



UML

목차

- ✓ Chap01. UML 개념
- ✓ Chap02. 유스케이스 다이어그램
- ✓ Chap03. 클래스 다이어그램
- ✓ Chap04. 시퀀스 다이어그램

Chap01.

UML 개념

▶ 모델링과 UML

✓ 모델링이란

말 그대로 모델을 만드는 작업을 뜻함. 즉, 현실 세계를 단순화 시켜 표현하는 기법

아래의 소프트웨어 개발 프로세스를 보면 요구사항, 분석, 설계 단계를 모델링 단계라고 함

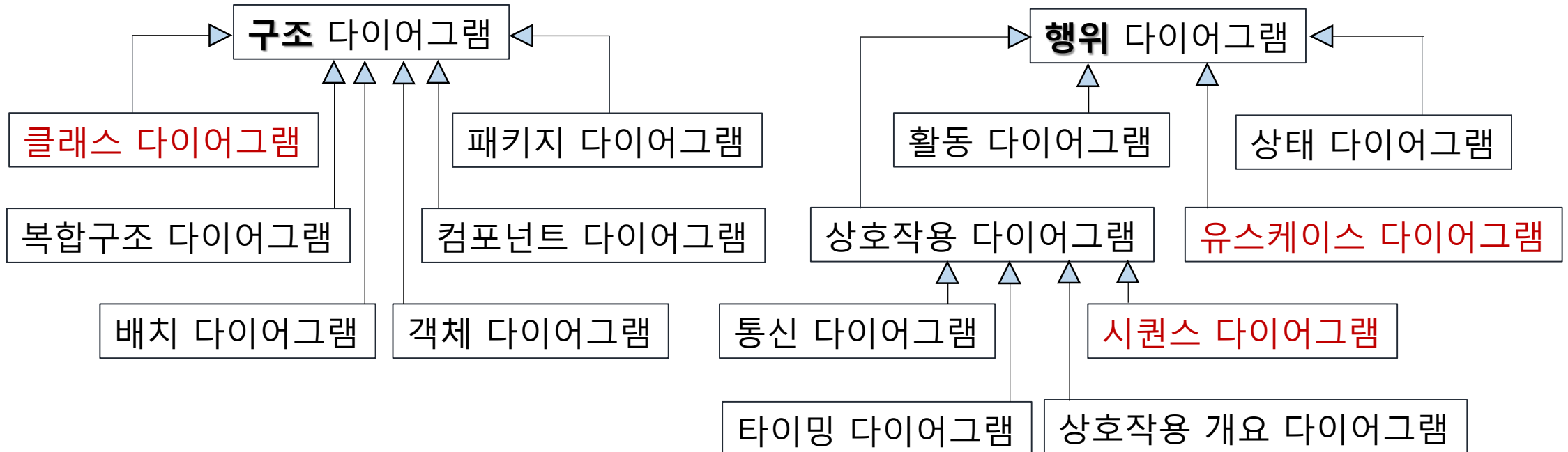


► UML

✓ UML이란

통합 모델링 언어(UML, Unified Modeling Language)는 소프트웨어 공학에서 사용되는 표준화된 범용 모델링 언어로 소프트웨어 개념을 다이어그램으로 그리기 위해 사용하는 시각적인 표기법

✓ UML 다이어그램 종류



▶ 모델링과 UML

✓ 소프트웨어(프로그램) 개발 프로세스



패키지 다이어그램

유스케이스
다이어그램

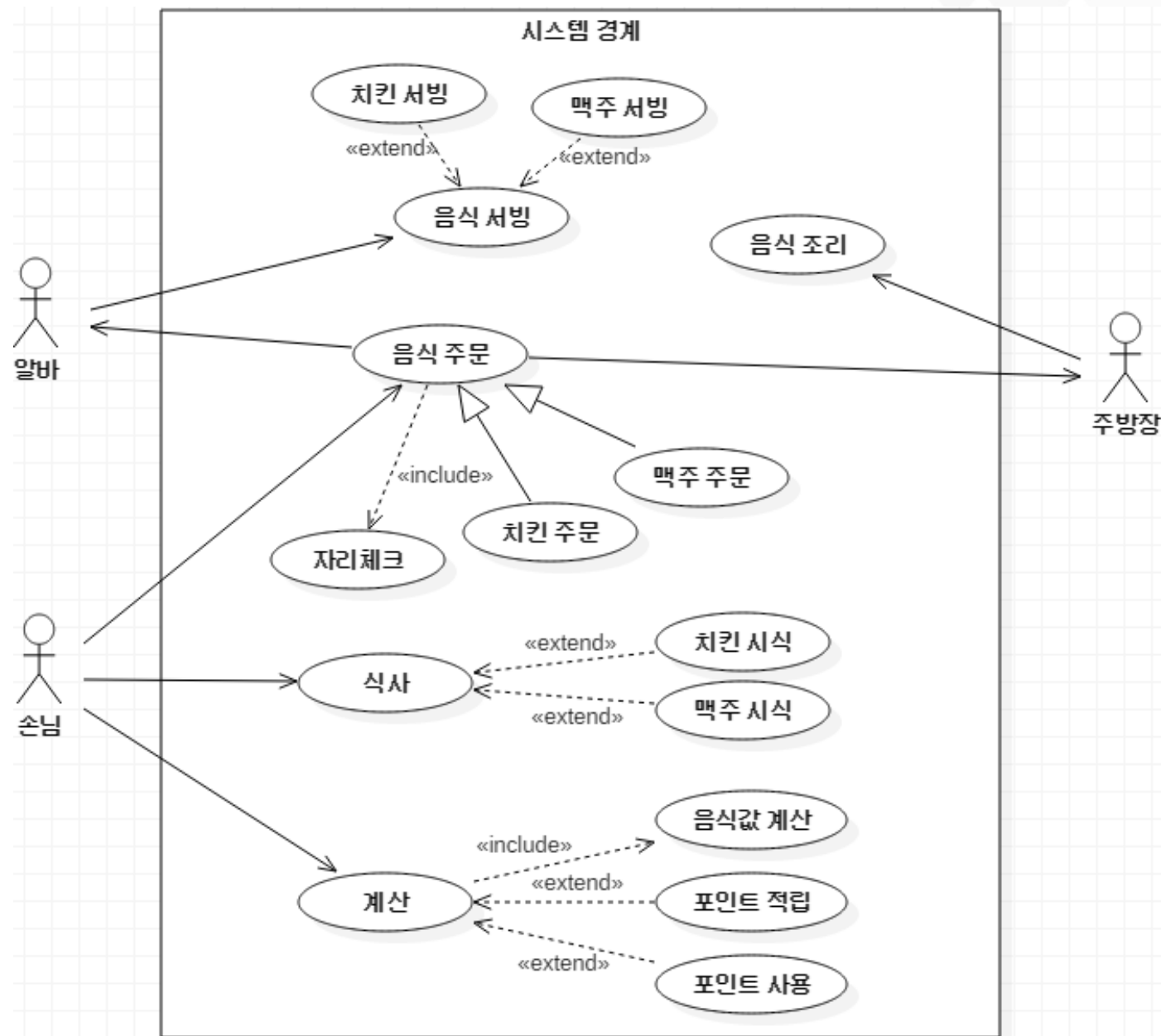
클래스 다이어그램
객체 다이어그램

상태 다이어그램
활동 다이어그램
상호작용 다이어그램

컴포넌트
복합구조
배치
다이어그램

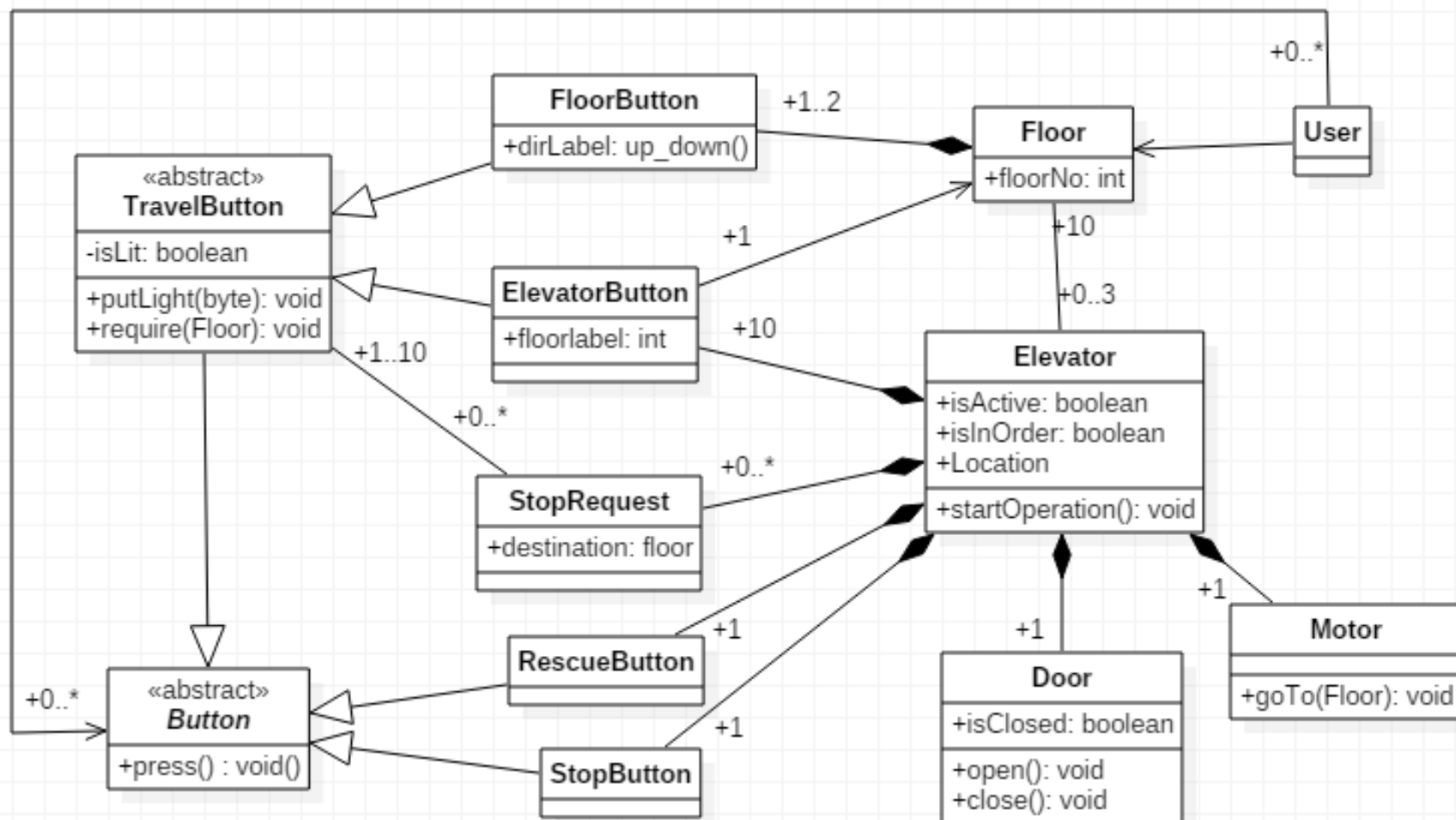
▶ 모델링과 UML

✓ 유스케이스 다이어그램



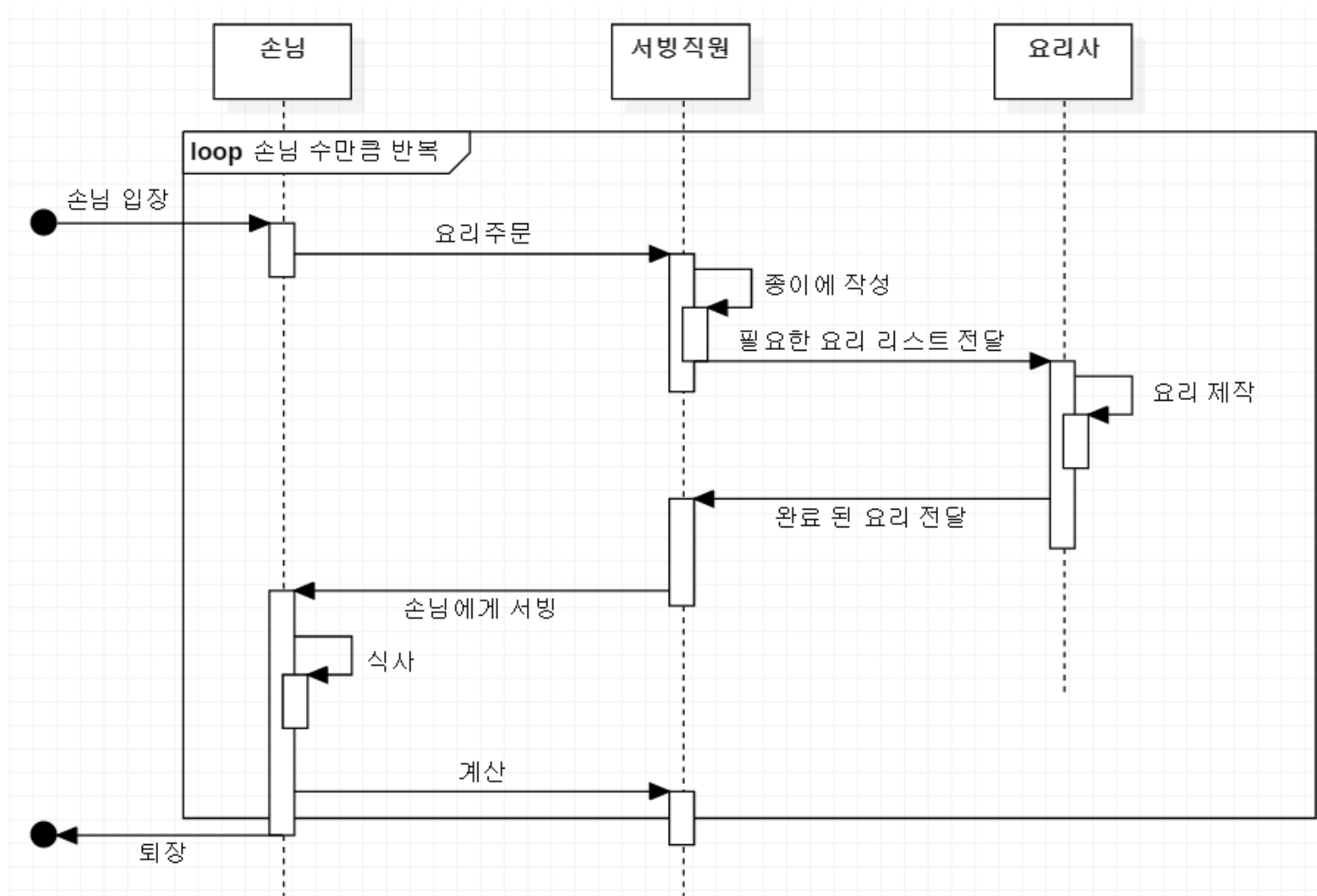
▶ 모델링과 UML

✓ 클래스 다이어그램



▶ 모델링과 UML

✓ 시퀀스 다이어그램



▶ UML의 V 프로세스

✓ 기능모델링

✓ 동적모델링

✓ 정보모델링

행위자

유스케이스

유스케이스
시나리오

유스케이스
시나리오의
정보

블랙박스 분석

요구사항 명세서

액티비티
다이어그램

시퀀스
다이어그램

클래스
다이어그램

화이트박스 분석



▶ UML 툴 설치

✓ StarUML 5.0 설치

UML 툴 중에서 가장 보편화 되어 있는 프로그램

1. <https://sourceforge.net/projects/staruml/files/staruml/5.0/>
2. Download Lastet Version 클릭
3. exe 설치
4. Next> 경로 설정하고 Next> ...



▶ UML 툴 활용

✓ StarUML 5.0 활용

1. New Project By Approach
Empty Project – Ok
2. 우측 Untitled 우클릭 – Add - Model
(Default로 Model1이라는 Model이 만들어짐)
3. Model1 우클릭 – Add Diagram 후
원하는 다이어그램 선택
4. 좌측에 생기는 기호를 끌어다가 쓰면서 다이어그램 완성

Chap02.

유스케이스 다이어그램

▶ 요구사항

✓ 요구사항이란

고객 및 소프트웨어 개발에 관계된 사람들이 시스템 개발에 앞서 개발되는 프로그램에 필요한 조건이나 능력을 말함

✓ 요구사항 프로세스



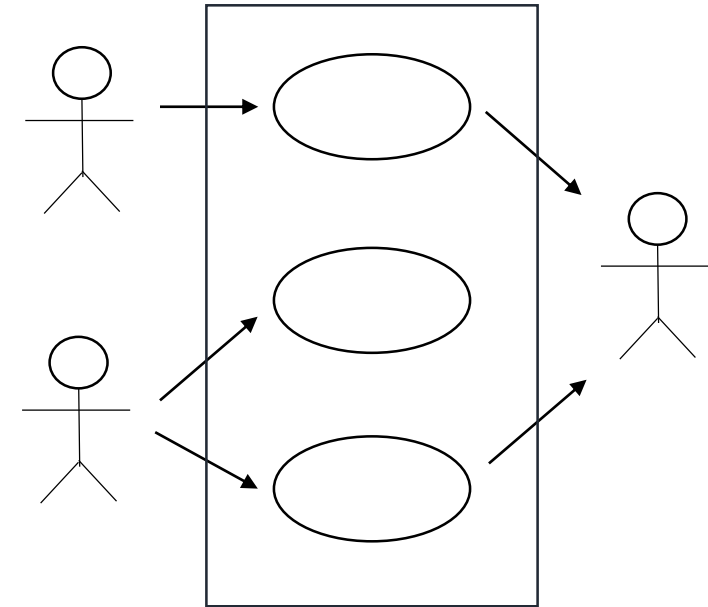
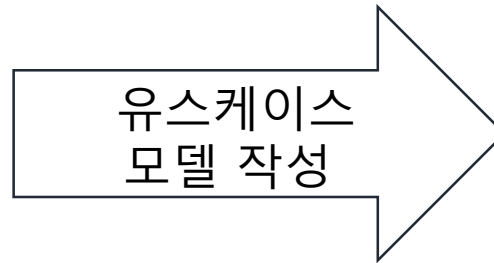
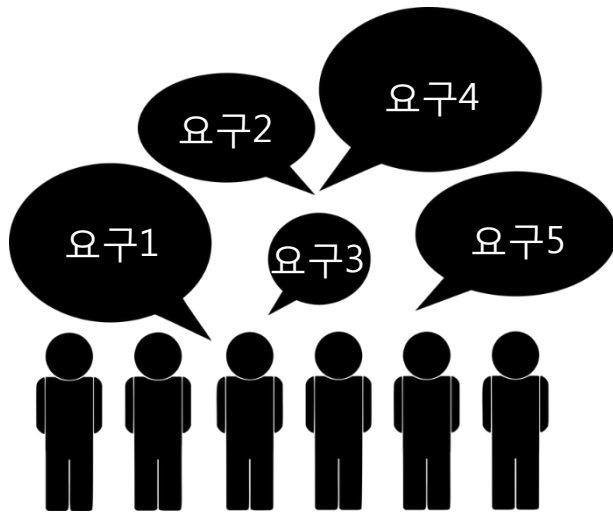
✓ 요구사항 조건

- 명확성 : 기술된 요구사항은 항상 동일한 의미로 해석되어야 함 → 모호하지 않아야 함
- 완전성 : 사용자가 기대하는 모든 요구사항이 기술되어야 함 → 누락되어서는 안 됨
- 일관성 : 서로 상충되는 요구사항이 있어서는 안 됨
- 검증 가능성 : 객관적으로 검증할 수 있도록 구체적이어야 함

▶ 유스케이스 다이어그램

✓ 유스케이스 다이어그램

동적(행위) 다이어그램으로 시스템 내의 활동들의 흐름을 보여줌
여러 업무 프로세스를 설명하는데 자주 활용

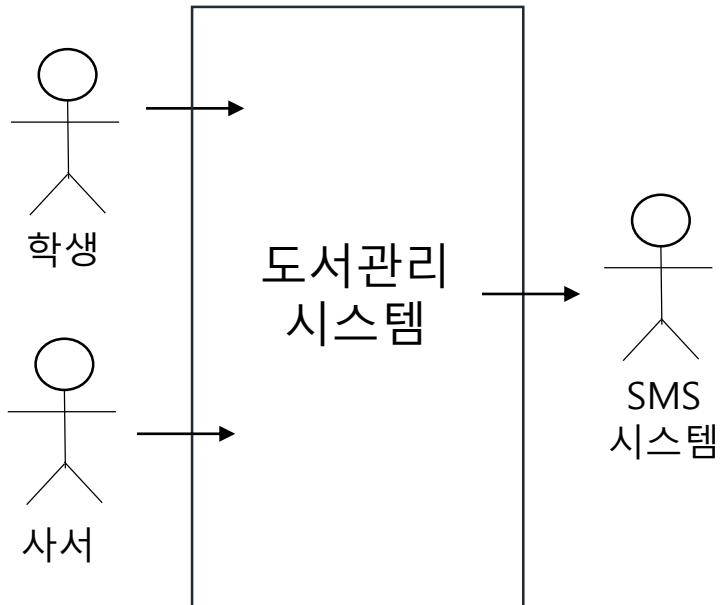


▶ 유스케이스 다이어그램

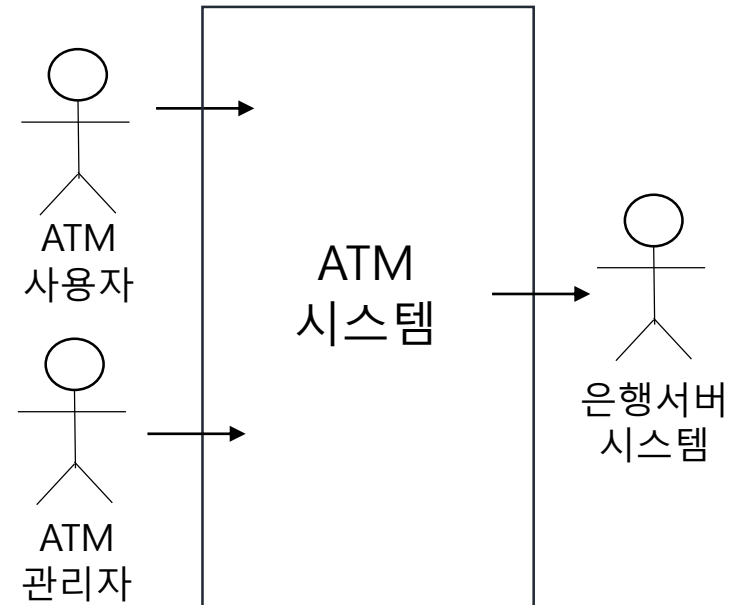
✓ 액터

시스템과 상호작용을 하는 시스템 외부의 존재로 개발 대상에 따라 달라질 수 있음
시스템 관점에서 바라본 사용자의 역할을 뜻해야 함

ex) 도서관리 시스템의 액터



ex) ATM 시스템의 액터

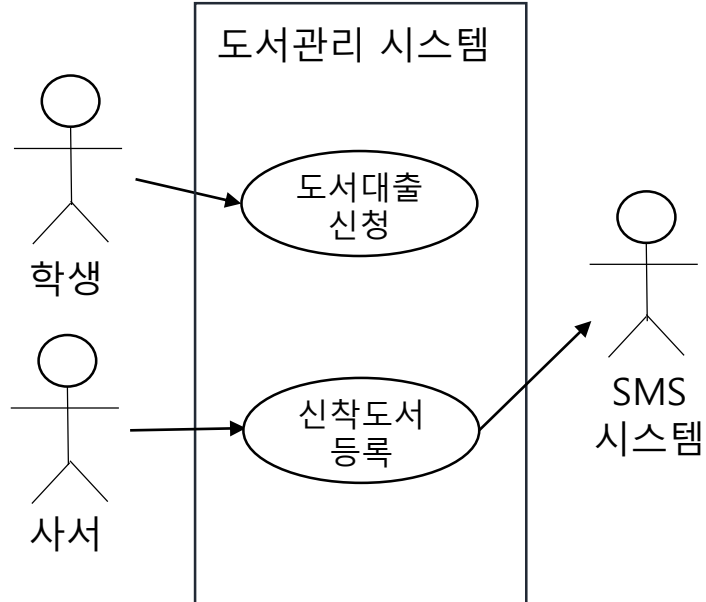


▶ 유스케이스 다이어그램

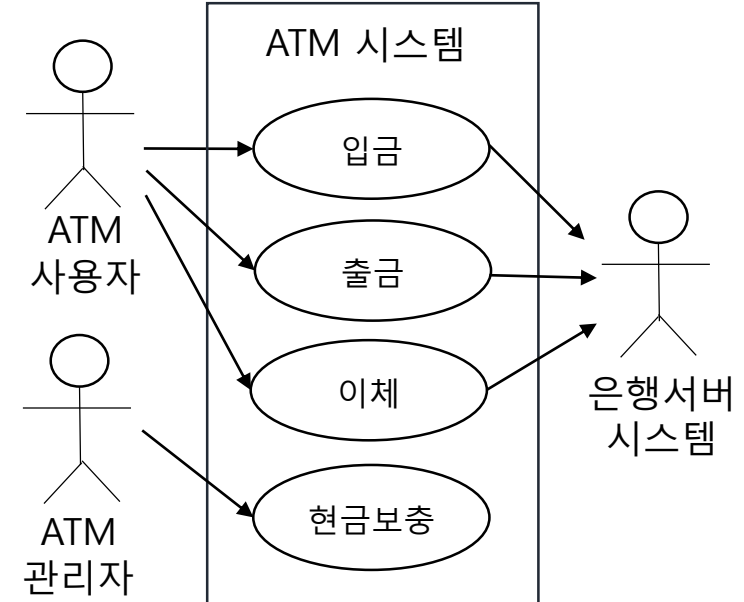
✓ 유스케이스

개발 대상이 되는 시스템이 제공하는 개별적인 기능을 뜻하는 것으로 시스템 동작 하나의 기술, 사용자가 인지할 수 있는(눈에 보이는) 하나의 기능 단위

ex) 도서관리 시스템의 유스케이스

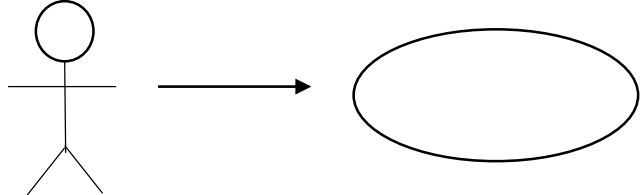
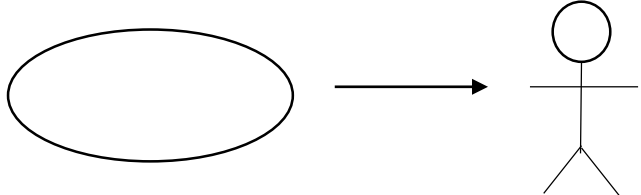
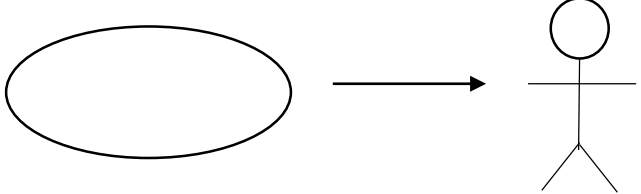


ex) ATM 시스템의 유스케이스



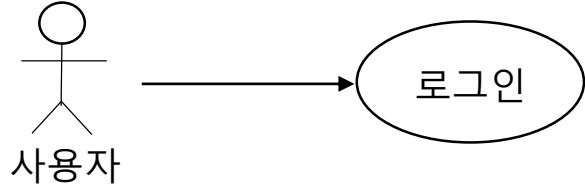
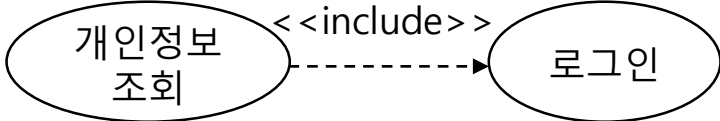
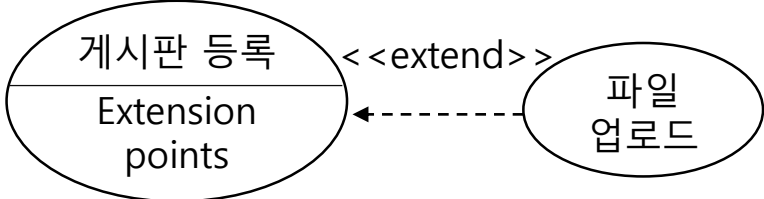
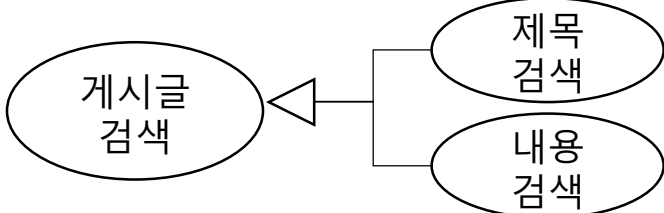
▶ 유스케이스 다이어그램

✓ 액터와 유스케이스 간의 연관 관계 방향

유형	설명	연관 관계 방향
활성화	액터가 유스케이스를 활성화 시킴	
수행결과 통보	유스케이스 결과가 액터에게 통보 됨	
외부 서비스 요청	외부 시스템에 서비스 실행을 요청함	

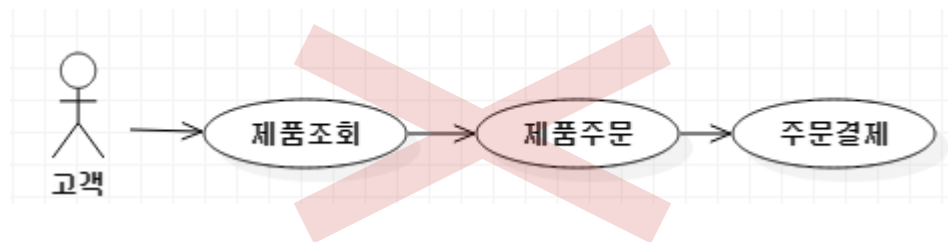
▶ 유스케이스 다이어그램

✓ 유스케이스 다이어그램 관계 종류

유형	설명	관계 방향
연관 관계	유스케이스와 액터 간 상호작용을 의미하는 관계	
포함 관계	한 유스케이스가 다른 유스케이스의 기능을 포함하는 관계 (반드시 해야만 하는 관계)	
확장 관계	기본 유스케이스에서 특정 조건이나 액터의 선택에 따라 발생하는 유스케이스 (선택적으로 할 수 있는 관계)	
일반화 관계	유사한 유스케이스들 또는 액터들을 추상화한 하나의 유스케이스로 그룹핑하여 이해도를 높인 관계	

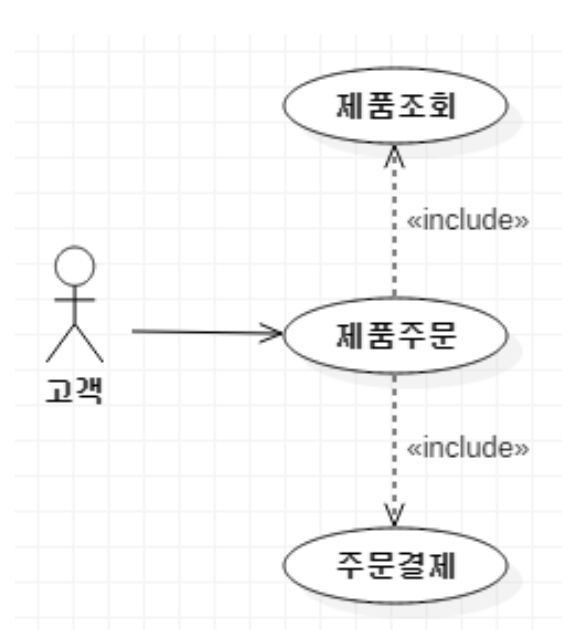
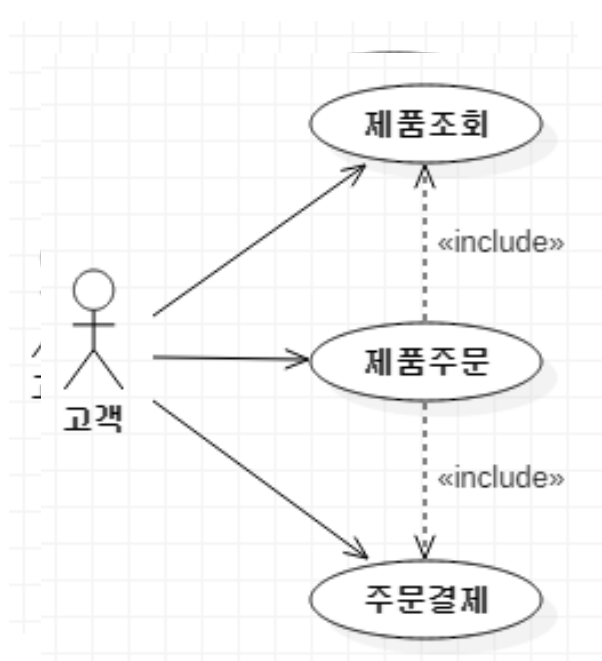
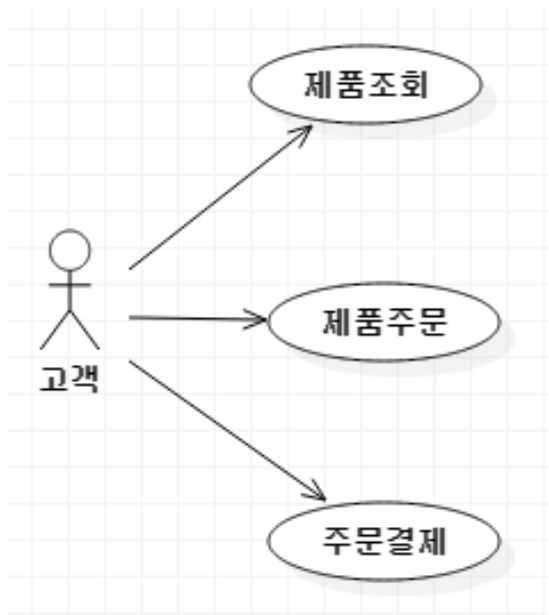
▶ 유스케이스 다이어그램 상황별 예시

✓ 시나리오상 다음과 같은 흐름 인식



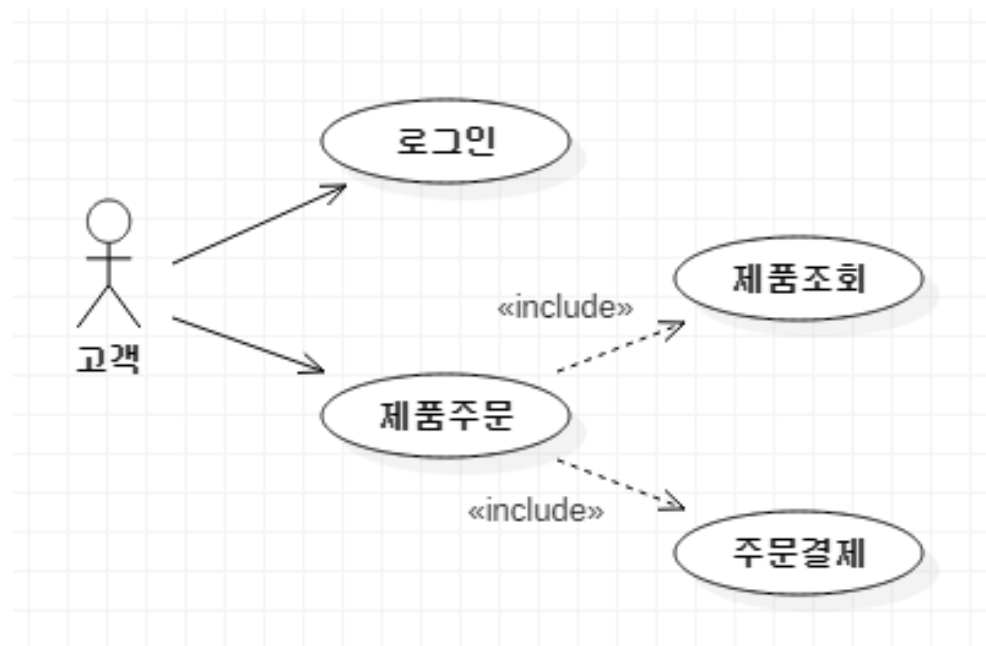
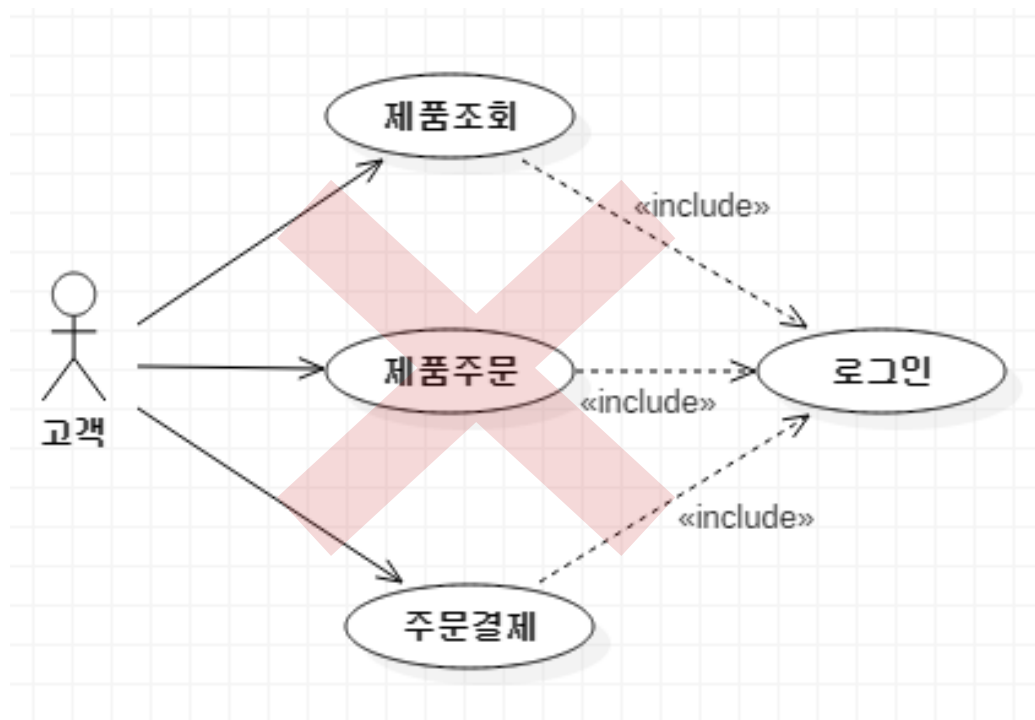
연관관계(실선)는 액터와 유스케이스 간만 사용 가능

✓ 의미에 따른 모델링



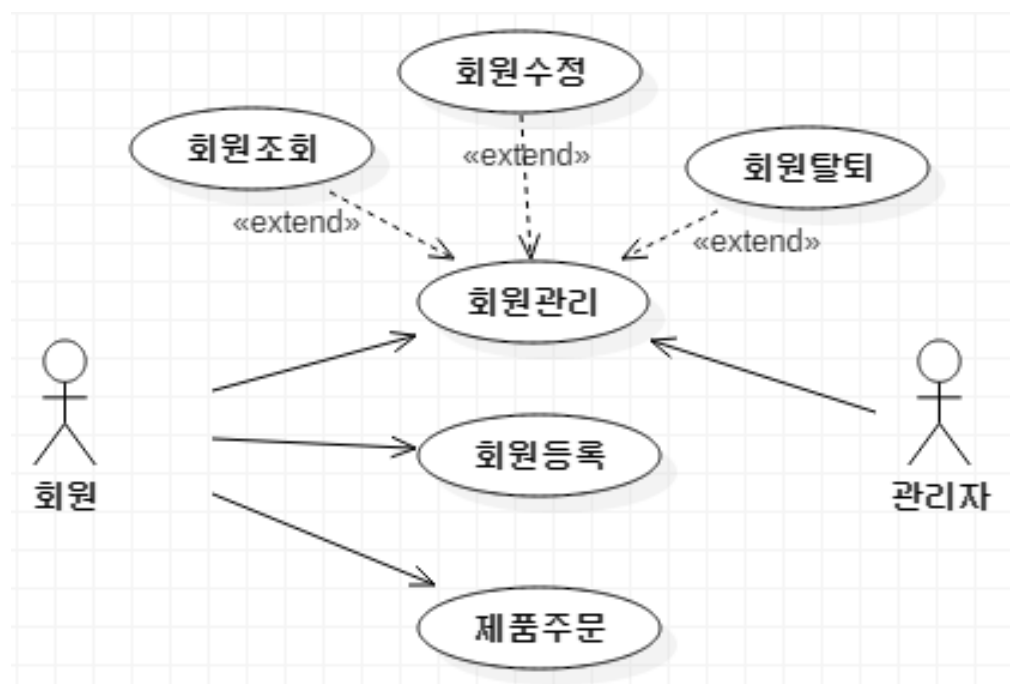
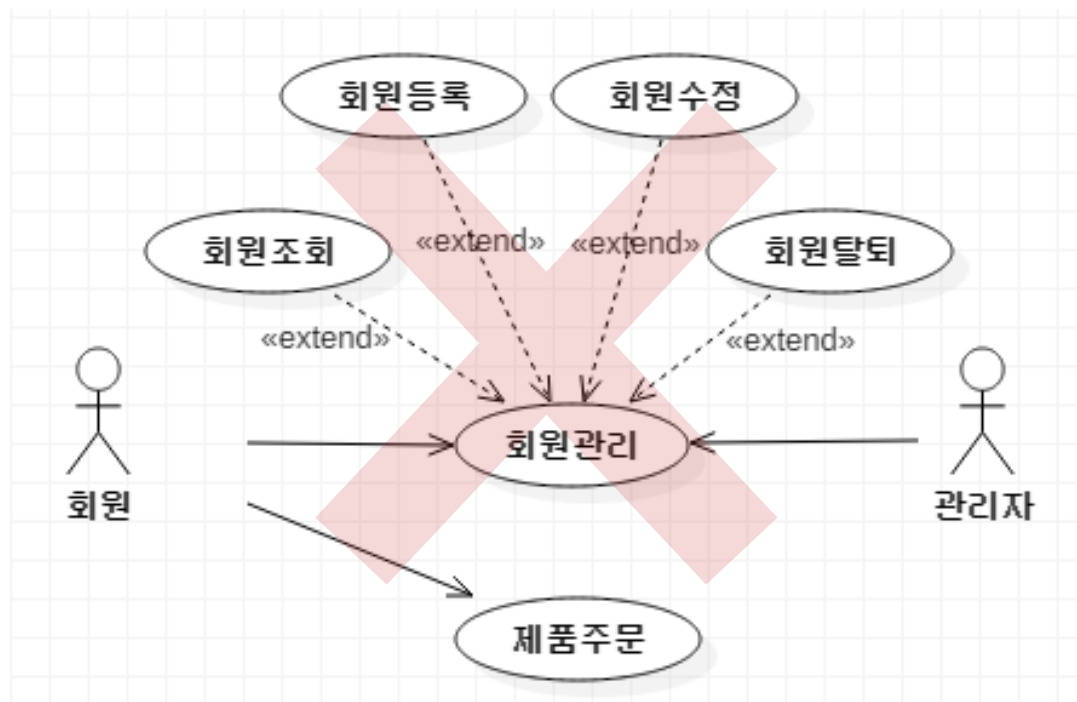
▶ 유스케이스 다이어그램 상황별 예시

✓ 로그인 유스케이스



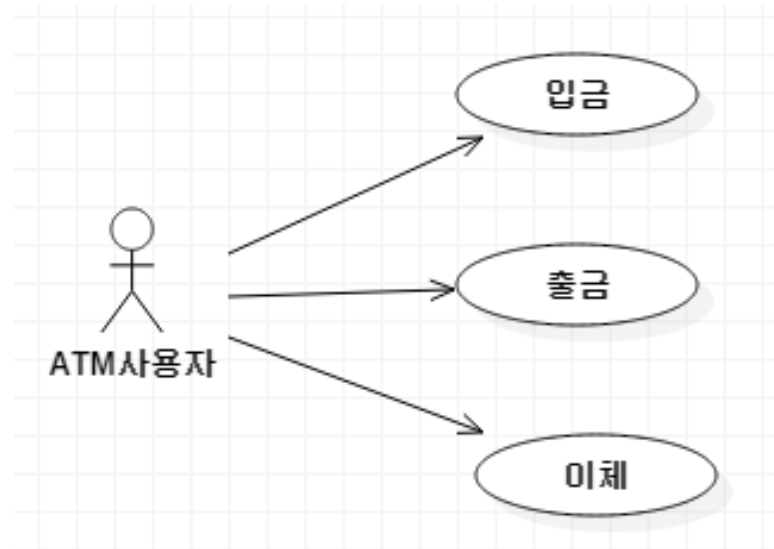
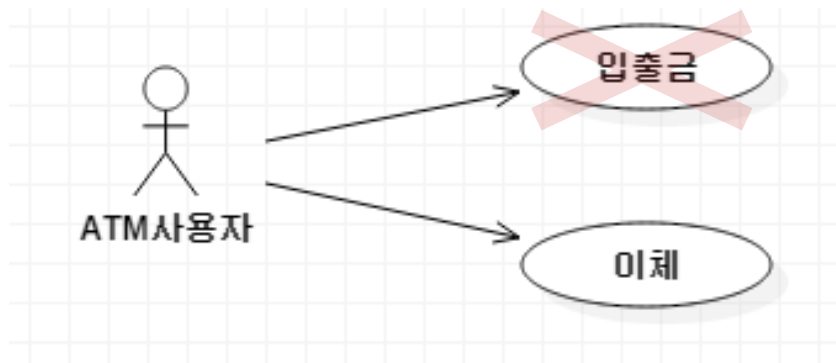
▶ 유스케이스 다이어그램 상황별 예시

✓ 유스케이스의 동일한 기능 제공

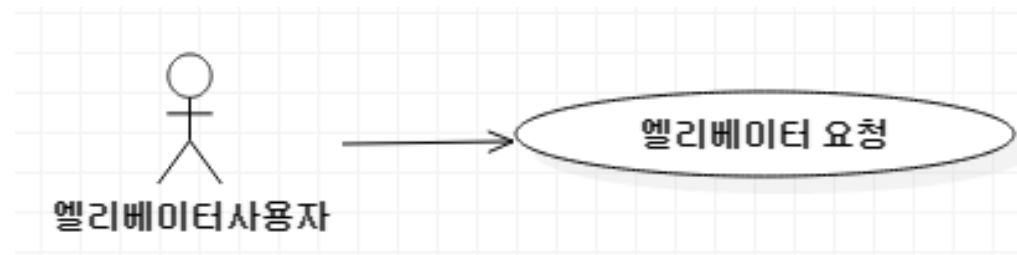
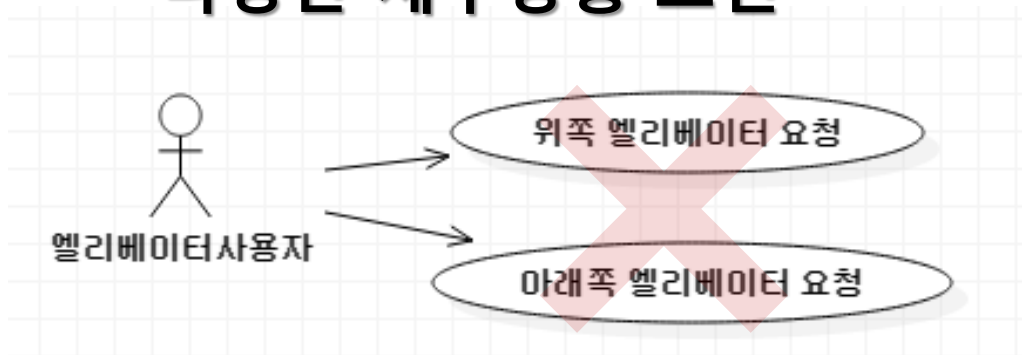


▶ 유스케이스 다이어그램 상황별 예시

✓ 유스케이스의 구체화



✓ 다양한 세부상황 표현



▶ 이벤트 흐름

✓ 기본 흐름

아무것도 잘 못 되지 않았다는 가정 하에 사용자의 자극에 시스템이 어떻게 반응하는지 기술

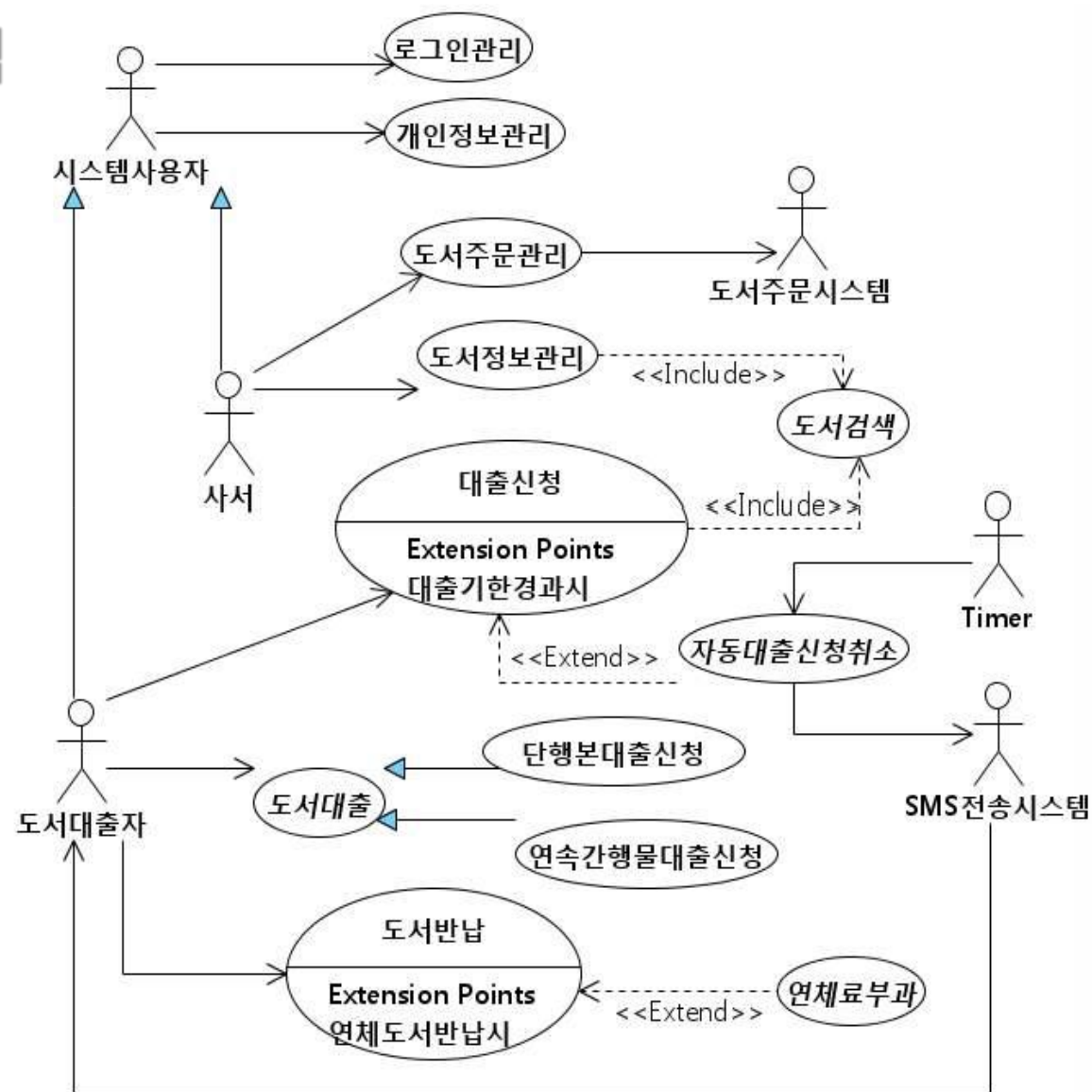
✓ 대안 흐름

세부 상황 중 일부가 일이 잘 못 되었을 경우를 고려한 흐름으로 선택흐름과 예외흐름이 존재

- 선택흐름 : 사용자 혹은 시스템에 의해 선택적으로 수행되는 흐름
- 예외흐름 : 시스템에서 발생하는 에러 등을 처리하기 위해 수행되는 흐름

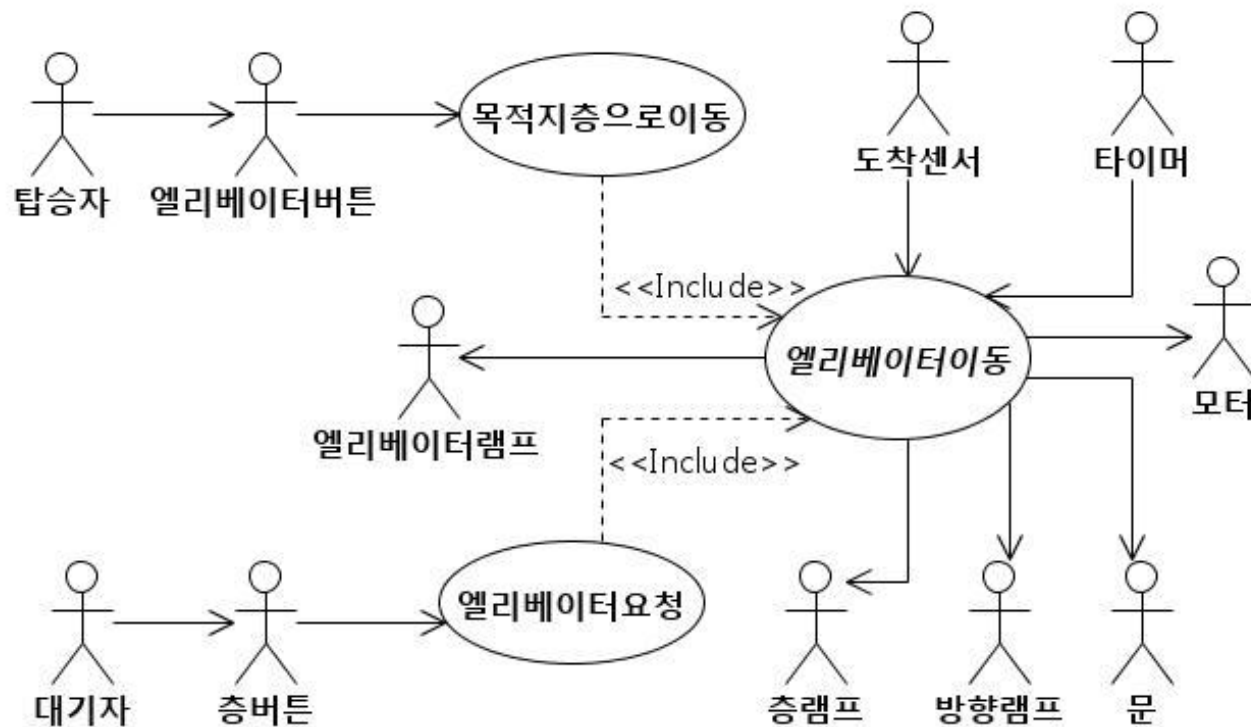
▶ 유스케이스 다이어그램 예시

✓ 도서관리 시스템



▶ 유스케이스 다이어그램 예시

✓ 엘리베이터 시스템



Chap03.

클래스 다이어그램

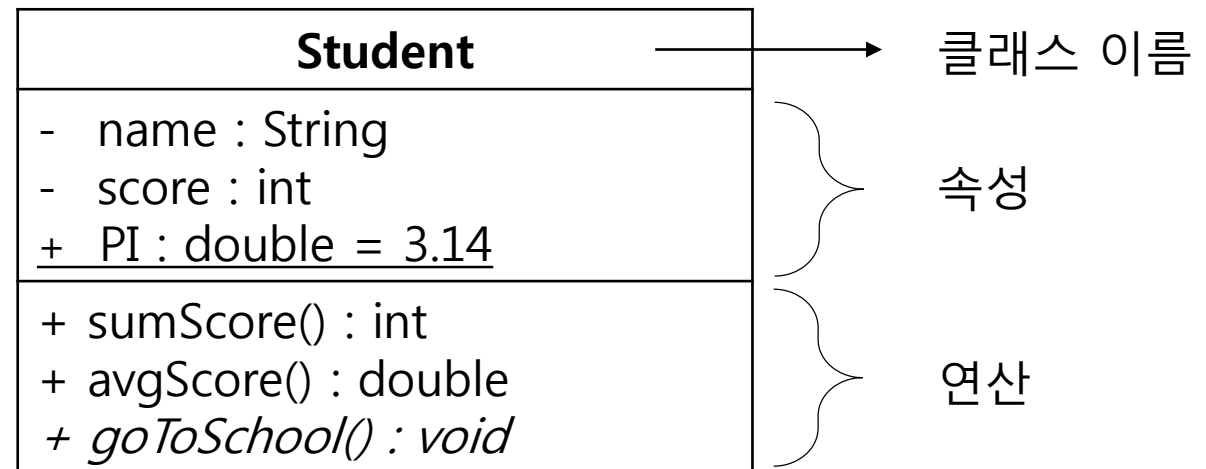
▶ 클래스 다이어그램

✓ 클래스 다이어그램


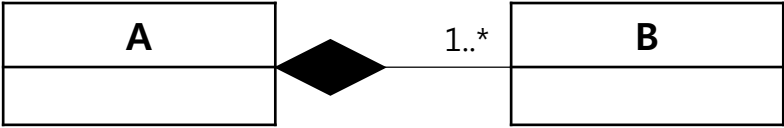
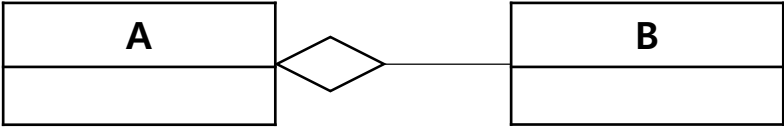
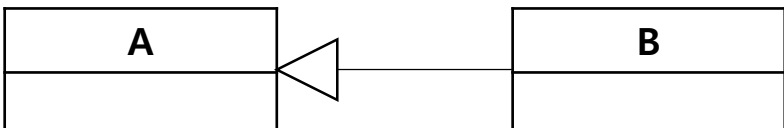
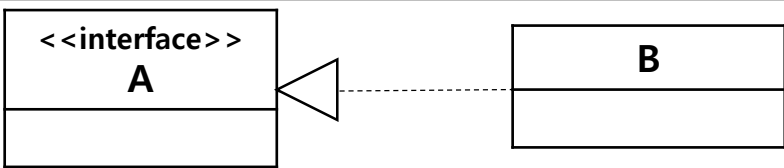

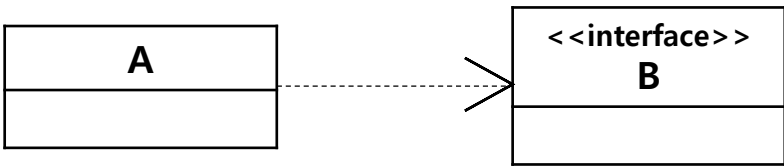
정적(구조) 다이어그램으로 UML모델링에서 가장 일반적으로 사용
시스템의 구조와 구조 간 상호 관계를 나타내며
시스템의 논리적 및 물리적 구성요소 설계 시 주로 활용

✓ 클래스의 표현

+	public
#	protected
~	default
-	private



▶ 클래스 다이어그램의 관계

관계	표기법	의미
연관 관계		클래스 A와 클래스 B는 연관 관계를 가지고 있다.
합성(포함)관계		클래스 A는 클래스 B를 한 개 이상 포함하고 있다.
집합 관계		클래스 B는 클래스 A의 부분이다.
일반화 관계		클래스 B는 클래스 A의 하위 클래스이다.
실체화 관계 (인터페이스 실현 관계)		클래스 B는 인터페이스 A를 실현한다.
의존 관계		클래스 A는 클래스 B에 의존한다.
인터페이스 의존 관계		클래스 A는 인터페이스 B에 의존한다.

▶ 연관 관계

✓ 연관 관계

한 클래스가 다른 클래스를 필드로 가지면서 클래스 간의 관련성을 뜻하는 것으로
메시지 전달의 통로 역할을 함



```
public class A{
    private B b;
}
```

```
public class B{
    private A a;
}
```

✓ 방향성이 있는 연관 관계

방향성은 메시지 전달의 방향을 뜻하며 반대 방향은 불가능



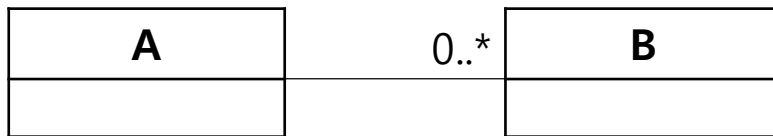
```
public class A{
    private B b;
}
```

```
public class B{
    private A a;
}
```

▶ 연관 관계

✓ 연관 관계의 다중성

관계를 맺을 수 있는 실제 상대 객체의 수를 다중성을 통하여 지정 가능
동일한 의미/역할의 복수 개의 객체와의 관계

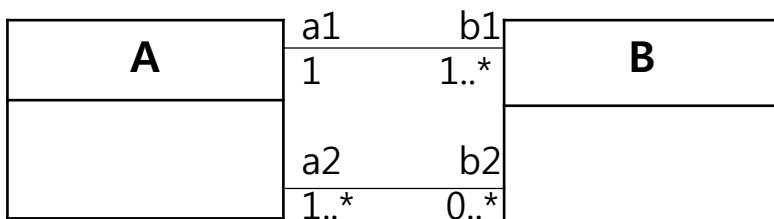


```
public class A{
    private Collection<B> b;
}
```

```
public class B{
    private A a;
}
```

✓ 다중 연관

동일한 클래스 간의 존재하는 복수 개의 연관 관계를 뜻 함
다른 의미/역할의 복수 개의 객체와의 관계



```
public class A{
    private Collection<B> b1;
    private Collection<B> b2;
}
```

```
public class B{
    private A a1;
    private Collection<A> a2;
}
```

▶ 합성(포함) 관계와 집합 관계

두 대상 간의 포함(소속) 표현으로 항상 Has-a 의미가 성립되어야 함

✓ 포함 관계

양 측 클래스의 관계를 1:1의 관계로 나타냄

→ 부분 객체가 오직 하나의 전체 객체에 포함 될 수 있음

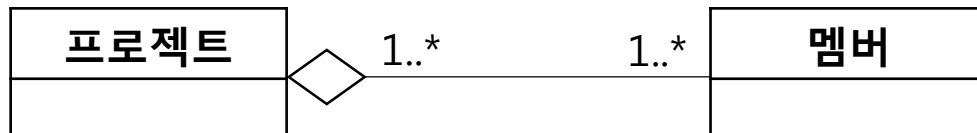


회사는 직원으로 구성된다
직원은 회사의 부분이다

✓ 집합 관계

연관 관계를 좀 더 세분화 하여 양 측 클래스의 관계를 1:N의 관계로 나타냄

→ 부분 객체가 다수의 전체 객체에 의해 공유 될 수 있음

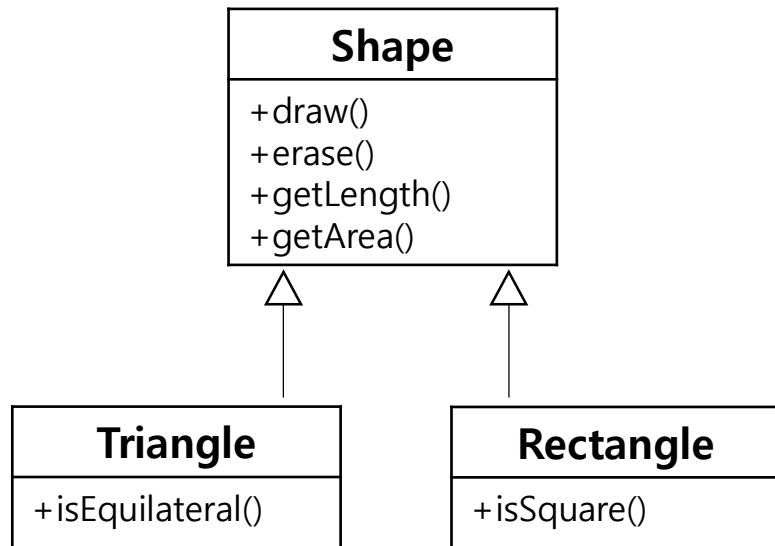


멤버는 프로젝트의 부분이다
프로젝트는 멤버로 구성된다

▶ 일반화 관계와 실체화(인터페이스 실현) 관계

✓ 일반화 관계

보다 일반적인 클래스와 보다 구체적인 클래스 간의 관계를 뜻하는 것으로
한 클래스(상위 클래스)가 다른 클래스(하위 클래스)보다 일반적인 개념/대상 임을 의미하는 관계



```
public class Shape {
    public void draw() {...}
    public void erase() {...}
    public int getLength() {...}
    public double getArea() {...}
}
```

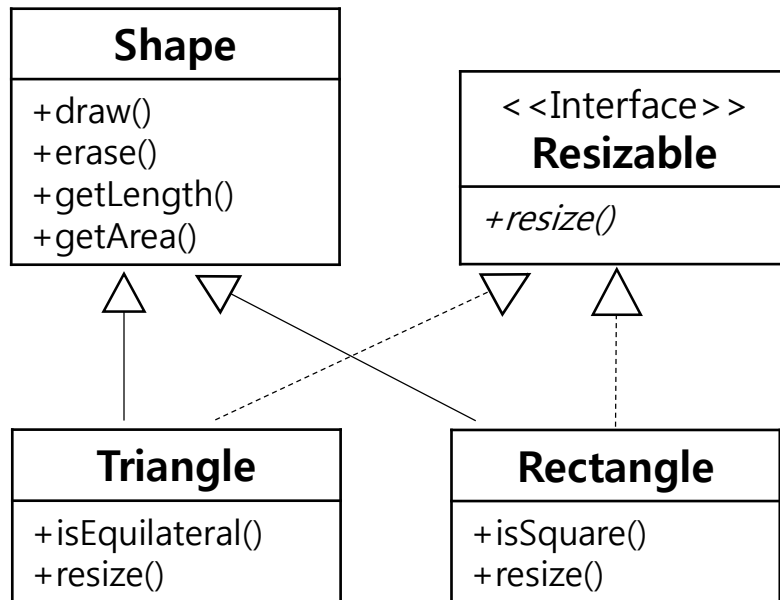
```
public class Triangle extends Shape {
    public boolean isEquilateral() {...}
}
```

```
public class Rectangle extends Shape {
    public boolean isSquare() {...}
}
```

▶ 일반화 관계와 실체화(인터페이스 실현) 관계

✓ 실체화(인터페이스 실현) 관계

인터페이스에 명시된 기능을 클래스에 의해서 구현한 관계 의미



```
public interface Resizable {
    void resize();
}
```

```
public class Triangle extends Shape
    implements Resizable {
    public boolean isEquilateral() {...}
    public void resize() {...}
}
```

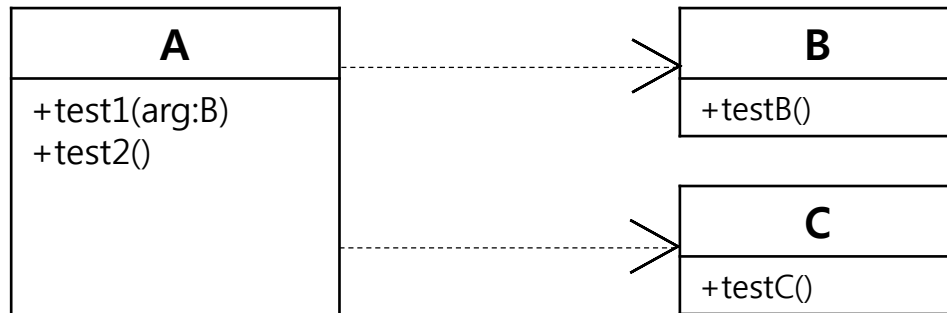
```
public class Rectangle extends Shape
    implements Resizable {
    public boolean isSquare() {...}
    public void resize() {...}
}
```

▶ 의존 관계와 인터페이스 의존 관계

✓ 의존 관계

두 클래스의 연산 간의 호출 관계를 표현한 것으로 제공자의 변경이 이용자에 영향을 미칠 수 있음을 의미(제공자의 변경이 이용자의 변경 유발)

이용자는 의존 관계를 통해서 제공자의 연산을 호출할 수 있음

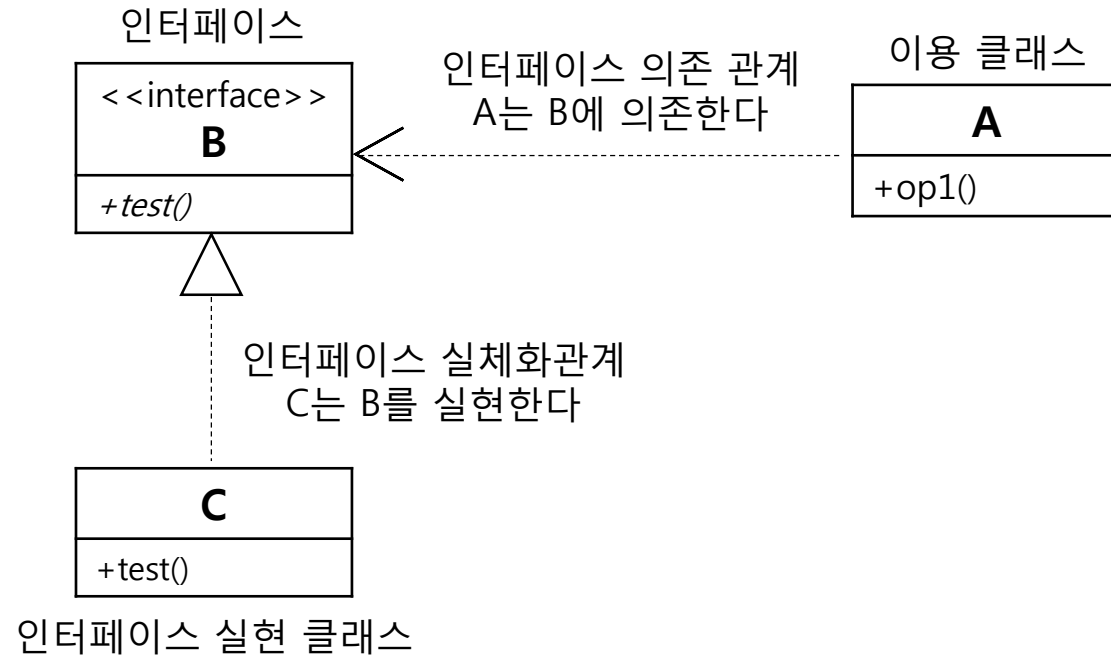


```
public class A{
    public void test1(B arg){
        arg.testB();
    }
    public void test2(){
        C c = new C ();
        c.testC();
    }
}
```

▶ 의존 관계와 인터페이스 의존 관계

✓ 인터페이스 의존 관계

인터페이스와 인터페이스 이용자 간의 이용관계를 표현할 때 사용 될 수 있음



```
public interface B{
    void test();
}

public class C implements B{
    public void test() {...}
}

public class A{
    public void op1(){
        B b = new C();
        b.test();
    }
}
```

▶ 연관 관계와 의존 관계

	연관 관계	의존 관계
역할	메시지 전달의 통로	
관계의 발생 형태	상대 클래스의 속성으로 대응	해당 연산의 인자 클래스 해당 연산 내부 객체의 클래스
관계의 지속 범위	해당 객체의 생명주기	해당 연산 내부
방향성	양방향 가능	단방향

Chap04.

시퀀스 다이어그램

▶ 시퀀스 다이어그램

✓ 시퀀스 다이어그램

동적(행위) 다이어그램으로 상호작용 다이어그램의 일부분

시스템 내부에서 동작하는 객체들 사이의 주고 받는 메시지를 시간 순서를 강조하여 표현

✓ 생명선과 메시지

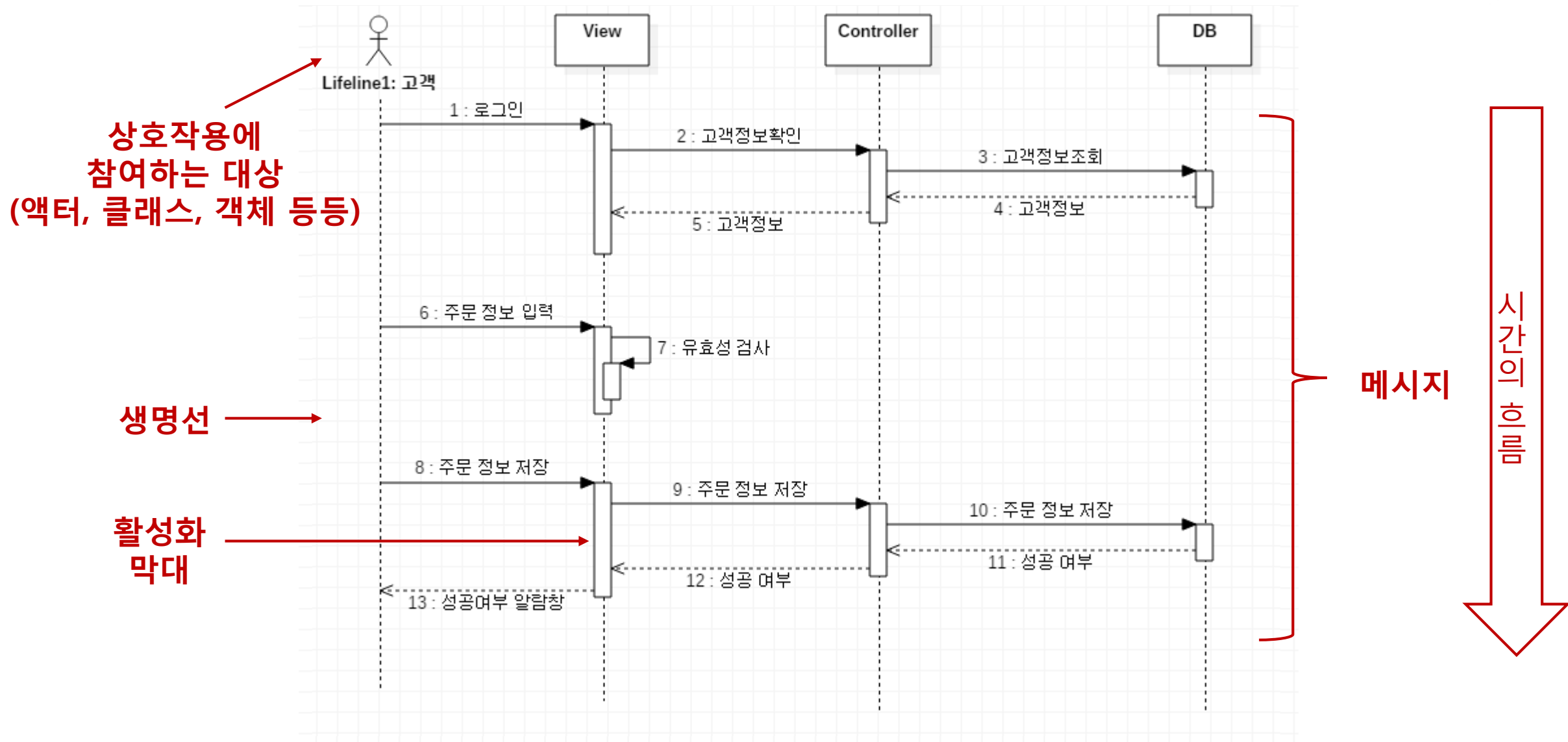
생명선 : 액터, 클래스 객체, 컴포넌트의 인스턴스 등, 상호작용에 참여하는 구체적인 대상 표현

생명선 끝에 X자로 끊겨 있으면 소멸

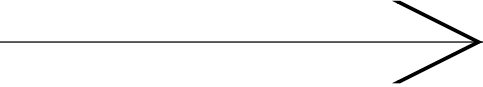

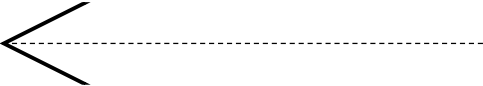
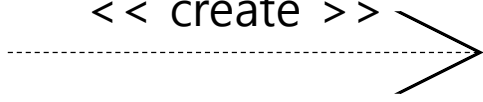


메시지 : 생명선 간에 전달되어 상태의 행위에 대한 호출

* 생명선의 소멸은 자바에서의 가비지 컬렉터에 넘기는 행동이 됨

시퀀스 다이어그램



▶ 시퀀스 다이어그램 메시지 종류

메시지 유형	표현법	설명
비동기적 메시지		송신자를 대기 시키지 않음
동기적 메시지		송신자를 대기시킴
대답 메시지		동기적 메시지의 수행 결과
생성 메시지		생명선 생성
발견된 메시지		모르는 송신자로부터의 메시지
유실된 메시지		모르는 수신자로부터의 메시지

시퀀스 다이어그램 제어흐름

Interaction Operator
(상호작용 연산자)

alt

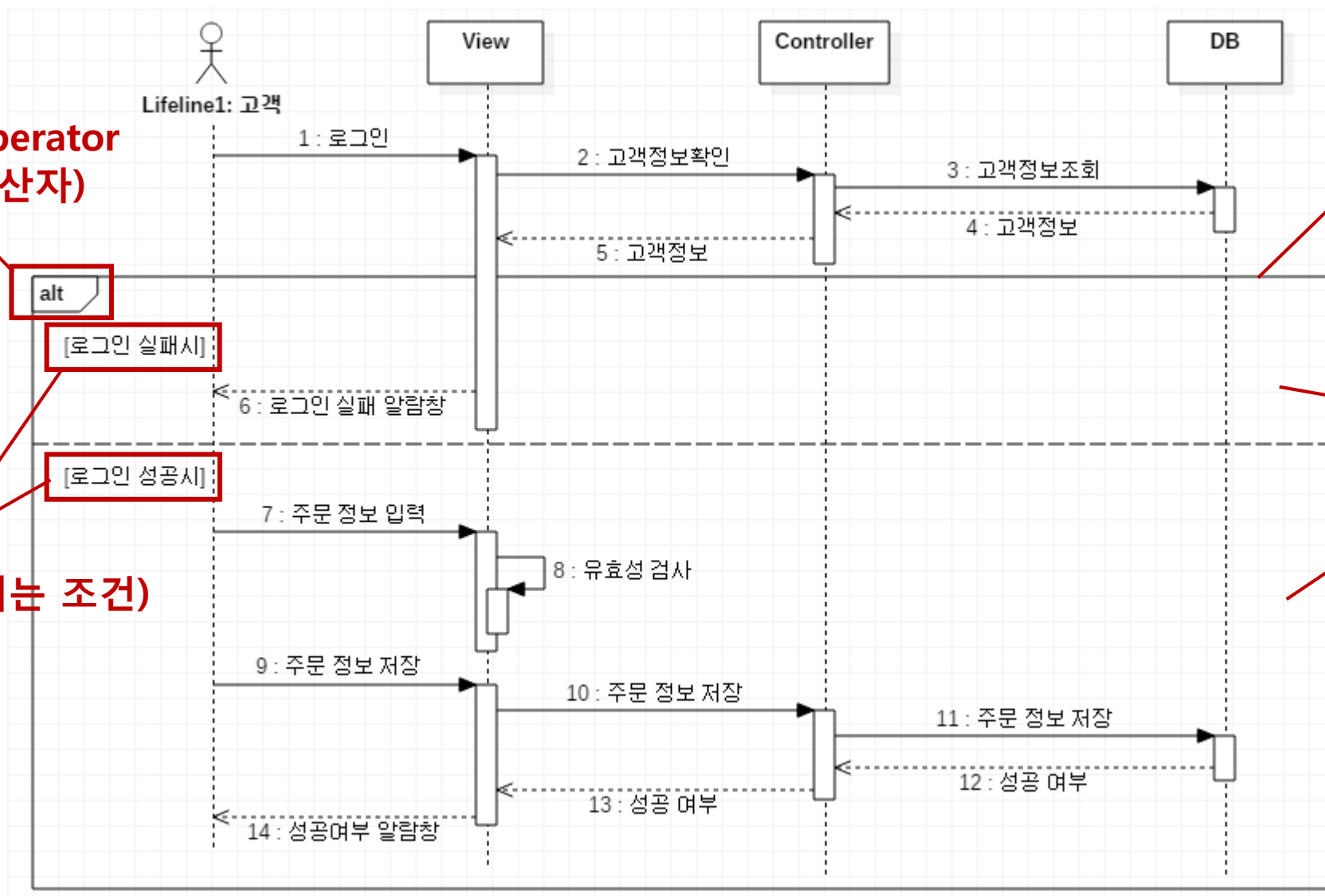
[로그인 실패시]

[로그인 성공시]

Combined Fragment
(결합 조각)

Fragment
(조각)

Guard
(메시지가 수행되는 조건)



▶ 시퀀스 다이어그램 상호작용 연산자 종류

연산자 종류		설명
alt	대체	메시지의 대체 시퀀스를 포함하는 연산자이며 어떤 상황에서도 하나의 시퀀스만 발생함 → if - else if 문에 해당하는 논리를 나타냄. 단, 모든 가드가 false이고 else가드가 존재하지 않으면 어떠한 조각도 실행되지 않음
opt	옵션	선택적 요소로 발생하거나 발생하지 않을 수 있는 시퀀스를 포함 → if 문에 해당하는 논리를 나타냄. 대안을 하나만 제공해야 하고 조건이 false일 경우 조각이 실행 되지 않음
loop	반복	루프 상호작용 연산자는 반복적으로 실행되는 부분을 나타냄 가드 안에 단편이 실행되는 횟수를 지정할 수 있음
break	중단	보통 loop 연산자와 함께 쓰이며 중단 상호작용 연산자는 기타 프로그래밍 언어의 중단 매커니즘과 유사함 조건이 true 일 경우 현재 실행을 포기하고 빠져나감

* 이 외에 critical, ignore, strict, seq 등이 존재한다.