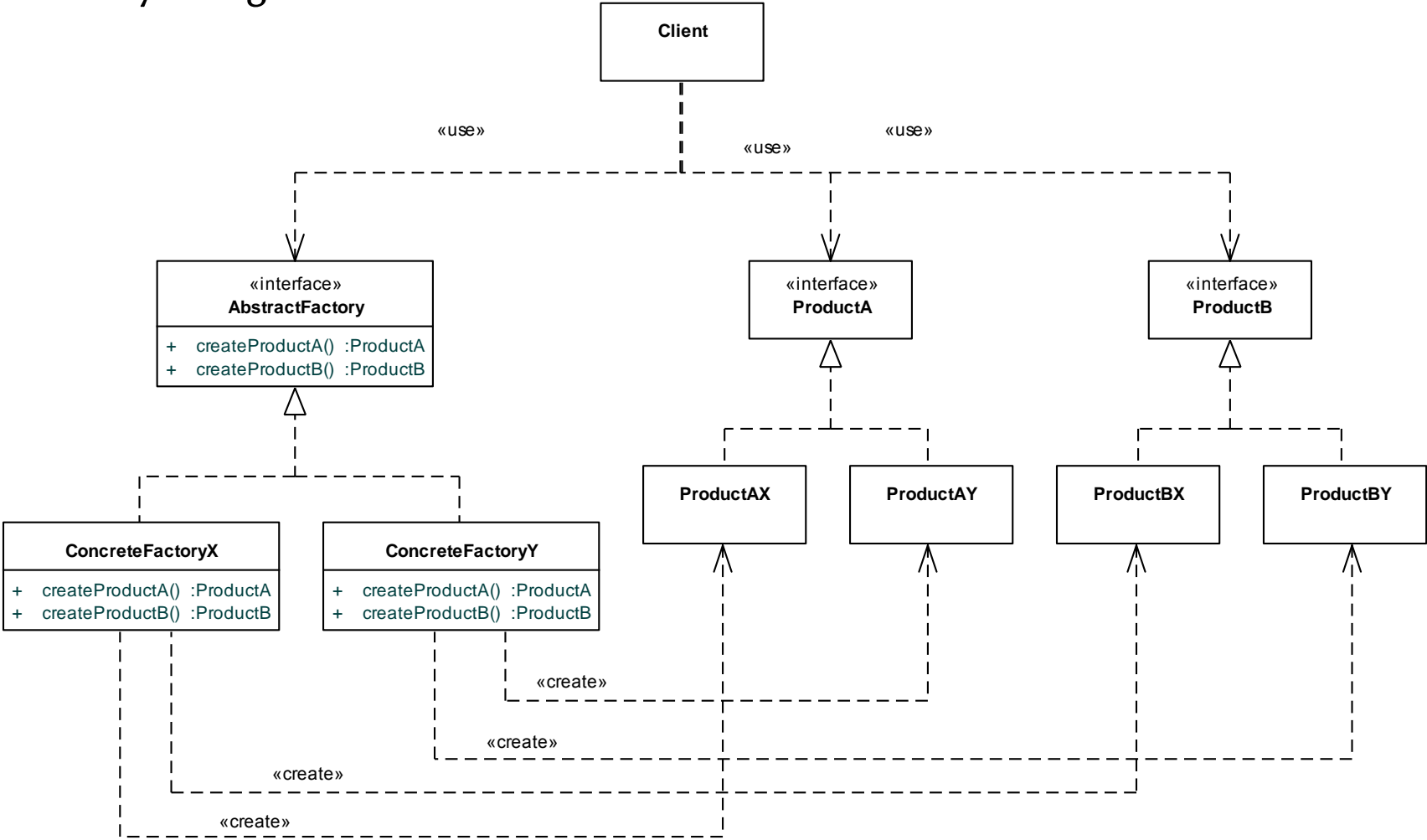
# 클래스 다이어그램 실습

## ✓ 도서관 도메인



# 클래스 다이어그램 실습

## ✓ Abstract Factory Design Pattern





### 3. 상호작용 다이어그램

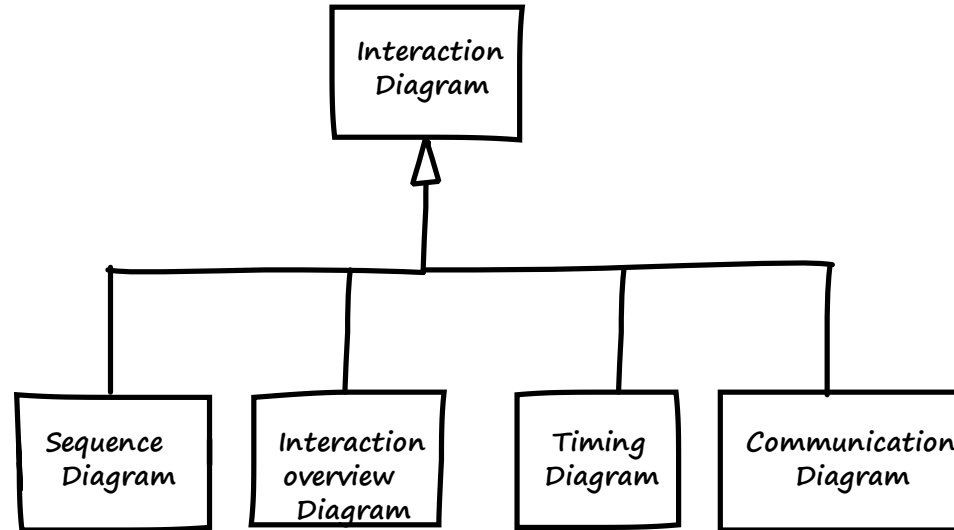
- 상호작용 다이어그램 개요
- 시퀀스 다이어그램



# 상호작용 다이어그램 개요

## ✓ 상호작용 다이어그램(Interaction Diagram)

- 행위다이어그램의 일종으로 목적을 달성하기 위하여 상호 협력하는 객체들이 메시지를 어떻게 교환하는지를 표현함
- UML 2.x 의 상호작용 다이어그램
  - Sequence diagram
  - Interaction overview diagram
  - Timing diagram
  - Communication diagram



# 시퀀스 다이어그램 *Sequence Diagram*

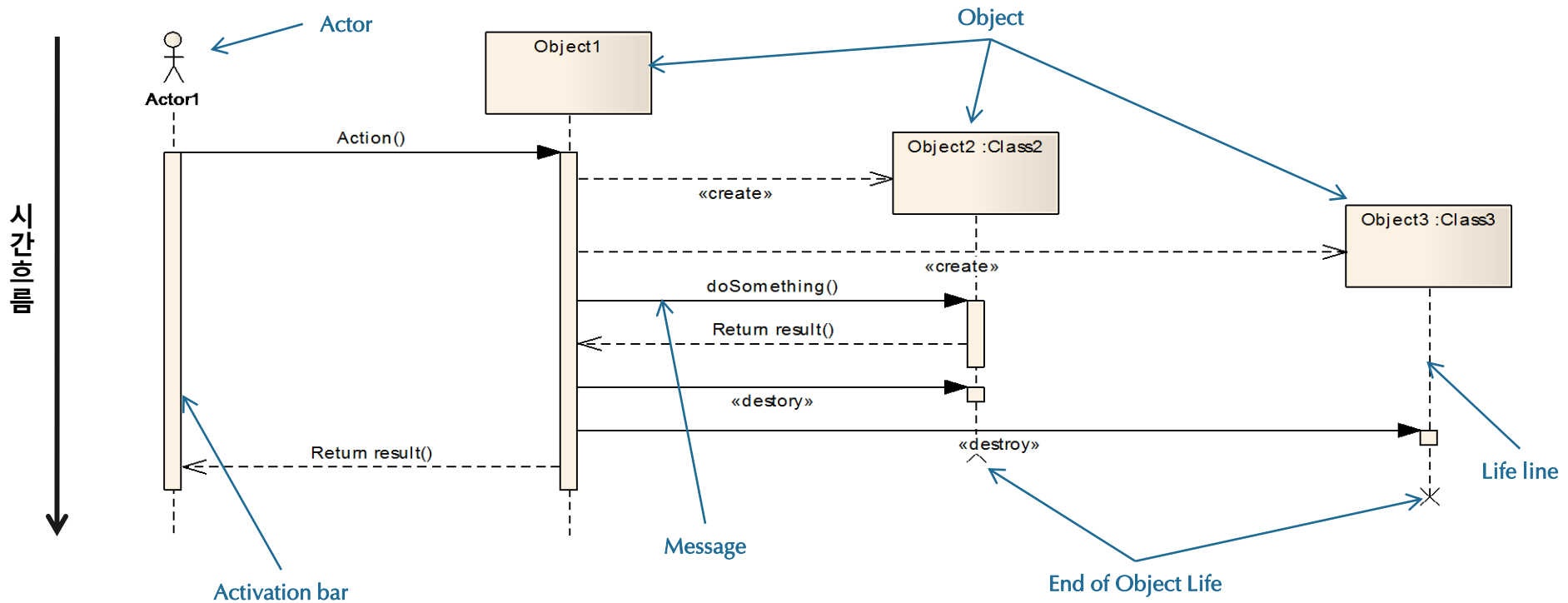
## ✓ 정의

- 상호작용 다이어그램(동적 모델링 기법)은 어떤 행동 안에서 객체 그룹들이 어떻게 협력하는지를 설명한다
- 시퀀스 다이어그램은 한 가지 시나리오에 대한 행동을 묘사
- 몇 개의 객체를 예로 들어서 유스 케이스 내에서 이 객체들 간의 전달되는 메시지를 표시한다
- 가장 많이 사용하는 상호작용 다이어그램 중 하나로 생명선 사이로 메시지 교류에 초점을 둔 다이어그램
- 객체, 실선 화살표로 그려지는 메시지 그리고 수직 진행 상황을 나타내는 시간으로 구성

# 시퀀스 다이어그램 구성요소

## ✓ 시퀀스 다이어그램의 구성요소

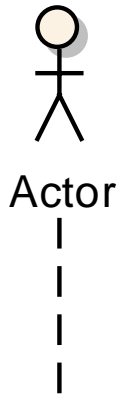
- 요소 : 액터 *Actor*, 객체(참가자) *Object*
- 관계 : 메시지 *Message*
- 기타 : 시간 *Time*, 생명선 *Life Line*, 활성화 *Activation bar*, 교류 프레임 *interaction frame*



# 액터 Actor

✓ 액터 Actor

- 시스템의 외부에 존재하면서 시스템과 교류 혹은 상호작용하는 것
- 시스템에 서비스를 요청하는 존재
- 시스템에게 정보를 제공하는 대상



액터의 예)

보험 시스템	관리자, 총무, 판매자, 계약자, 피보험자 등
오픈 마켓 시스템	회원, 구매자, 배달자, 관리자, 배송시스템 등
병원관리 시스템	의사, 간호사, 환자, 수납책임자 등

# 객체(참가자) *Object*

✓ 객체(참가자) *Object*

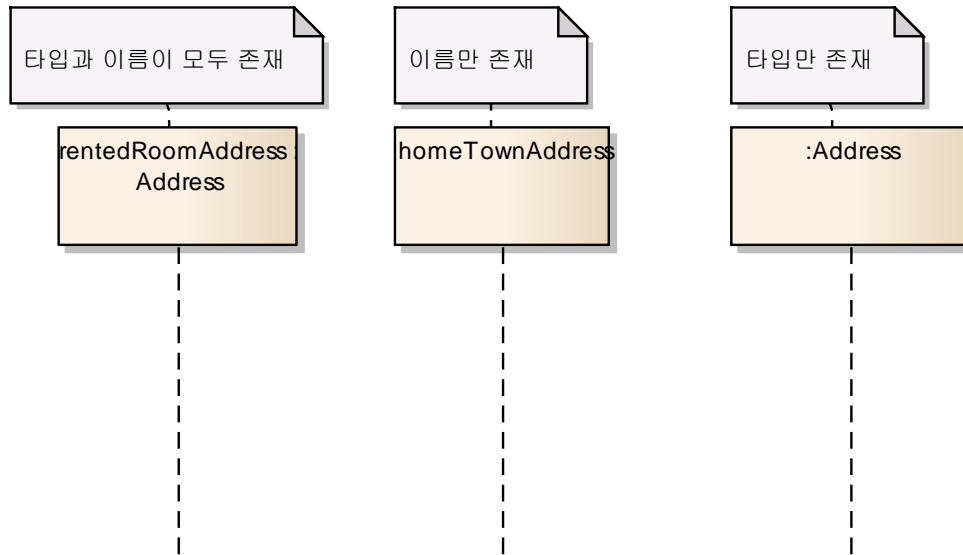
- 클래스의 인스턴스
- 시스템에서는 해당 클래스 타입으로 선언된 변수의 형태로 존재
- 생성되고 소멸되기까지의 생명주기 동안 다양한 상태의 변화가 존재

객체의 예)

홍길동 : 사원	: 사원 클래스의 객체 중 홍길동 객체를 의미
홍길동	: 홍길동 객체로 클래스는 정의가 없음
: 사원	: 사원 클래스의 일반 객체 (일반적인 객체를 지칭)
홍길동 : 사원	: 사원 클래스의 홍길동 객체로서 객체 속성을 모두 표현
사번=20391 이름=홍길동 급여=4000 성별=남 직위=과장	

## ✓ 생명선 *Life line*

- 생명선은 상호작용하는 개별적인 참가자를 대표하는 요소
- 참가자의 생존 기간을 의미
- 생명선은 머리에 해당하는 사각형과 참가자의 수명을 나타내는 수직선으로 표현된다
- 참가자 박스의 아래쪽 중심에서 점선을 그리며 내려감



## ✓ 메시지 *Message*

- 메시지는 상호작용하는 생명선 사이에서 명세화된 종류의 의사소통을 정의하는 요소
- 객체지향 패러다임에서 객체와 객체가 통신하는 유일한 수단
- 향후 클래스의 오퍼레이션으로 구현
- 생명선의 첫 부분에서 다른 생명선의 끝부분으로 향하는 화살표로 표현
- 화살표의 머리모양은 메시지의 형태를 의미

## ✓ 표기법

- Message : signal-or-operation-name (arguments) : [return-value]

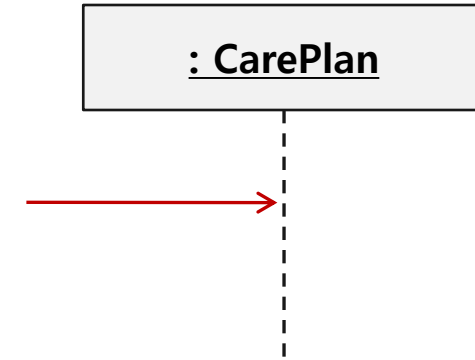
## ✓ 변화된 메시지

- 1.x 메시지 유형
  - Flat flow of control
  - Nested flow of control
  - Asynchronous of control
  - Return flow
- 2.x 메시지 유형
  - 액션에 의한 메시지
    - . synchronous call
    - . asynchronous call
    - . Create Message
    - . Delete Message
    - . Reply Message
  - 이벤트에 의한 메시지
    - . Lost Message
    - . Found Message



## ✓ Flat flow of control

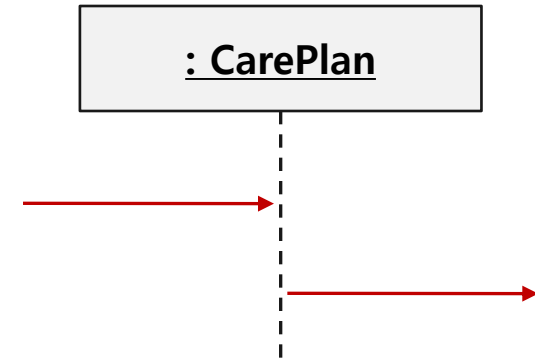
- 가장 일반적인 메시지 형태
- 객체에 메시지를 연결할 때 사용



*Flat flow of control*

## ✓ Nested flow of control

- Procedural call과 같은 의미로 사용
- 중첩Nested되는 경우 내부 메시지의 결과가  
모두 돌아와야 가장 처음 시작한 메시지의  
결과가 되돌려 보내짐
- 메시지 결과가 돌려지게 될 때까지  
다음 처리를 진행하지 않는 동기화 메시지

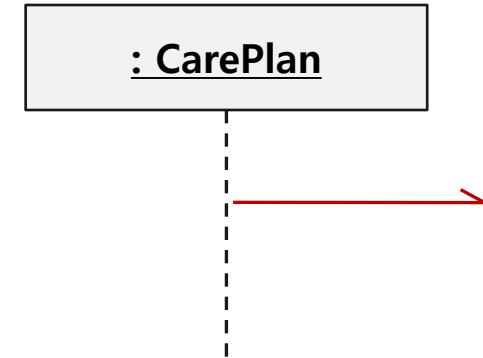


*Nested flow of control*

## 메시지 *Message* (4/11)

### ✓ Asynchronous of control

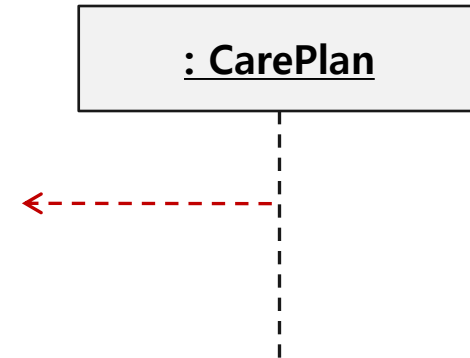
- 객체가 보낸 메시지의 결과를 기다리지
- 않고 다음 처리를 진행할 경우 사용



*Asynchronous flow of control*

### ✓ Return flow

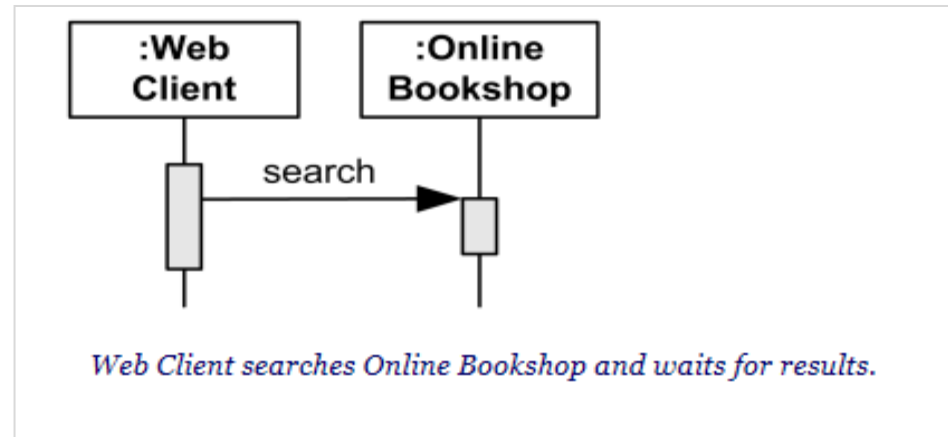
- 메시지를 처리한 결과 리턴을 의미
- 필요한 경우에만 표현



*Return flow*

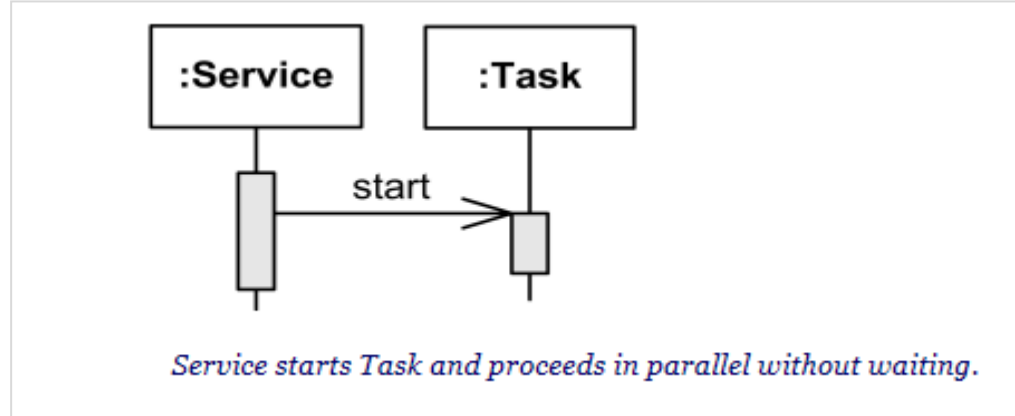
## ✓ Messages by Action Type

- synchronous call
  - 동기호출은 일반적으로 오퍼레이션을 호출하거나 메시지를 보내고 응답을 기다리는 동안 실행을 일시 중지하는 것을 나타냅니다
  - 동기호출은 아래와 같이 화살표의 머리가 채워져 있다



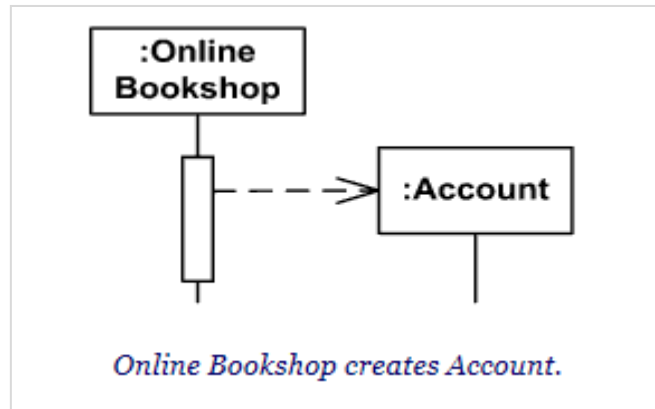
## ✓ Messages by Action Type

- asynchronous call
  - 비 동기호출은 메시지를 보내고 반환 값을 기다리지 않고 즉시 진행되는 것을 나타냅니다
  - 비 동기호출은 아래와 같이 화살표의 머리가 열려져 있다



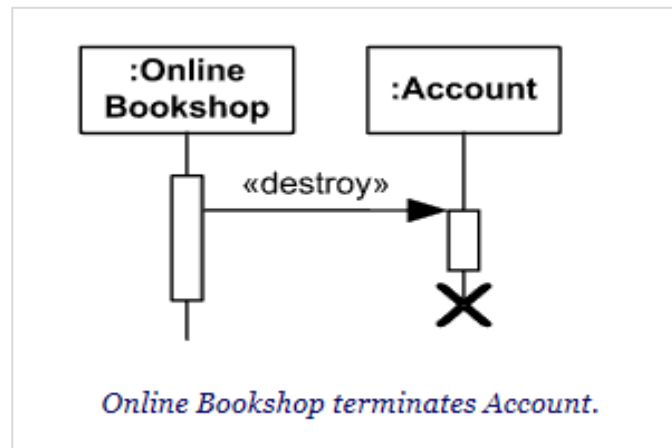
## ✓ Messages by Action Type

- Create Message
  - 생성 메시지는 그 자체로 생명선을 생성시킨다.
  - 생성 메시지의 화살표도 동기 호출과 유사하게 화살표가 열려져 있으며 생성된 생명선 머리를 가리키고 있다



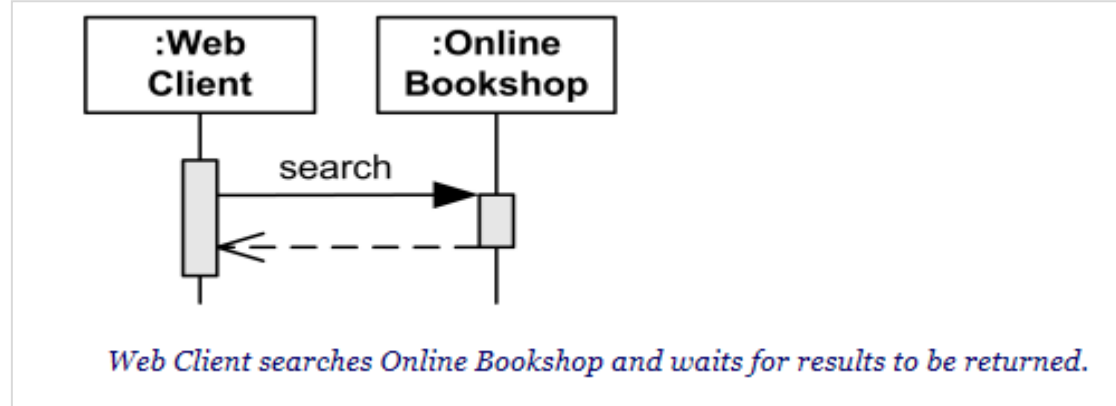
## ✓ Messages by Action Type

- Delete Message
  - 삭제메시지는 생명선을 종료시킨다
  - 생명선 끝에 X 형태로 삭제 발생이 보인다



## ✓ Messages by Action Type

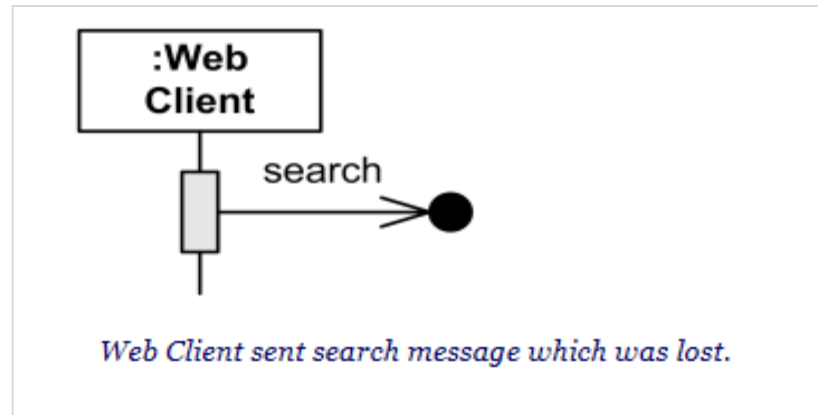
- Reply Message
  - 오퍼레이션을 호출한 회신 메시지는 점선으로 이어진 열려진 화살표로 표기한다
  - 생성 메시지와 비슷하다



## ✓ Messages by Presence of Events

### ▪ Lost Message

- 분실 메시지는 보내는 이벤트는 알고 있지만 수신하는 이벤트는 알 수 없다
- 메시지가 목적지까지 도달하지 못했다고 해석된다
- 분실 메시지는 메시지의 화살표 끝 부분에 작고 검은 색 원으로 표기한다





## ✓ Messages by Presence of Events

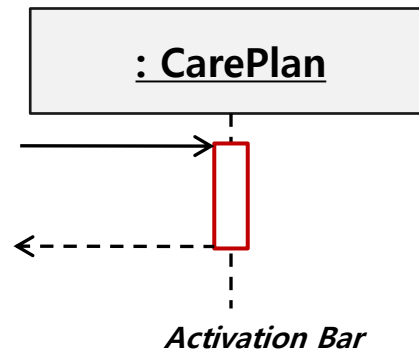
- Found Message
  - 발견 메시지는 보내는 이벤트는 알 수 없지만 수신하는 이벤트는 알 수 있다
  - 메시지가 시작한 설명할 수 있는 범위가 아니다
  - 발견 메시지는 메시지의 화살표 시작 부분에 작고 검은 색 원으로 표기한다



## 활성 막대 *Activation bar*

### ✓ 활성 막대 *Activation Bar*

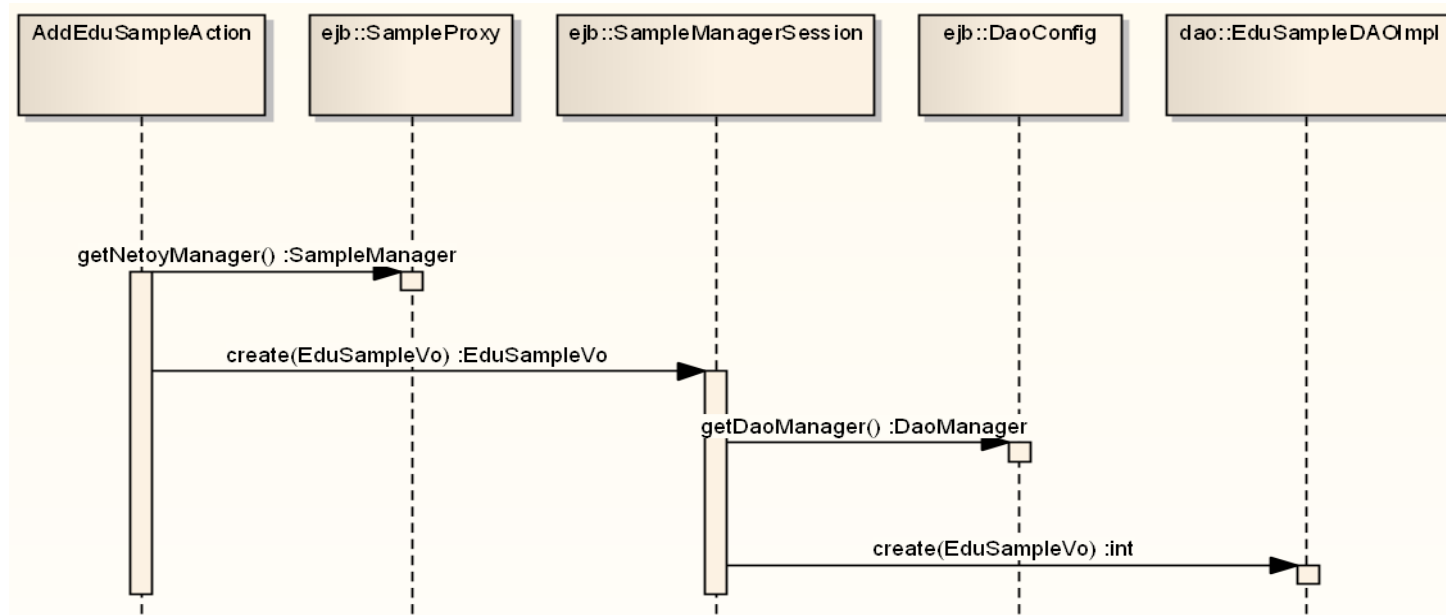
- 객체가 활성화 되어 있는 기간
- 객체의 method 중 하나에 대응
- 객체가 외부 메시지를 받고 다른 객체에 보낸 메시지에 대한 Return Flow를 기다리는 시간을 의미
- Life line 위에 긴 사각형으로 표시



# 시간 *Time*

## ✓ 시간 *time*

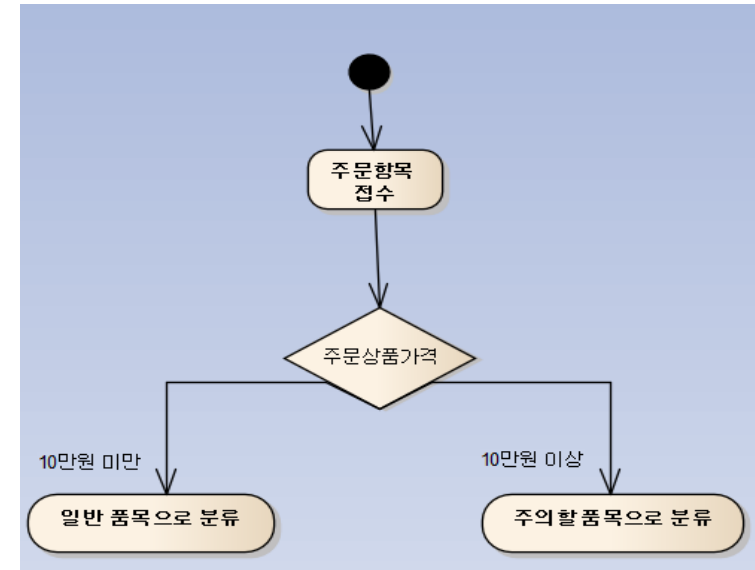
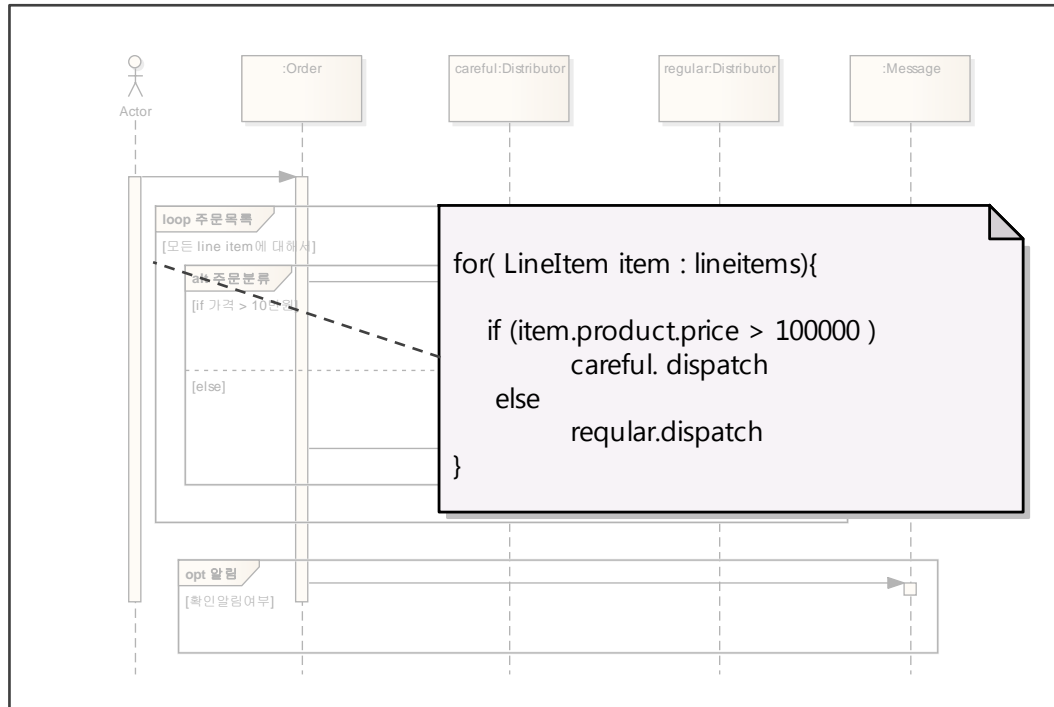
- 시간은 수직방향으로 나타내며, 가장 윗부분에서 아래로 향해 흐르기 시작함
- 객체 사각형과 간격이 더 가까운 메시지일수록 그보다 아래에 있는 메시지 보다 더 먼저 전송된 것
- 시퀀스 다이어그램은 2차원 다이어그램으로서 왼쪽에서 오른쪽으로 방향은 객체의 배열, 위에서 아래방향은 시간의 흐름



# 교류 프레임 *Interaction frame* [1/4]

## ✓ 교류 프레임

- 교류 프레임은 시퀀스 다이어그램에서 반복적인 행위와 어떤 조건에 따라 다르게 처리되는 행위를 표현하기 위한 것이다 하지만, 시퀀스 다이어그램은 교류 프레임을 표현하는 데 좋지 않다
- 교류 프레임으로 표현하고자 하는 제어구조는 액티비티 다이어그램이나 코드 자체를 이용하는 것이 좋다



# 교류 프레임 Interaction frame (2/4)

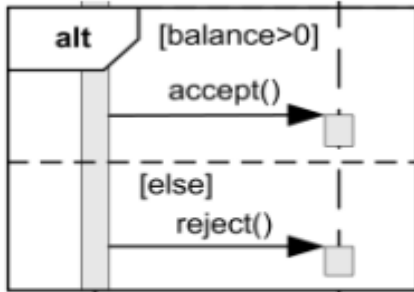
✓ 교류 프레임 종류

- 모든 프레임에는 연산자가 있고 가드를 가진다

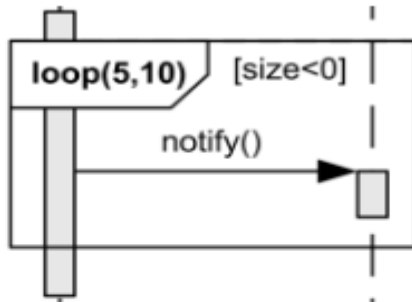
연산자	의미
Alt	선택적 : 여러 개의 부분 중에 조건을 만족하는 단 한 개만 실행
Opt	부가적 : 주어진 조건이 참일 때만 실행
Par	병렬 : 각각의 부분이 병렬로 실행
Loop	반복 : 여러 번에 걸쳐 실행. 반복의 조건이 가드에 명시
Neg	무의미한 : 의미 없는 교류를 가진 부분
Ref	참조 : 다른 다이어그램에 정의된 부분을 참조
Seq	시퀀스 다이어그램 : 원할 경우, 시퀀스 다이어그램 전체를 둘러쌀 때 사용

## 교류 프레임 *Interaction frame* (3/4)

### ✓ 교류 프레임 사용

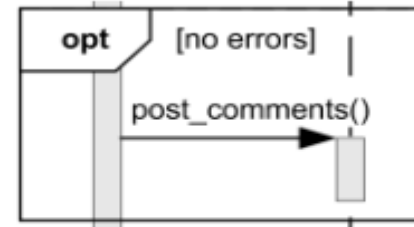


*Call accept() if balance > 0, call reject() otherwise.*

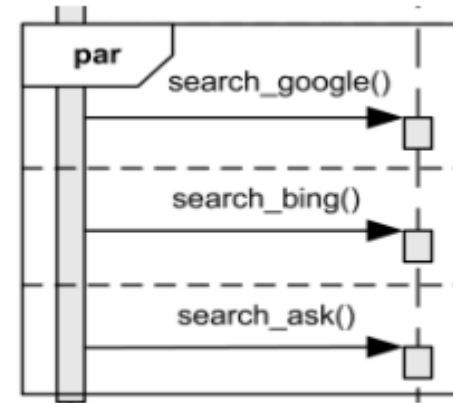


*We may guess that as per UML 2.3, the loop is expected to execute minimum 5 times and no more than 10 times.*

*If guard condition [size < 0] becomes false loop terminates regardless of the minimum number of iterations specified.  
(Then why do we need that min number specified?!)*



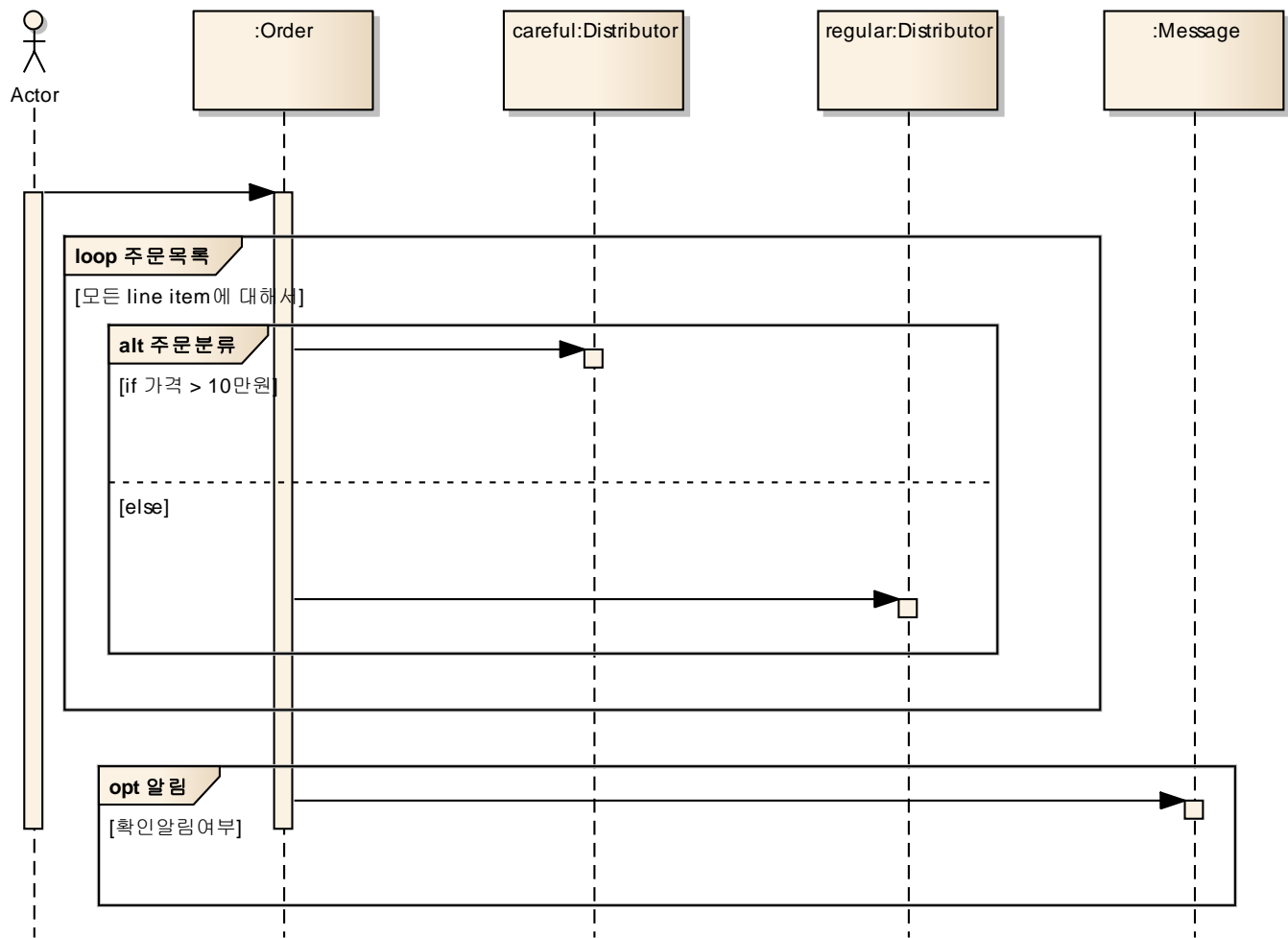
*Post comments if there were no errors.*



*Search Google, Bing and Ask in any order, possibly parallel.*

# 교류 프레임 Interaction frame (4/4)

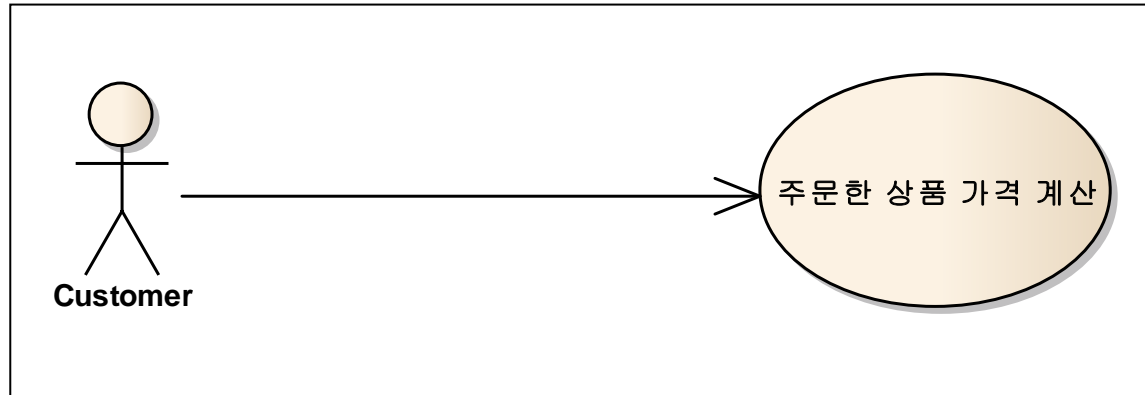
## ✓ 교류 프레임 사용



# 시퀀스 다이어그램 실습 [1/3]

## ✓ 한 가지 시나리오에 대한 행동 묘사를 표현하는 시퀀스 다이어그램

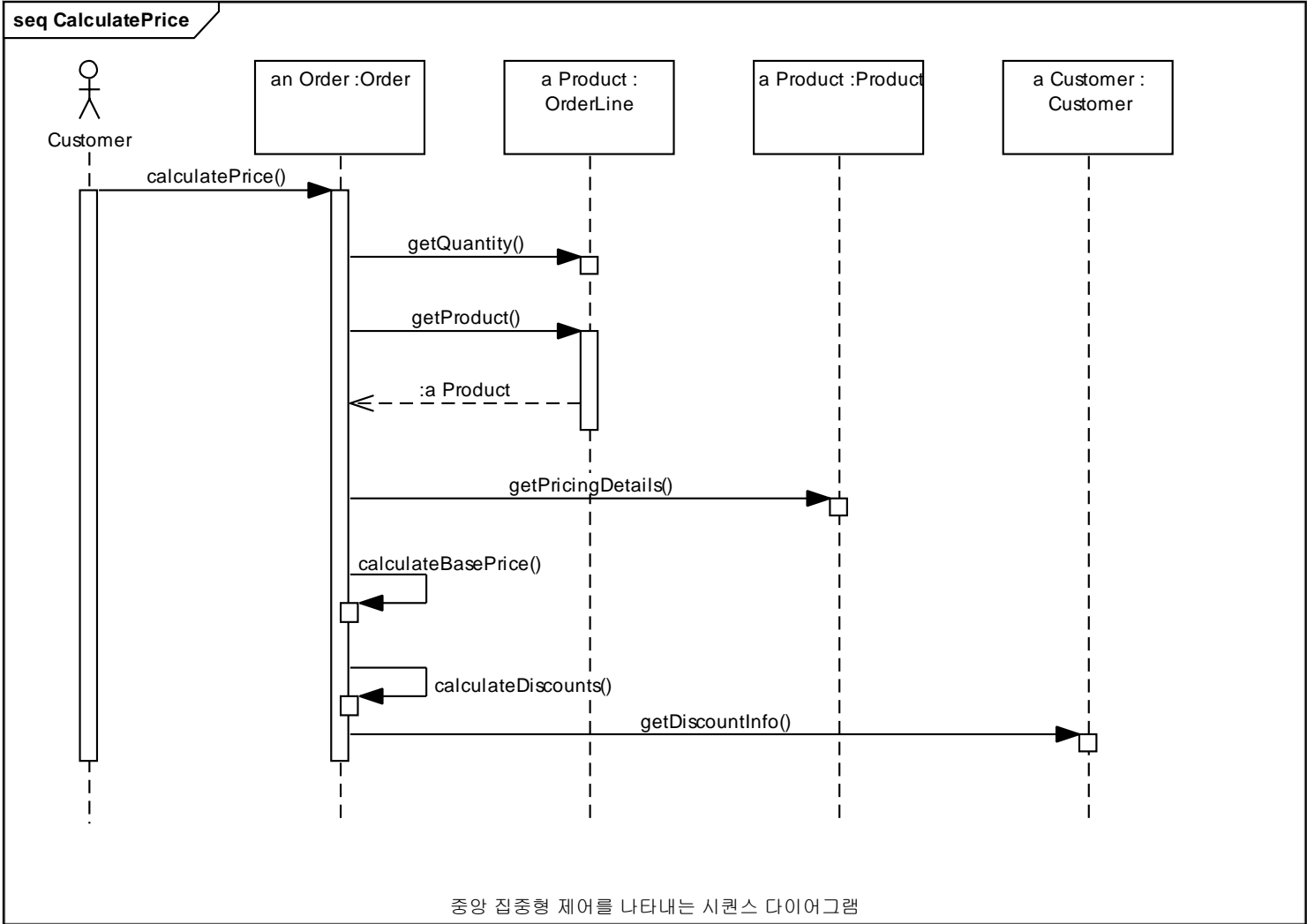
- 하나의 Use case : 고객은 자신이 주문한 상품 가격을 요청한다
- 하나의 Sequence를 그리기 위한 시나리오 작성
  - 주문(order)는 주문을 받은 모든 주문품목(Line Item)을 보고 그것들의 가격을 계산한다
  - 계산은 품목별 주문 내역(Order Line)의 상품(product)의 계산법에 기준해서 이뤄진다
  - 모든 주문 품목(Line Item)에 대해서 계산이 끝나면 주문(order)는 고객(customer)에 대해서 정의된 규칙에 따라 전체 할인을 계산한다





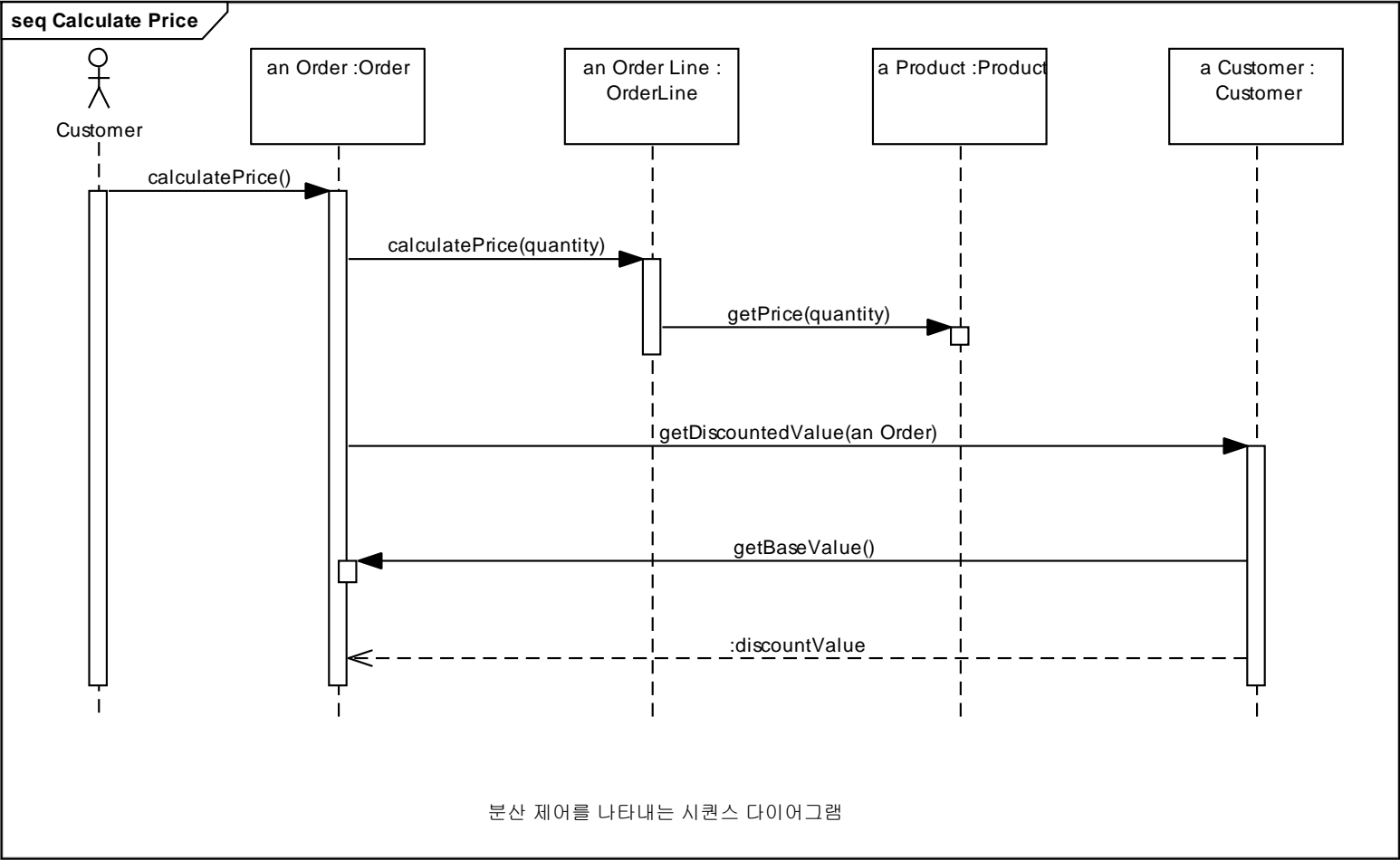
# 시퀀스 다이어그램 실습 [2/3]

## ✓ 주문한 상품에 대한 가격 계산 시퀀스 다이어그램 작성 #1



# 시퀀스 다이어그램 실습 [3/3]

## ✓ 주문한 상품에 대한 가격 계산 시퀀스 다이어그램 작성 #2



# 중앙 집중형 제어와 분산제어

## ✓ 중앙 집중형 제어

- 특성 : 한 참가자가 거의 모든 처리를 하고 다른 참가들은 데이터를 제공한다
- 장점 : 분산제어보다 단순하고, 객체를 추적하는 감각이 보다 수월하다
  - 모든 제어 로직이 한 곳에 집중되어 있기 때문에
- 단점 : 변경에 대한 영향도가 크다. 변경사항과 무관한 로직들이 변경될 수 있다

## ✓ 분산제어

- 특성 : 처리 작업이 여러 참가자들에게 분산되어 각각의 참가자가 전체 알고리즘의 작은 부분만 수행한다
- 장점 : 변경에 대한 영향을 최소화해야 한다는 목표를 달성 할 수 있다
- 단점 : 프로그램을 이해하려면 객체들을 따라 다녀야 한다

# 작성 및 주의사항 [1/2]

## ✓ 시퀀스 다이어그램 작성 단계

- 작성 대상 선정
- 유스케이스의 액터를 파악
- 유스케이스를 실현하기 위해 참여할 클래스(객체)들을 선정
- 시간 순서에 따른 객체 간 메시지 정의
- 필요한 객체를 추가로 정의

단계	내용
대상 선정	유스케이스 다이어그램을 이용하여 시퀀스 다이어그램의 작성 대상을 선택한 후 하나의 유스케이스를 선택하고 유스케이스 정의서를 분석한다.
액터 파악	액터가 둘 이상일 경우라도 모두 좌측부터 액터를 위치시켜야 한다. 순서는 중요하지 않고 메시지 선이 적게 교차하도록 배치하는 것이 좋다.
클래스(객체) 선정	정의된 클래스 중에 유스케이스의 처리에 참여하는 것들을 식별하고 시퀀스 다이어그램에 위치시켜야 하지만 순서는 중요하지 않다.
메시지 정의	유스케이스를 실현하기 위해 필요한 객체들 간의 메시지를 정의하되, 시간 순서에 유의하도록 한다. (시간흐름은 위에서 아래로 흐름)
객체 추가	정의되지 않은 객체가 있으면 시퀀스 다이어그램에 추가하고 객체 사이의 메시지도 정의하여 추가해야 한다.

## 작성 및 주의사항 [2/2]

### ✓ 시퀀스 다이어그램 작성 시 주의사항

- 동일한 상호작용을 여러 시퀀스 다이어그램에 중복되지 않게 작성
- 복잡한 알고리즘이나 looping, conditional behavior를 보여주기에 적합하지 않음.  
이런 경우에는 activity diagram을 사용(표현방법은 제공)
- 가독성이 좋도록 적당한 코멘트를 사용
- 메시지 흐름은 액터로부터 시작되게 작성
- 클래스 다이어그램에 표기된 클래스명과 매칭 가능하도록 객체이름을 표기

기본적으로 시퀀스 다이어그램은 유스케이스 별로 하나씩 작성해야 한다. 하지만 경우에 따라서 하나의 유스케이스에 여러 개의 시퀀스 다이어그램을 작성하거나 여러 유스케이스에서 공통으로 사용하는 상호작용을 하나의 시퀀스 다이어그램으로 작성할 수도 있다.



## 4. 행위 다이어그램

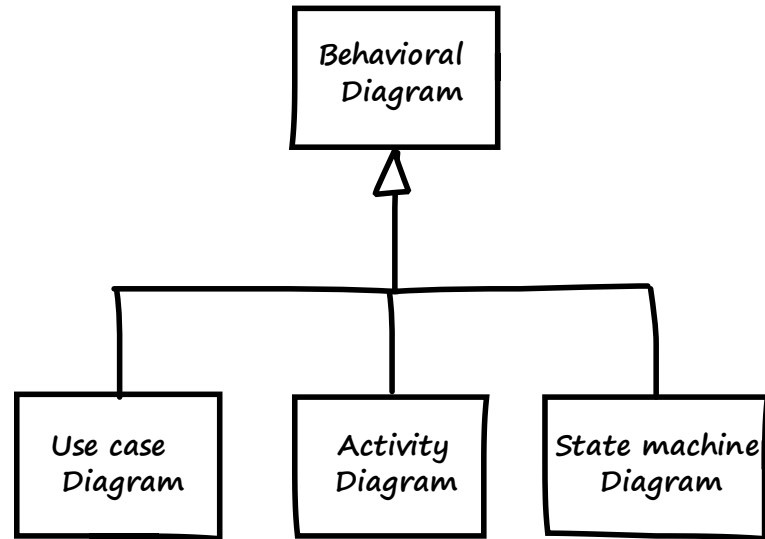
---

- 행위 다이어그램 개요
- 액티비티 다이어그램

# 행위 다이어그램 개요

## ✓ 행위 다이어그램(Behavioral Diagram)

- 요청에 따른 시스템의 반응을 보여주고 시간의 흐름에 따른 시스템의 진화를 보여주기 위하여 사용함
- UML 2.x 의 행위 다이어그램
  - Use case diagram
  - Activity diagram
  - State machine diagram





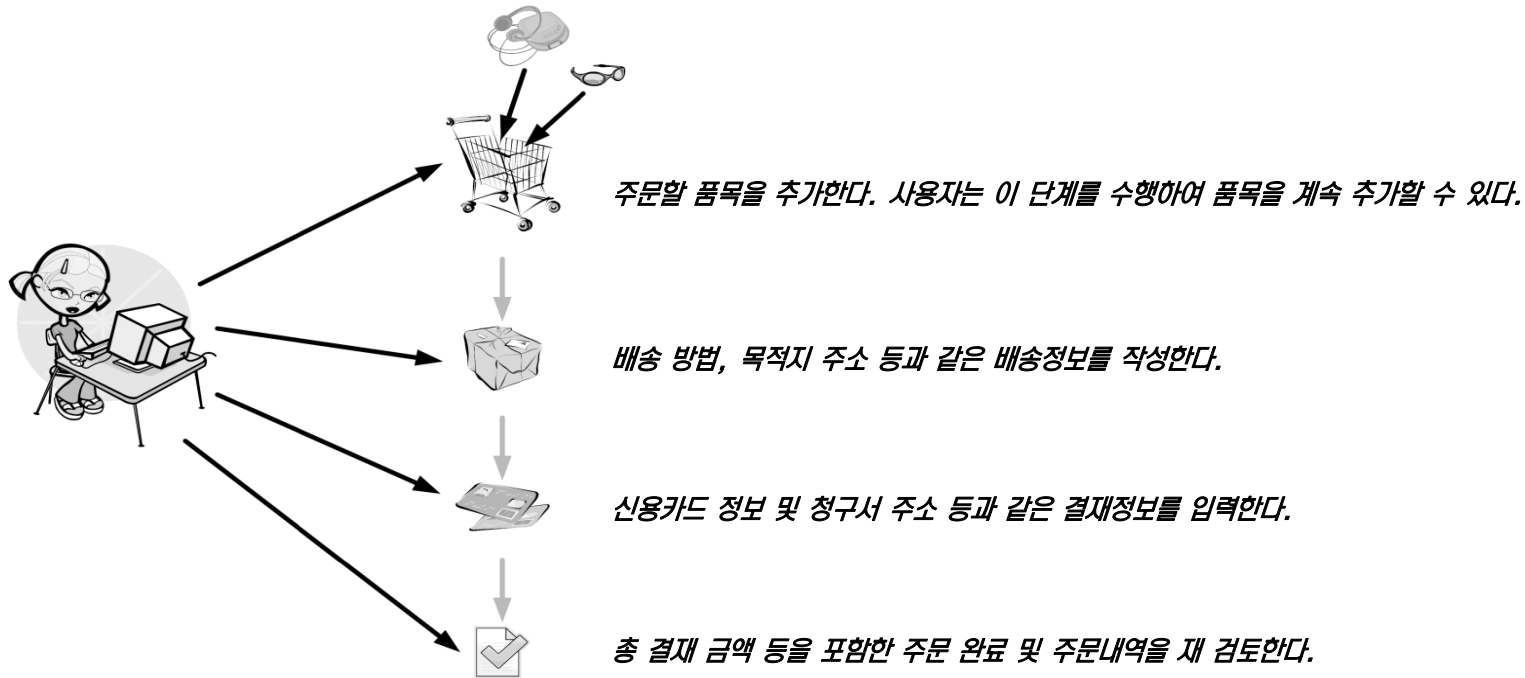
# 액티비티 다이어그램 *Activity Diagram*

## ✓ 정의

- 순차 로직, 업무 절차 그리고 워크 플로우를 기술하는 방법
- 플로우 차트와 유사하나 액티비티 다이어그램은 병렬 행동을 지원함

## ✓ 메타포 *Metaphor*

- 장바구니 기능을 이용한 물품구매

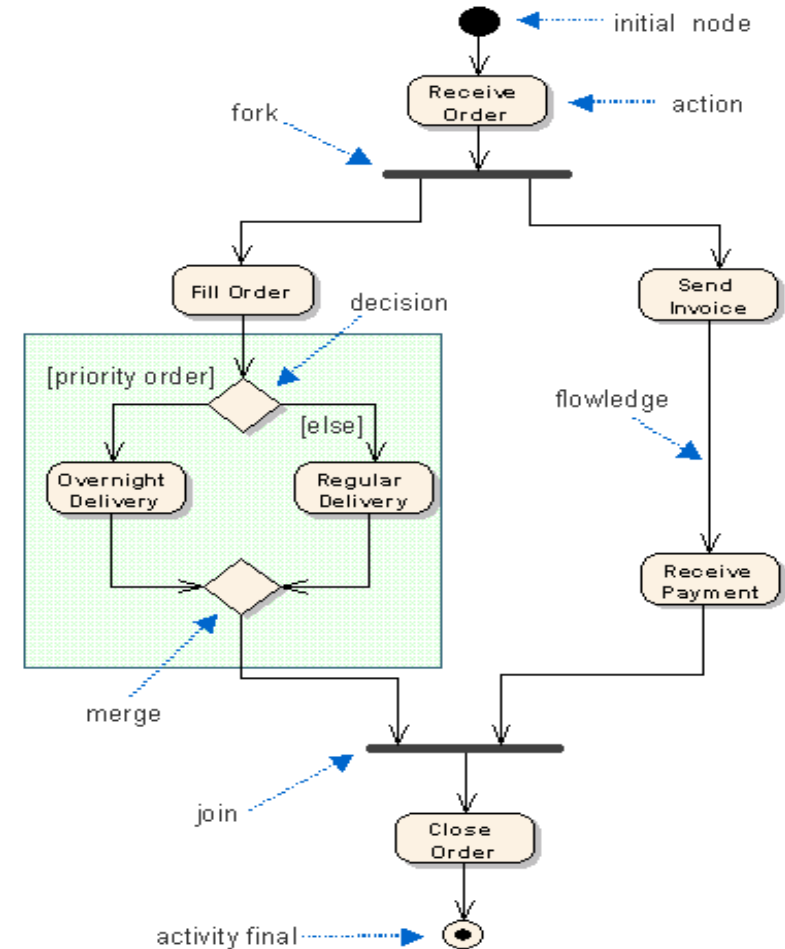




# 액티비티 다이어그램 개요 (1/3)

## ✓ 액티비티 다이어그램의 특징

- 오퍼레이션이나 처리과정이 수행되는 동안 일어나는 일들을 간단하게 하기 위해 고안
- 복잡한 알고리즘을 나타내기에 적합
- 각각의 활동은 가늘고 모서리가 둥근 사각형으로 나타냄
- 시작점과 종료점도 가지고 있으며, 상태 다이어그램과 동일



# 액티비티 다이어그램 개요 [2/3]

---

## ✓ 작성목적

- 대상에 상관없이 처리 순서를 표현하기 위해 작성
- 비즈니스 프로세스를 정의
- 유스케이스 실현
- 프로그램 로직을 정의

## ✓ 작성시기

- 업무 프로세스 정의 시점
- 유스케이스 정의 작성 시점
- 오퍼레이션 정의 시점
- 기타 처리흐름이나 처리절차가 필요한 시점
- **요구사항 정의 → 분석 → 기본설계 → 상세설계 → 개발 → 구현**

# 액티비티 다이어그램 개요 [3/3]

## ✓ 플로우차트와 액티비티 다이어그램

- 플로우차트: 순차처리만 표현
- 액티비티 다이어그램: 병렬처리 역시 표현

## ✓ 상호작용 다이어그램과 액티비티 다이어그램

- 상호작용 다이어그램은 객체에서 객체로의 제어흐름 강조
  - 상호작용 다이어그램에는 시퀀스 다이어그램과 커뮤니케이션 다이어그램이 있음
- 액티비티 다이어그램은 액션에서 액션으로의 제어흐름 강조

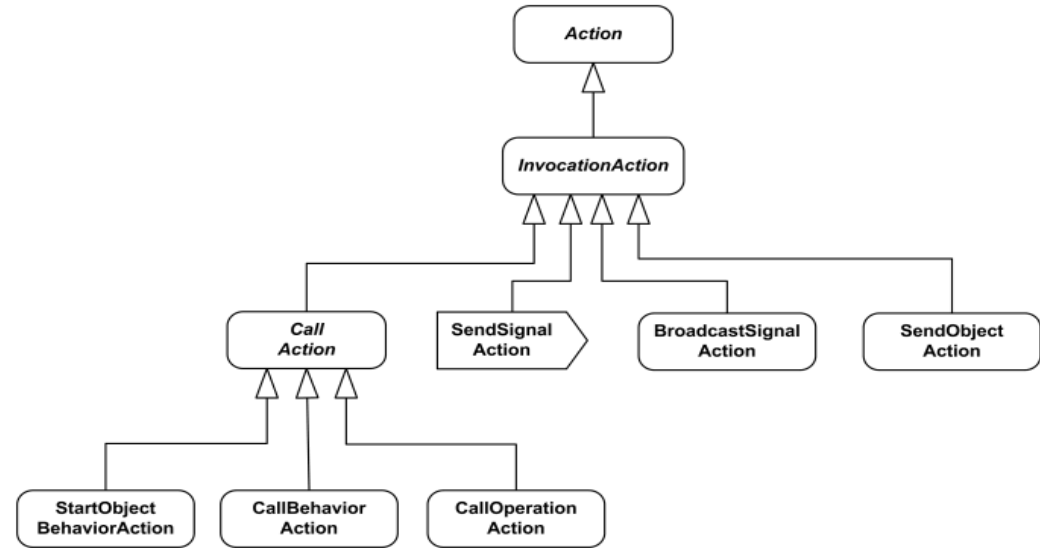
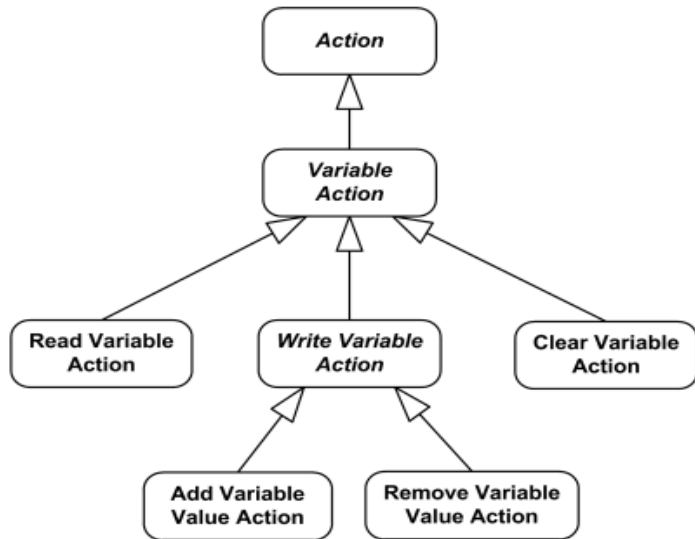
## ✓ 상호작용 개요 다이어그램과 액티비티 다이어그램

- 상호작용 개요 다이어그램은 액티비티 다이어그램과 상호작용 다이어그램의 조합을 의미
- 상호작용 개요 다이어그램은 전체로 보면 하나의 액티비티 다이어그램인데 각각의 활동 내부를 보면 상호작용 다이어그램으로 이뤄짐

# 액션 Action

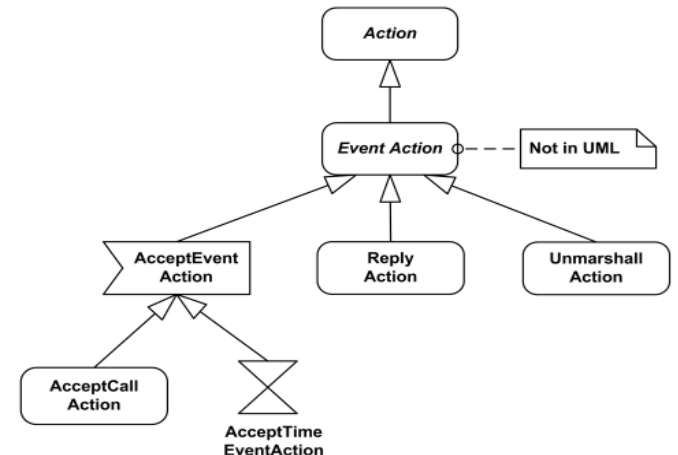
## ✓ Activity를 구성하는 액션 Action

- UML 1: 액티비티(activity), 분기(branch)
- UML 2: 액션(action), 결정(decision)
  - Object Action
  - Variable Action
  - Invocation Action
  - Event Action



### Object actions:

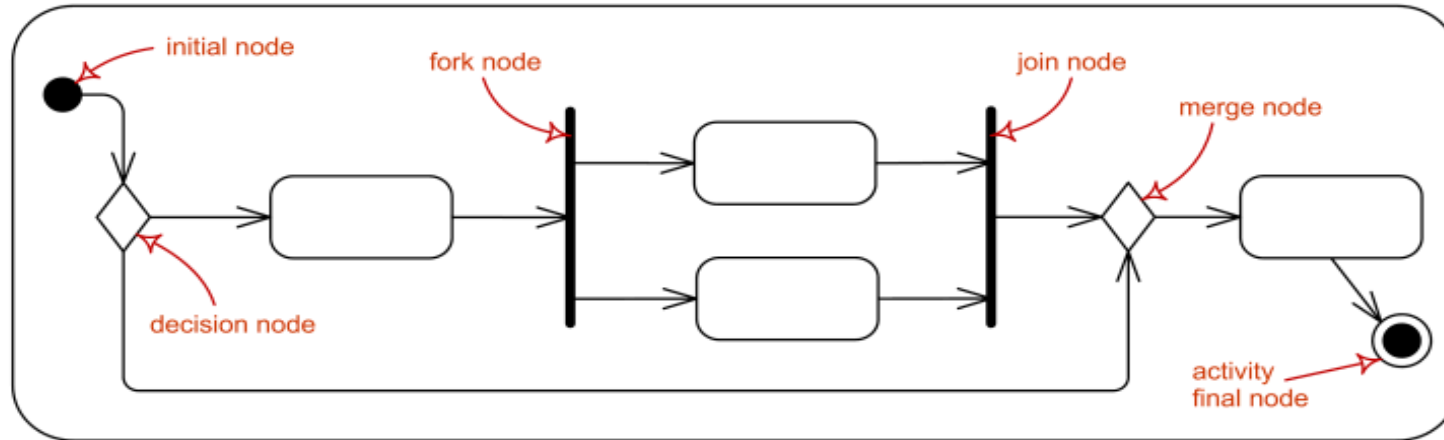
- create object action
- destroy object action
- test identity action
- read self action
- value specification action
- start classifier behavior action
- read is classified object action
- reclassify object action
- read extend action



# 제어노드 *Control node*

## ✓ 제어 노드 control node

- 초기노드 initial node : 시작시점
- 마지막 흐름 노드 Flow final : 단일 flow의 완료 시
- 마지막 노드 activity final node : 완료 시점
- 결정 노드 decision node
- 병합 노드 merge node
- 포크 노드 fork node
- 조인 노드 join node



*Activity control nodes overview.*

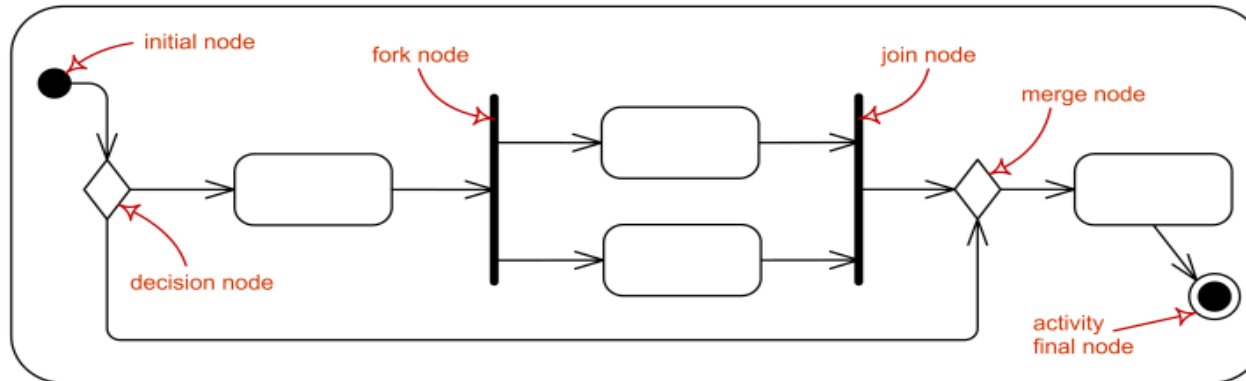
# 조건행동과 병렬행동

## ✓ 조건행동

- 결정(decision) : 한 개의 들어오는 플로우와 가드를 가진 몇 개의 나가는 플로우를 가짐. 병합의 시작
- 병합(merge) : 여러 개의 들어오는 플로우와 한 개의 나가는 플로우를 가짐. 결정의 마무리

## ✓ 병렬행동

- 포크(Fork) : 하나로 들어온 플로우와 여러 개로 나가는 동시 플로우를 가짐. 조인의 시작
- 조인(Join) : 여러 플로우와 하나의 플로우를 가짐. 포크의 마무리



Activity control nodes overview.

# 파티션 Partition

## ✓ 일반적 이슈

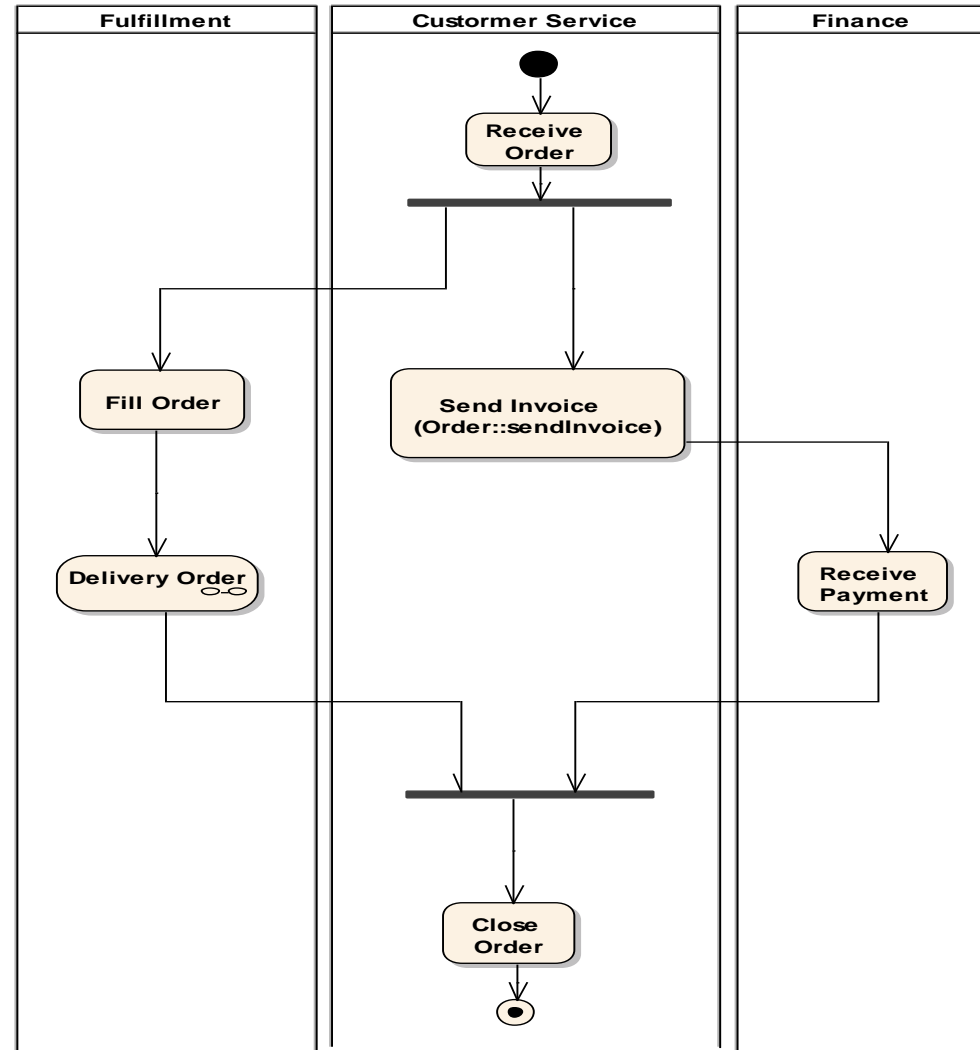
- 액티비티 다이어그램은 어떠한 액션을 수행하는지는 알 수 있지만 누가 수행하는지 모호
- 프로그래밍 관점에서 어느 클래스가 어떤 액션을 수행하는지 그 책임을 알 수 없음
- 비즈니스 프로세스 모델링 관점에서 각 액션을 책임지는 사람이나 부서를 알 수 없음

## ✓ 해결책

- 각 액션의 책임 클래스 또는 역할을 표기
- 파티션을 사용하여 책임성 부여

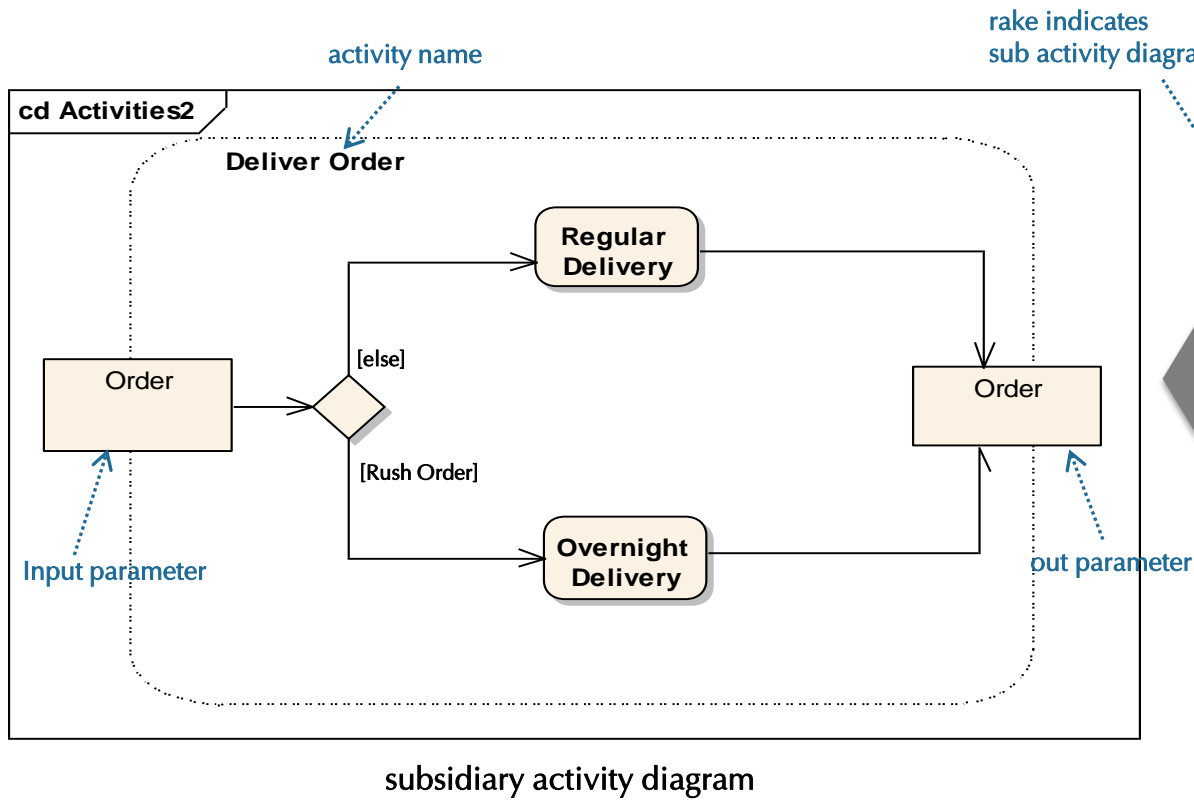
## ✓ 표기법

- UML 1.x : swimlanes
- UML 2.x : partitions

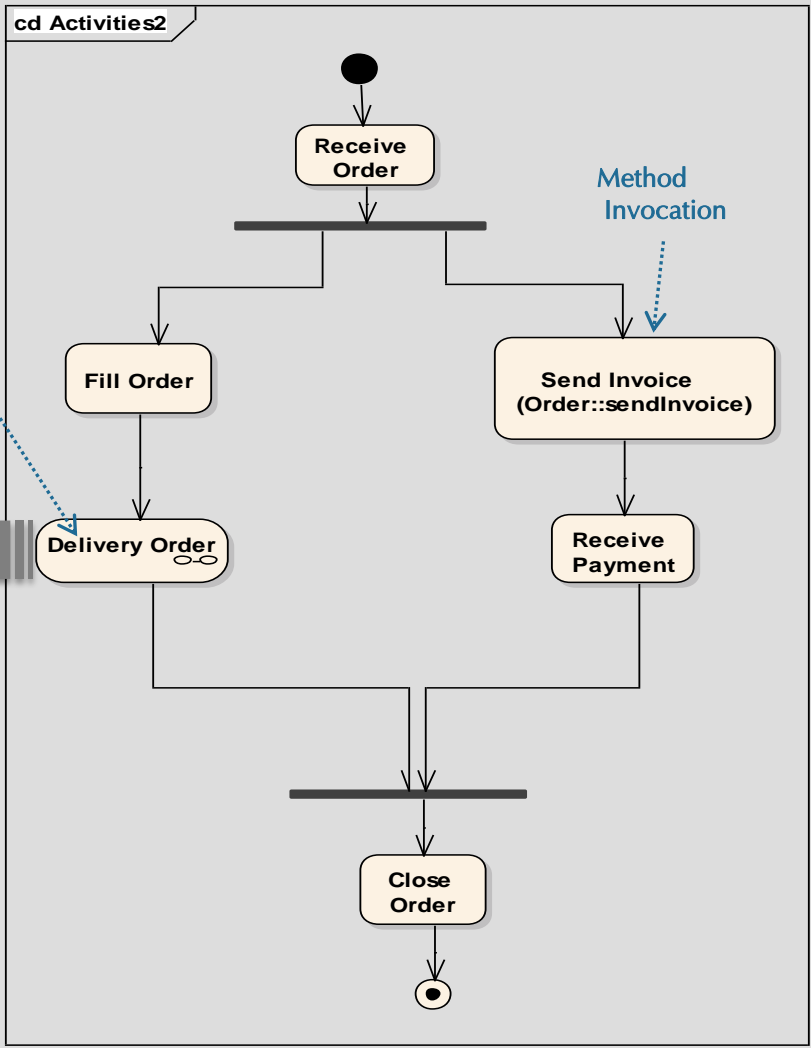


# 액션의 분해

✓ 액션은 서브 액티비티로 분해될 수 있음



rake indicates  
sub activity diagram

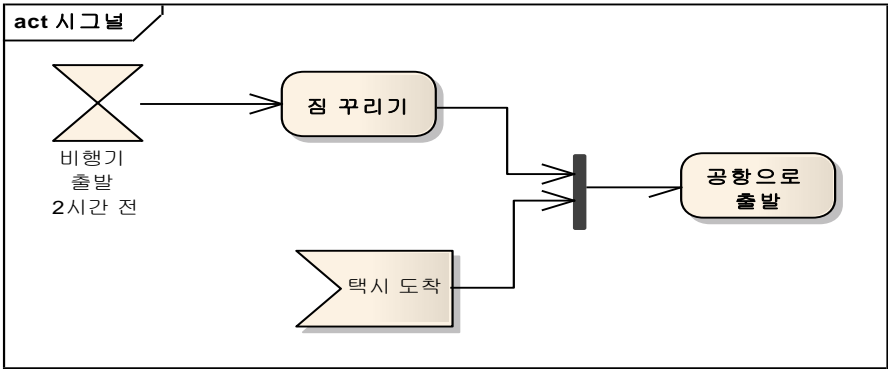




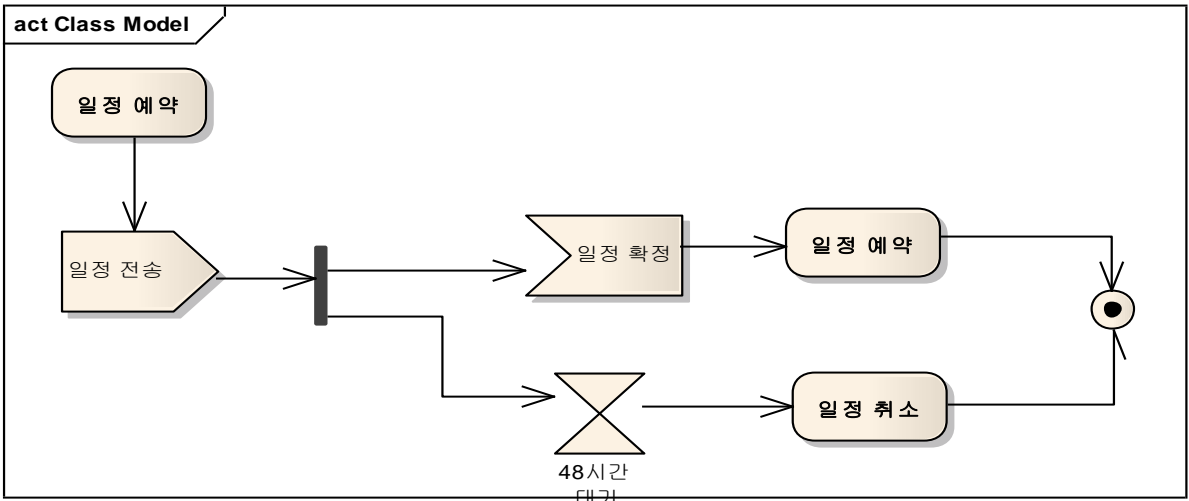
# 시간 시그널 *Time signal*

✓ 시간 시그널 *Time signal*

- 시간의 흐름에 따라서 발생



- 시그널은 액티비티가 외부의 프로세스로부터 이벤트를 수신함을 표현



# 작성 및 주의사항 [1/2]

## ✓ 액티비티 다이어그램 작성 단계

- 작성 대상을 선정
- 필요한 경우 파티션 정의
- 액티비티를 사용하여 처리절차 모델링

단계	내용
작성대상 선정	액티비티 다이어그램의 작성 대상을 선정한다. 대부분의 경우 액티비티 다이어그램은 업무 프로세스를 모델링 하거나, 오퍼레이션 사용을 정의하는 용도로 사용된다.
파티션 정의	대상영역에 명확한 역할을 정의할 수 있을 경우, 역할을 식별하여 파티션으로 표현한다. 파티션은 필수적으로 정의해야하는 대상은 아니다.
처리절차 모델링	처리절차를 모델링 할 경우 시작점과 끝점이 표현되어야 하고 처리흐름이 도중에 끊겨 미야상태가 되지 않아야 한다.

## 작성 및 주의사항 [2/2]

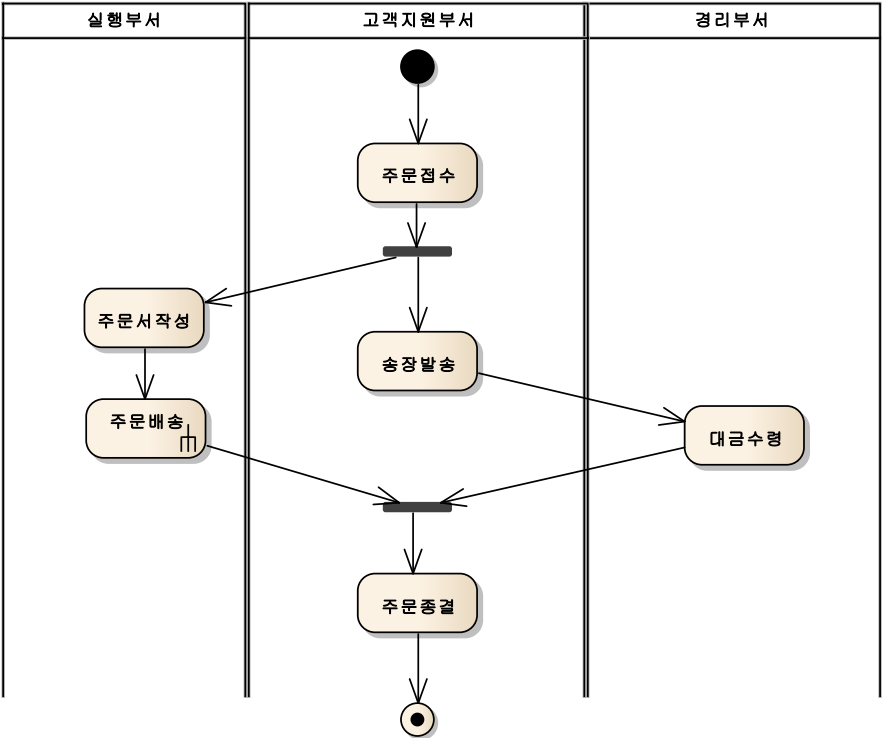
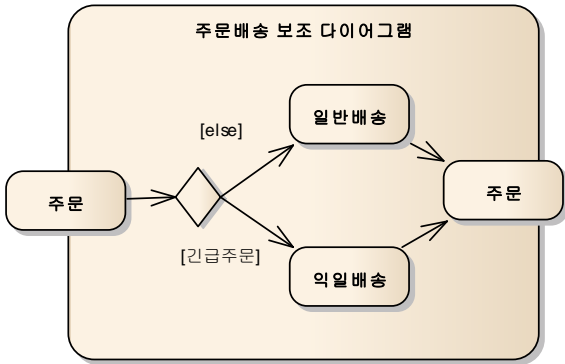
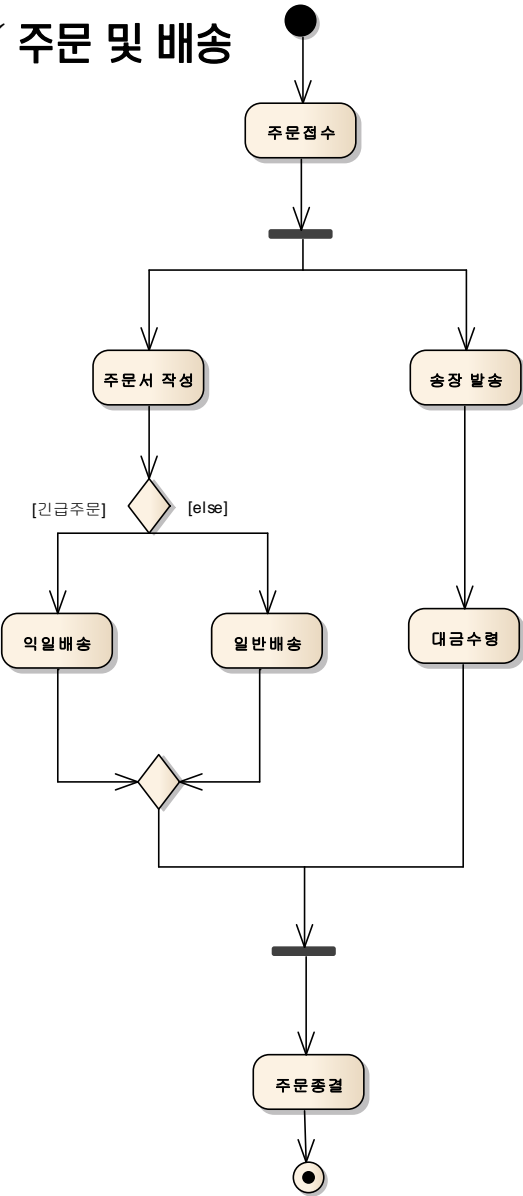
---

### ✓ 액티비티 다이어그램의 작성 시 주의사항

- 해당 부분을 이해하는데 필수적인 요소들만 표현
- 추상화 수준에 맞는 상세성을 일관되게 제공
- 중요한 의미를 이해하기 적절한 단위로 표현
- 목적을 전달할 수 있는 명칭의 부여
- 주 흐름으로부터 시작하여 전이, 분기, 동시성을 표현
- 교차선이 최소화하도록 요소를 배치
- 중요한 부분은 노트, 색 등을 이용하여 시각적 효과를 사용

# 액티비티 다이어그램 실습

✓ 주문 및 배송



# 토의

---

- ✓ 질의 응답
- ✓ 토론

감사합니다...