

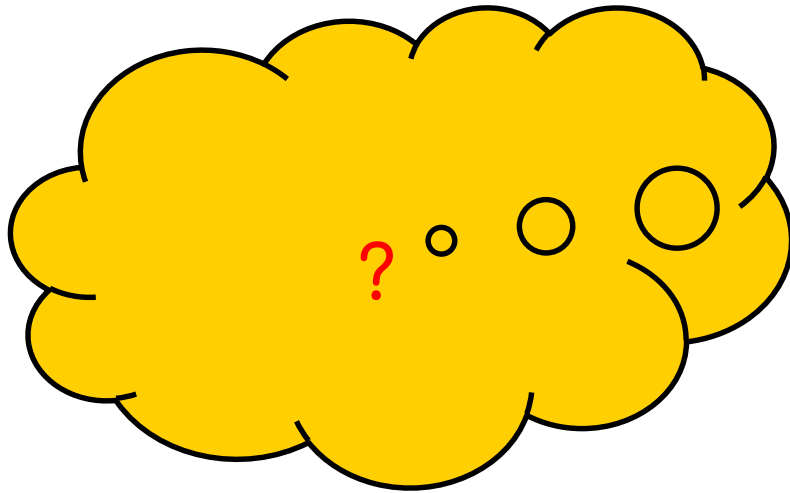


# Ch.3 概念모델링

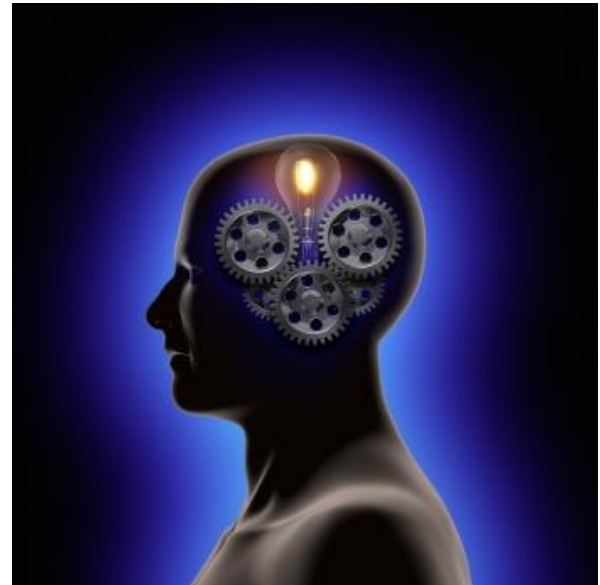
---

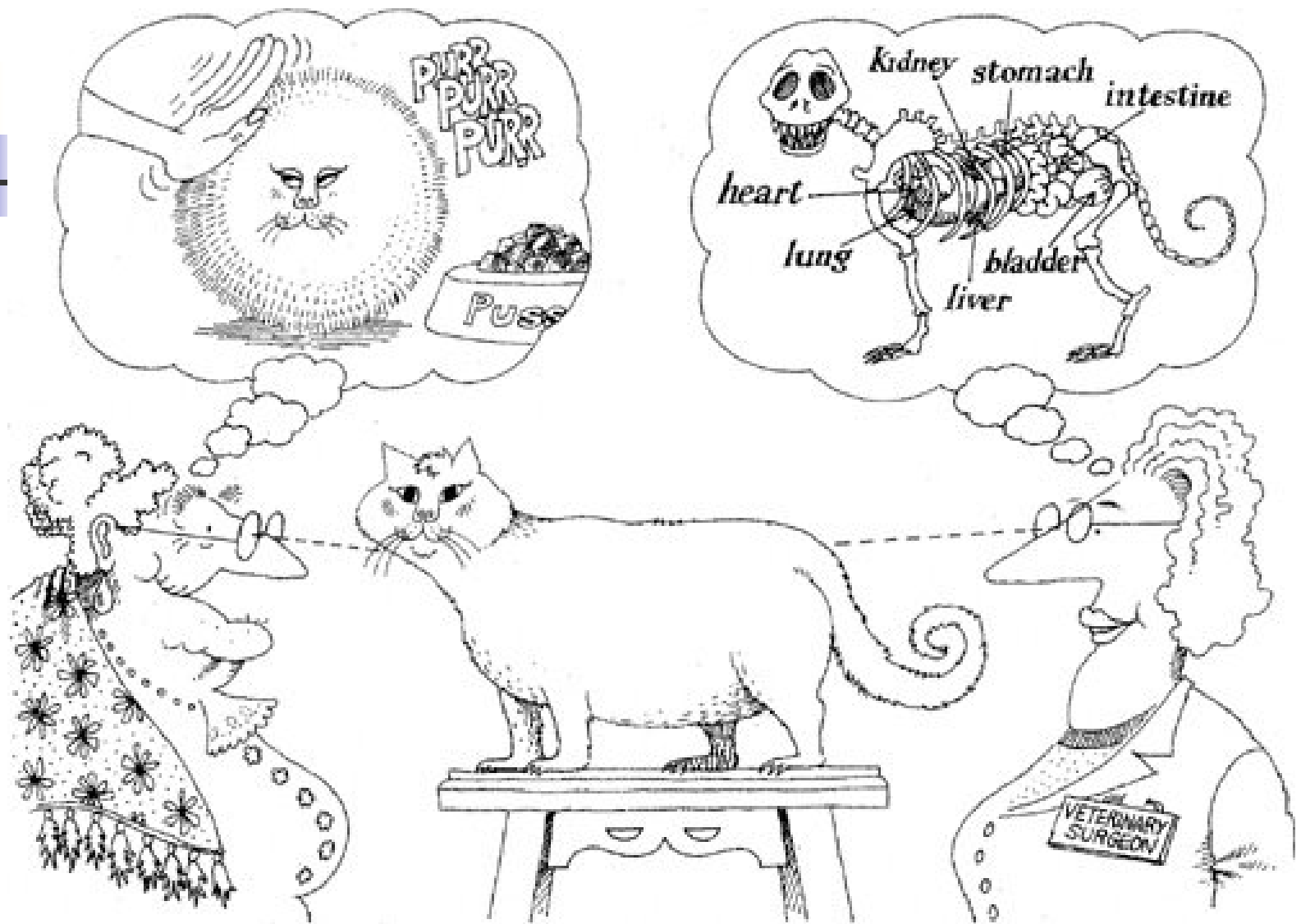
# 개념 (概念, Concept)

- 어떤 사물 현상에 대한 일반적인 지식
- A concept is a cognitive unit of
  - meaning, an abstract idea, knowledge, ...



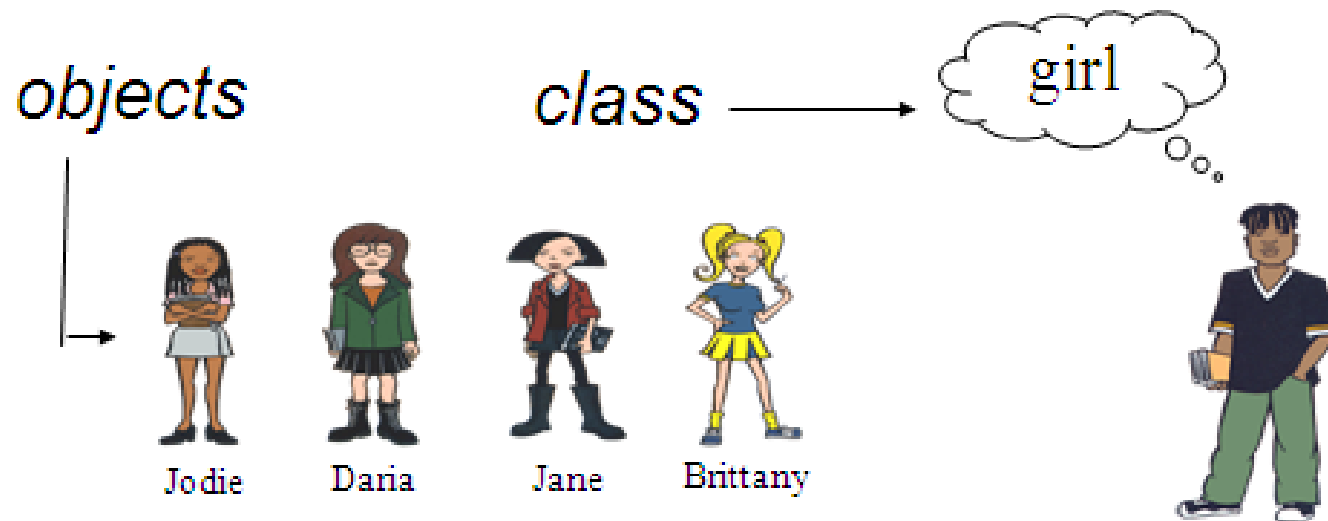
Domain





# Objects and Classes

- Classes reflect concepts, objects reflect instances that embody those concepts.



- A **class** captures the common properties of the objects instantiated from it.
- A class characterizes the common behaviors of all the objects that are its instances.



## 3.1 개념객체 (Concept Objects)

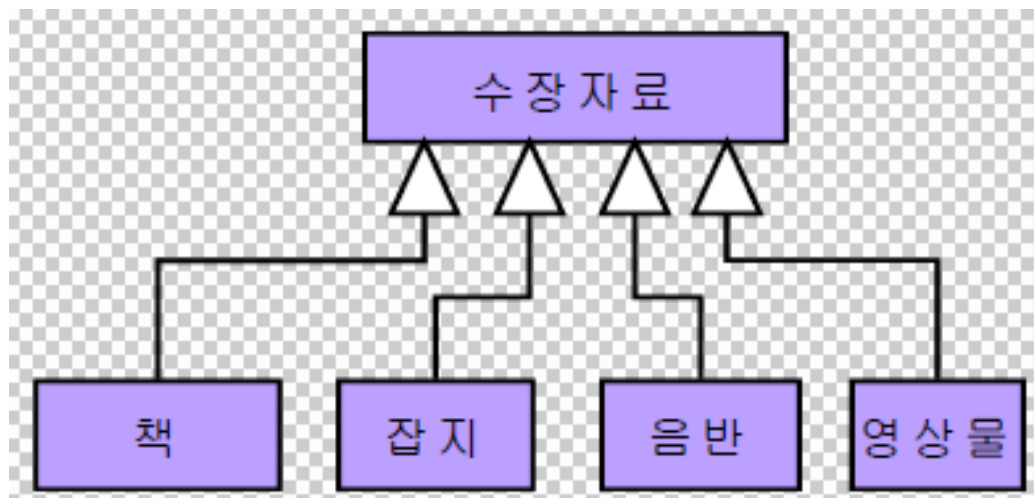
---

- 개발시스템에서 **key**가 되는 정보, 기본적인 개념
  - “도서관관리시스템”의 개념객체 “후보”
    - Object flow
    - Use Case의 목적어
    - UC Scenario의 메시지 매개변수
    - UC의 사전/사후조건의 목적어
  - Object
    - 내부데이터와 동작을 갖고, 외부로부터의 메시지를 수신하여 동작하는 기본계산단위
- 개념객체 ➔ 시스템의 중심이 됨.
- “Class”

개념객체

## 3.2 “도서관관리시스템”의 개념객체

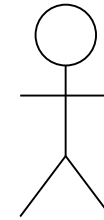
- “수장자료”
  - 도서관에 수장되어 있는 자료
  - 책, 잡지, 음악 및 영상물, ...등 다종다양
- “수장자료정보”보다는 “수장자료”가 적합
  - 실세계의 객체를 모델링



## 3.2 “도서관관리시스템”의 개념객체

### ■ “이용자”

- 수장자료를 대출하는 사람(개념적)



이용자

### ■ 혼동하지 말것!

- Actor(“이용자”) ➔ 진짜 사람
- 개념객체(“이용자”) ➔ 시스템 내부에 존재하는 가상의 사람

이용자



# 대출 & 예약

---

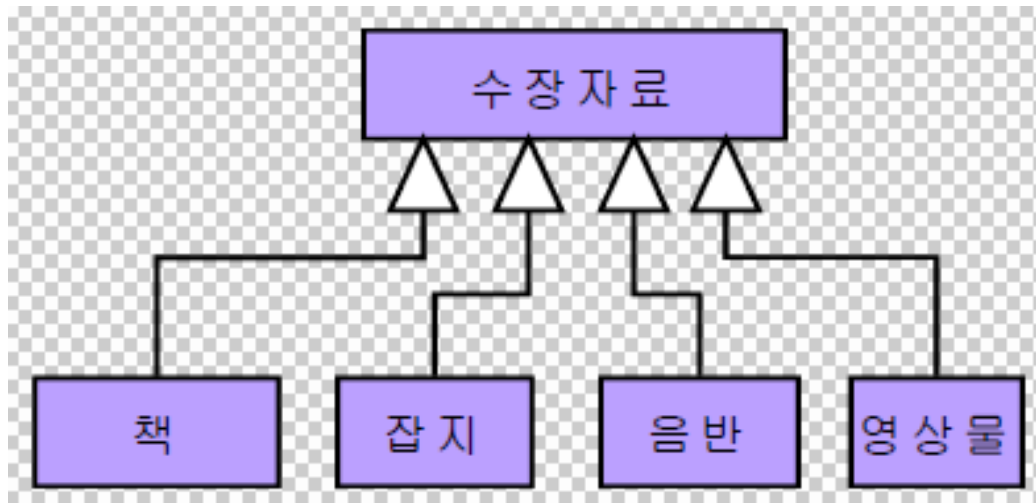
- 도서관에 책이 있고, 이용자가 존재하는 상태에서, “대출한다”를 살펴보면...
  - 행위?
  - 따라서, 객체로 생각하기에는 부적절?
- “대출하는 일” = “대출”
  - 정보(대출기록,...)가 포함되어 시스템에 있어서 의미있음.
  - 따라서, (개념적인) 객체로 인정가능!
- 같은 의미로 생각해 보면, “예약”도 역시 객체!

대출

예약



# 이용자가 대출 또는 예약하기 위한 재료가 모두 준비 완료!



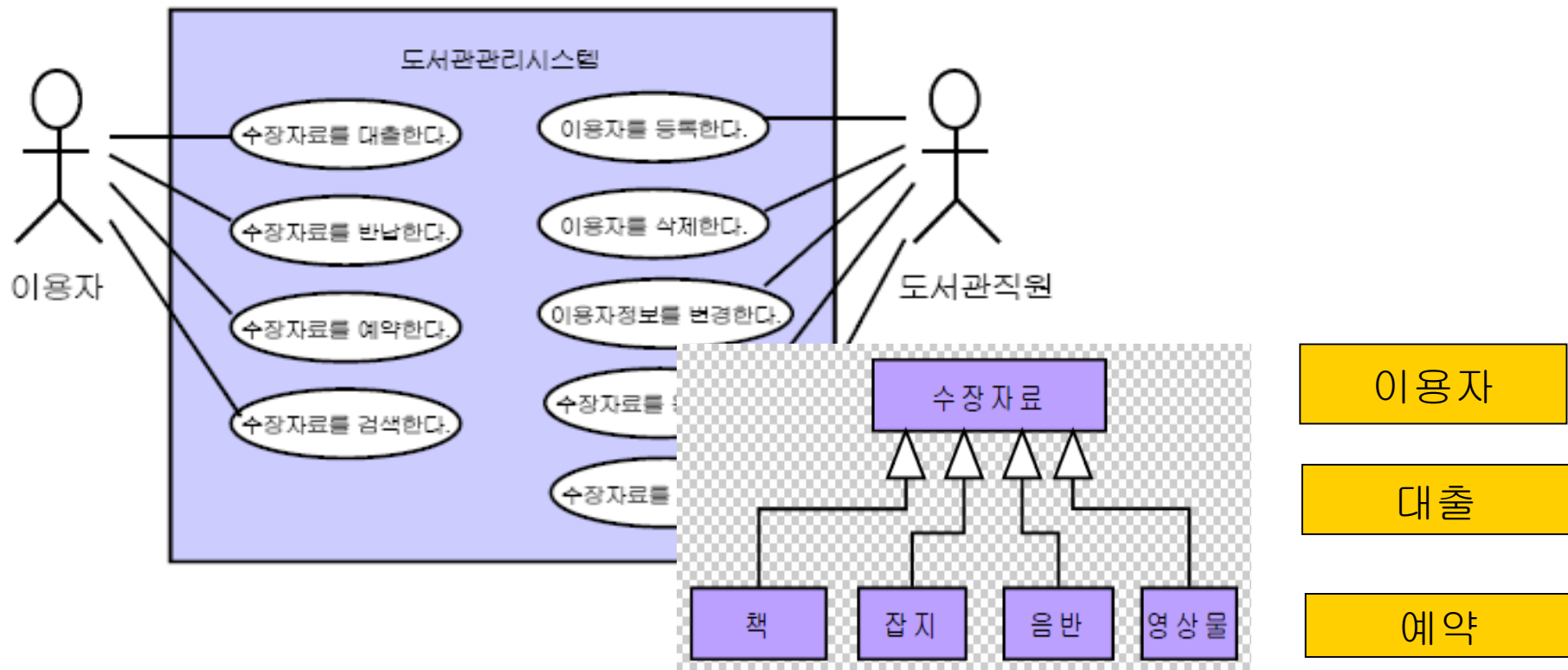
이용자

대출

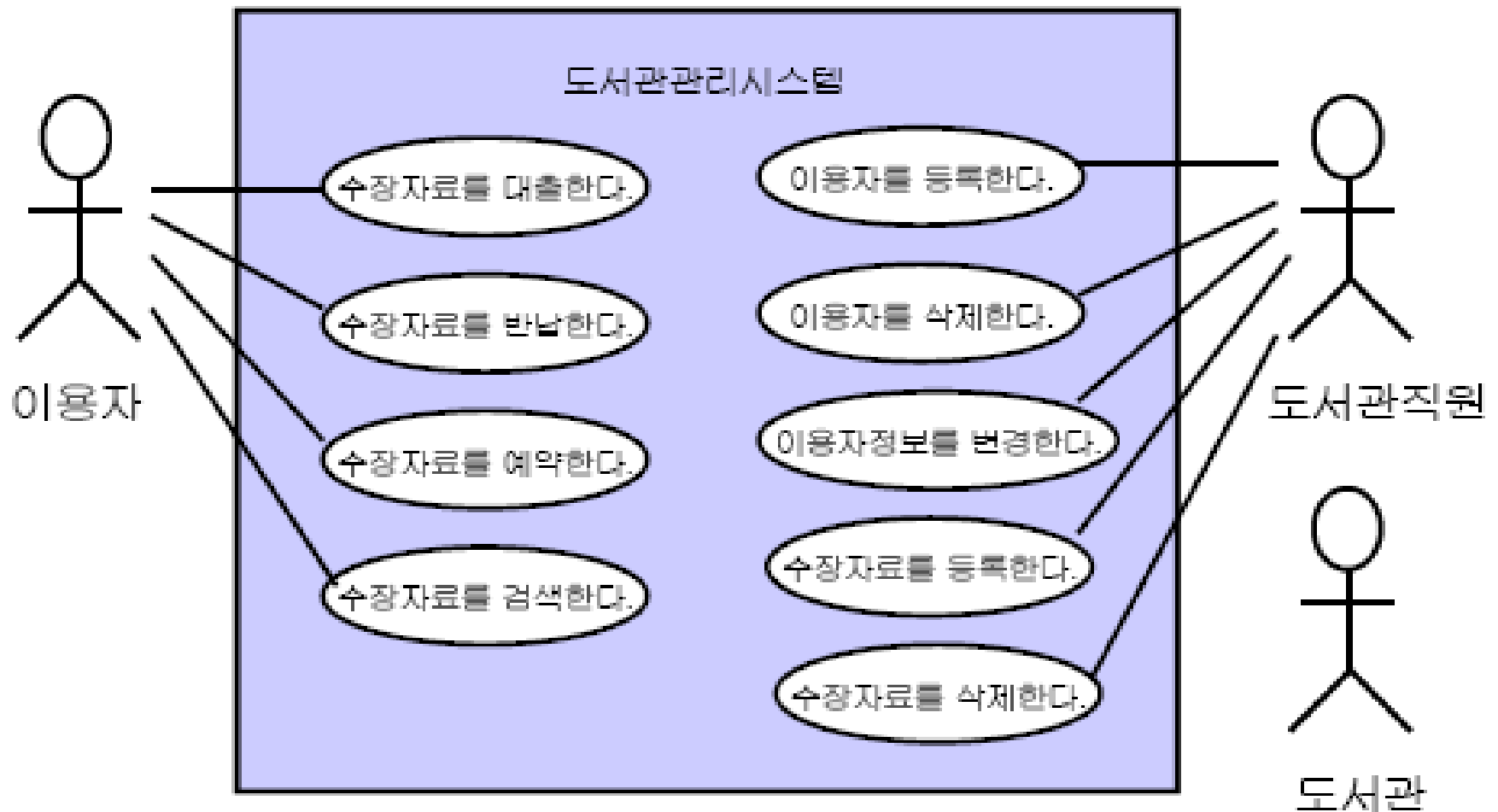
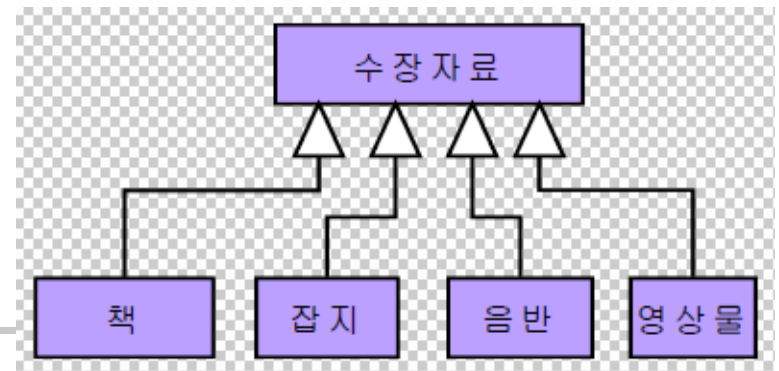
예약

## 3.3 개념객체와 Use Case

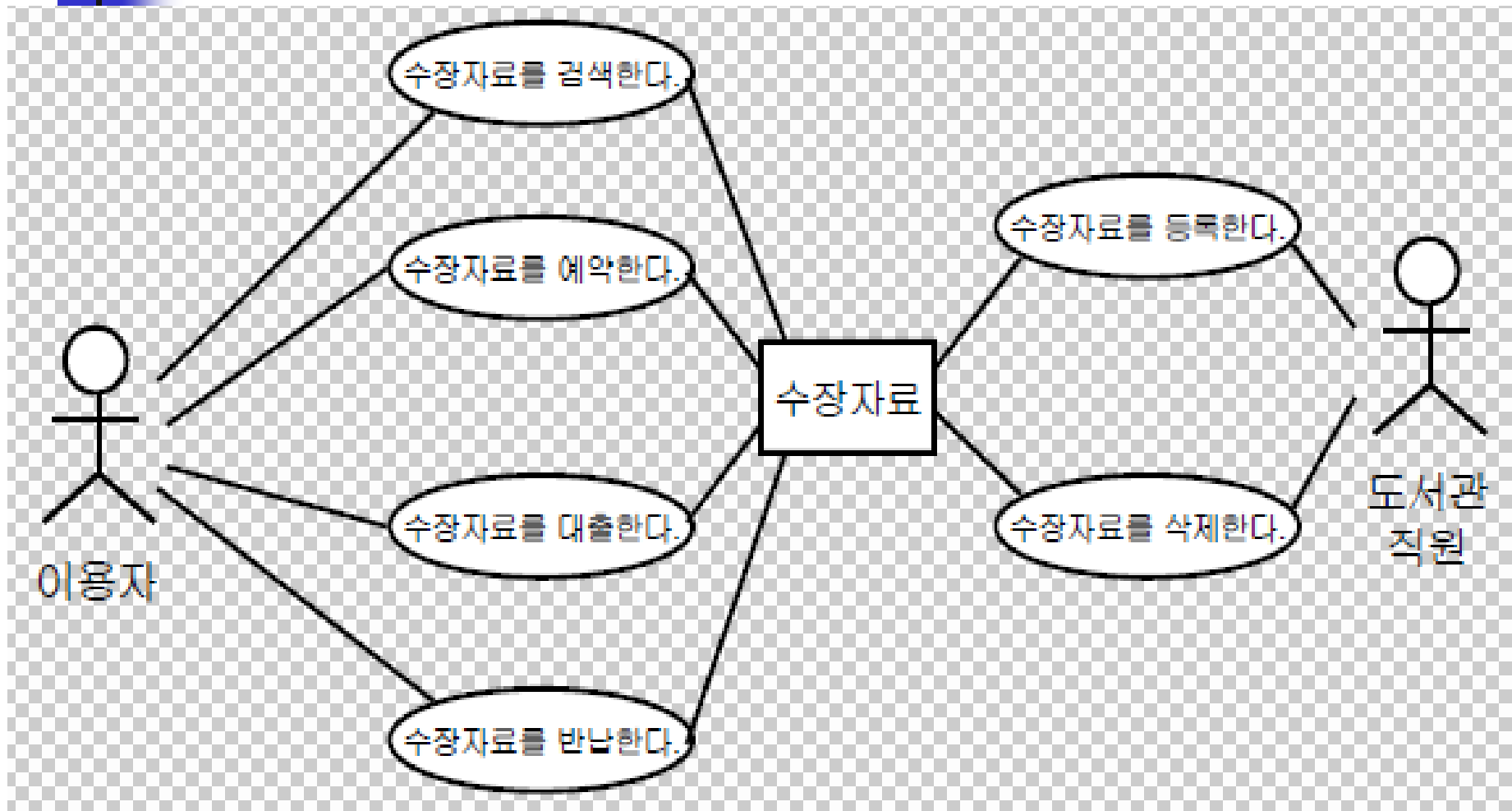
- 예전 Use Case Diagram을 참조하여 Use Case Diagram에 개념객체들이 어떻게 관련되는지 살펴보자!



# “수장자료”

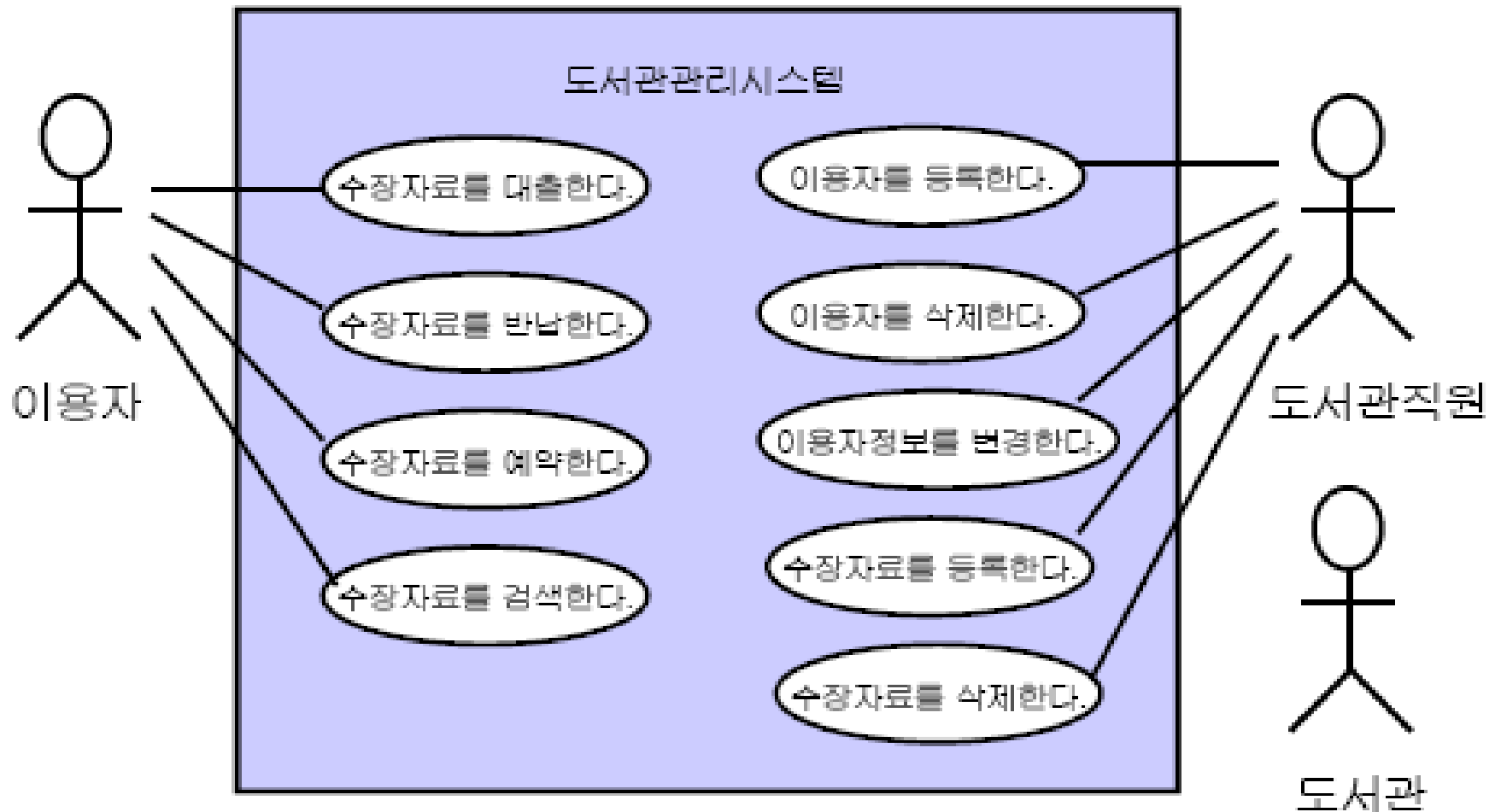


# “수장자료”개념객체를 UCD에 기입, 관련관계 표기

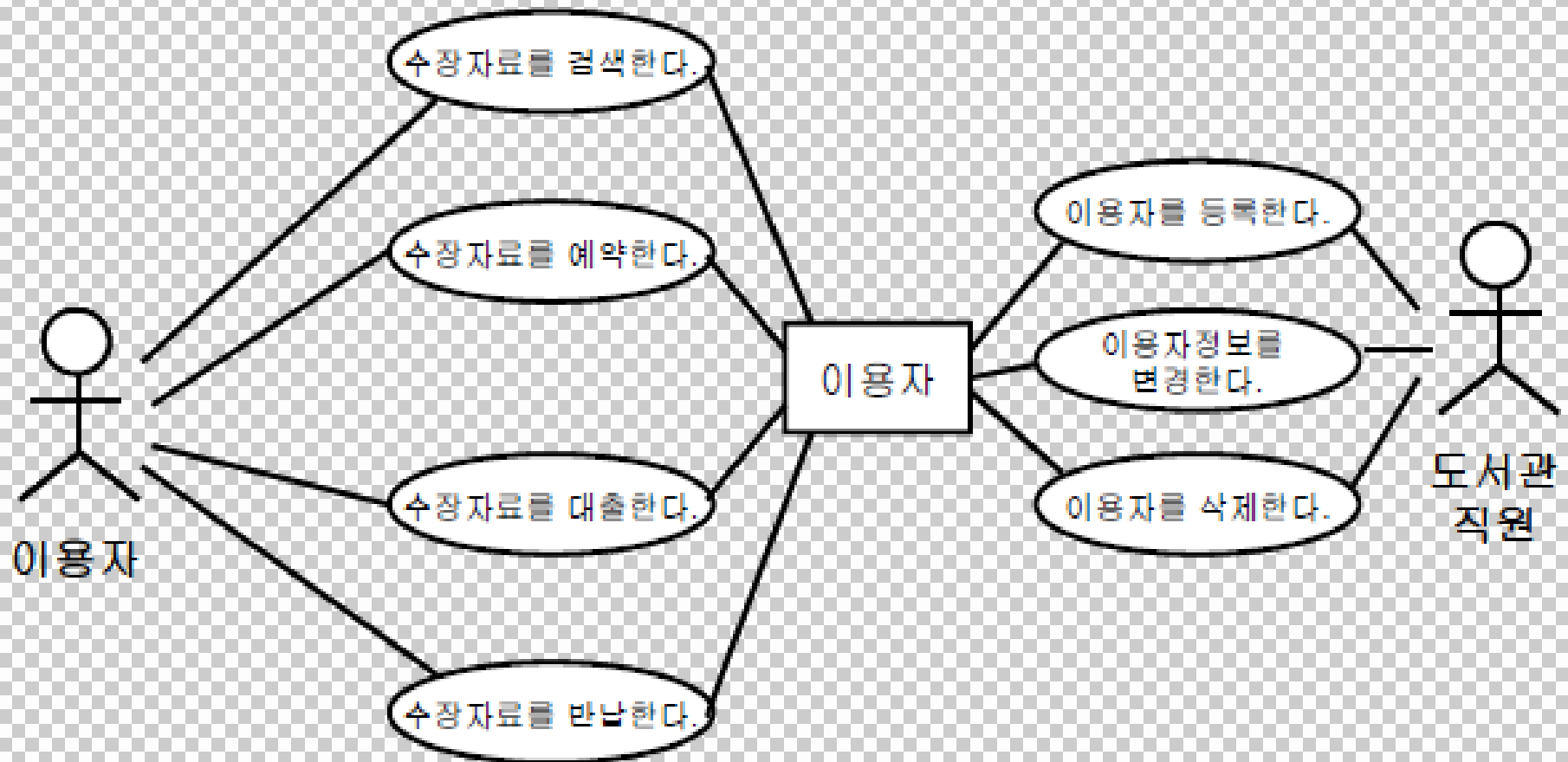


# “이용자”

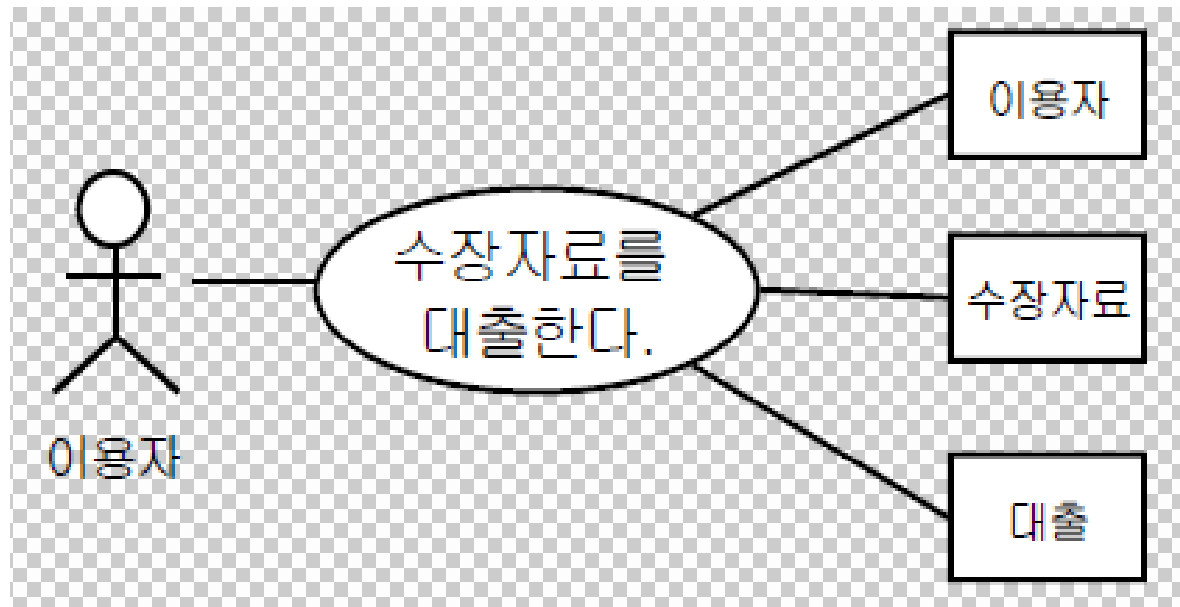
이용자



# “이용자”객체의 추가



# “수장자료를 대출한다” UC를 중심으로 표현



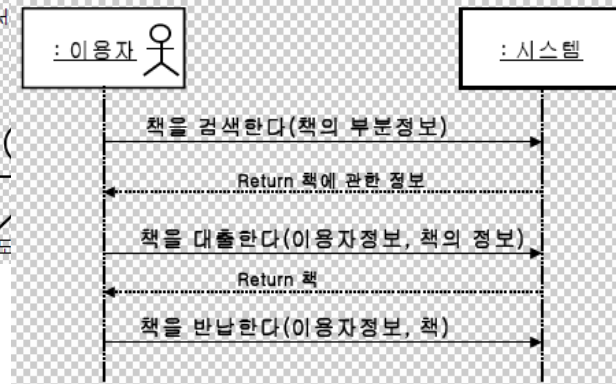
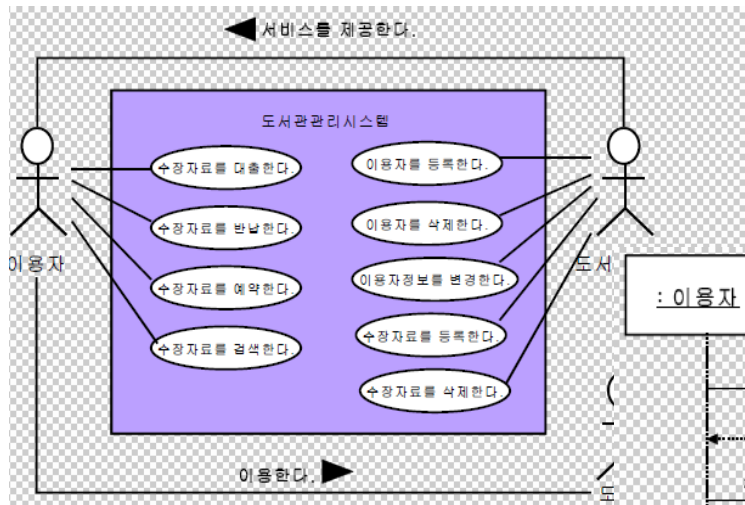
# 도서관관리시스템 개발

도서관관리시스템

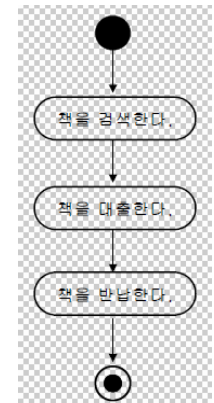
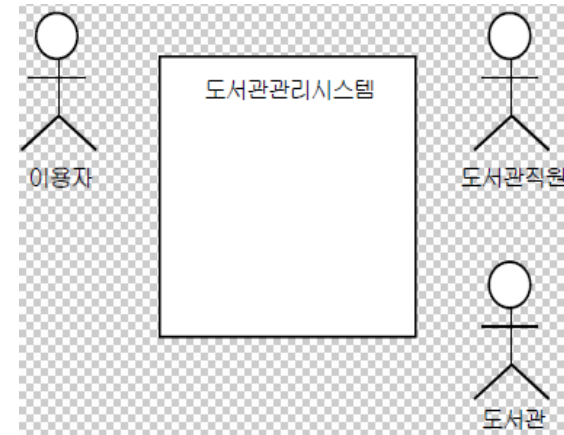
시스템 외부요소 파악 → Actor, Workflow

시스템과 외부경계 파악 → UC, UCS

이제는, 시스템의 내부 상세화



(c)SHwang, 2022







## 3.4 생명주기 (LifeCycle)

---

- 대부분의 객체들은 일생이 부여되어 있음.
- 일생동안 다양한 “상태” 변화
- **StateChart Diagram**으로 표현
- 즉, **SCD** = 객체의 일생 (**LifeCycle**) 표현물
- 기타 **Dgm**들과의 차이점
  - **SD** : 객체들 사이의 수수작용을 표현
  - **AD** : “행위”의 변화를 표현



# SCD vs. AD

---

- SCD : “상태”의 변화
  - “XX초등학교시절” ➔ “YY중학교시절” ➔ ...
- AD : “행위”의 변화
  - “XX초등학교입학” ➔ “YY중학교입학” ➔ ...

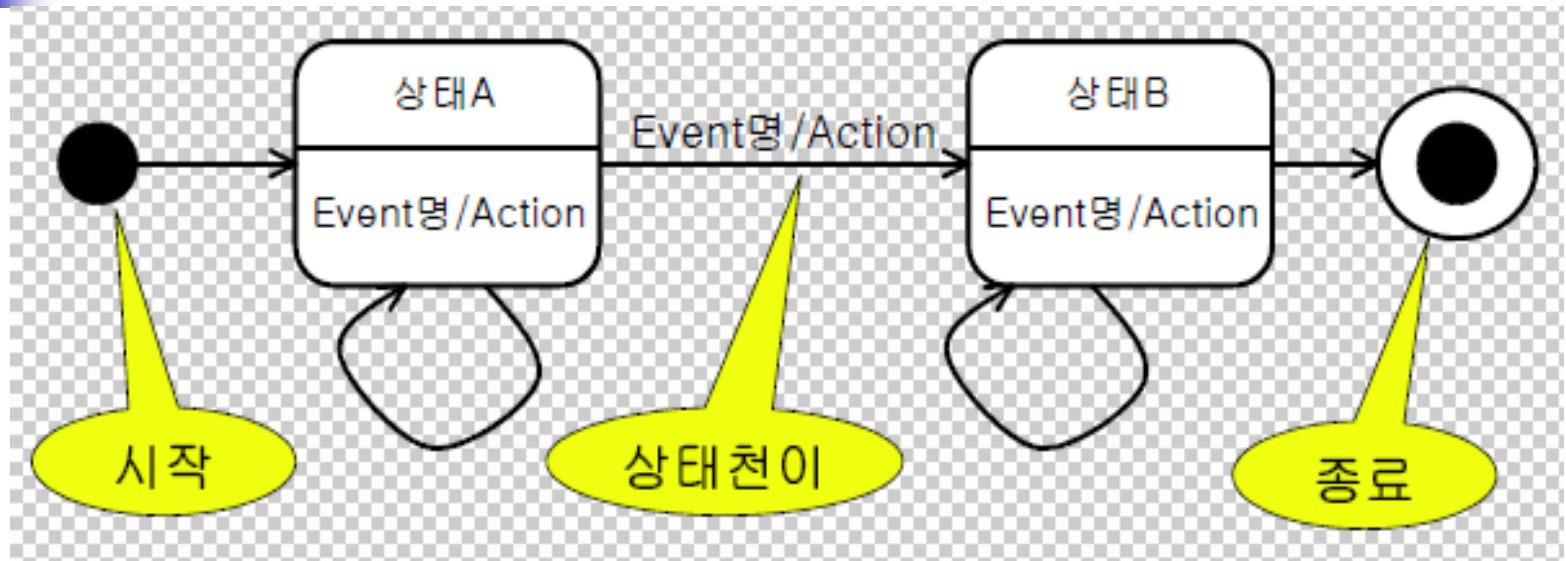


# SCD 표현

---

- “계좌”객체의 일생?
  - 상태가 너무 변화무쌍(잔고=1원, 2원,...) → 상태파악곤란
  - 해결책!
    - “계좌가 비어있는 상태,
    - 잔고가 있는 상태,
    - 대출상태,
    - ...

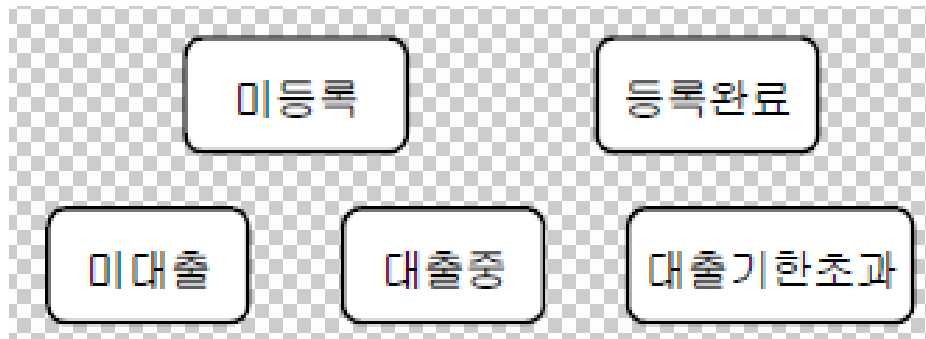
## 3.5 StateChart Diagram 遷移



- 한 개의 상태에서 여러 개의 상태들로의 천이가 가능
- 한 개의 상태에 여러 개의 상태를 내포 가능

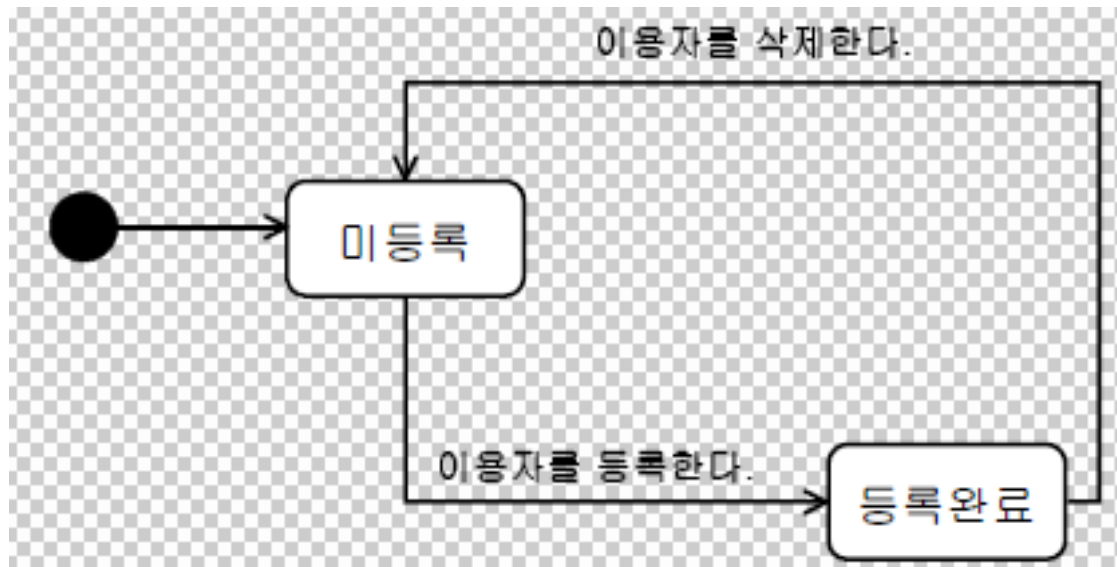
## 3.6 각 개념객체들의 생명주기

- “이용자”의 상태
  - 이용등록하지 않은 상태
  - 이용등록한 상태
  - 아무것도 대출하지 않은 상태
  - 책 등을 대출한 상태
  - 기한을 넘기고도 반납하지 않은 상태

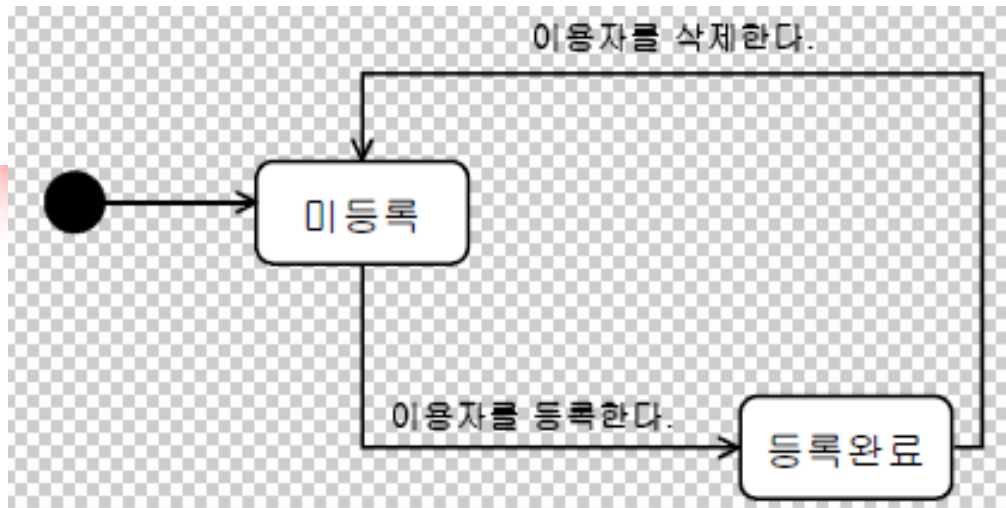


# “이용자”의 SCD

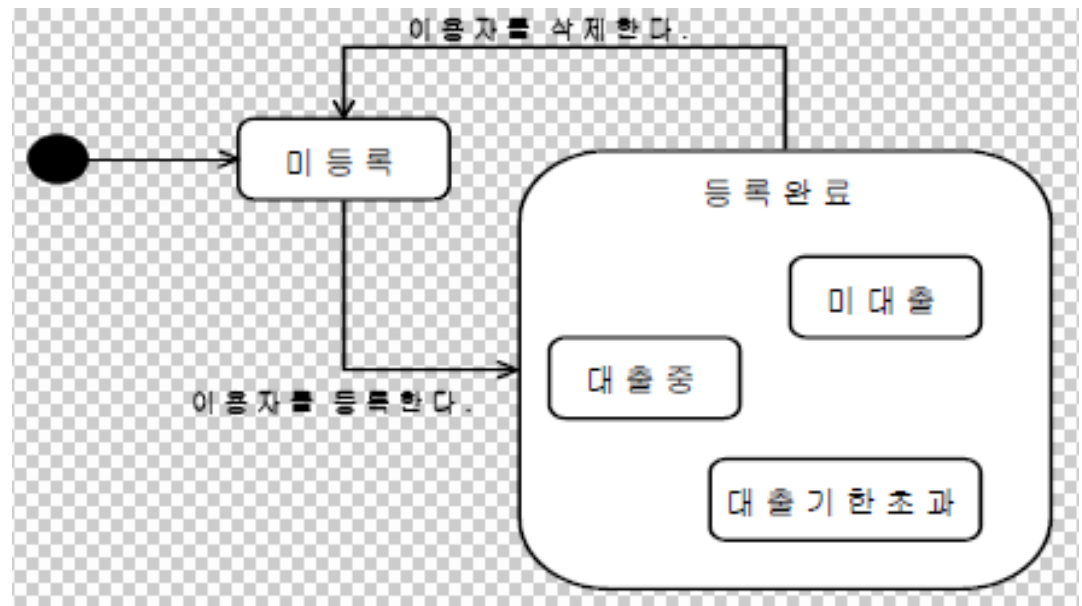
- “이용자”가 생성되면, 등록하지 않은 상태(미등록상태)
- “등록한다”를 수행하면, “등록완료”상태로 천이
- “삭제한다”를 수행하면, “미등록”상태로 천이

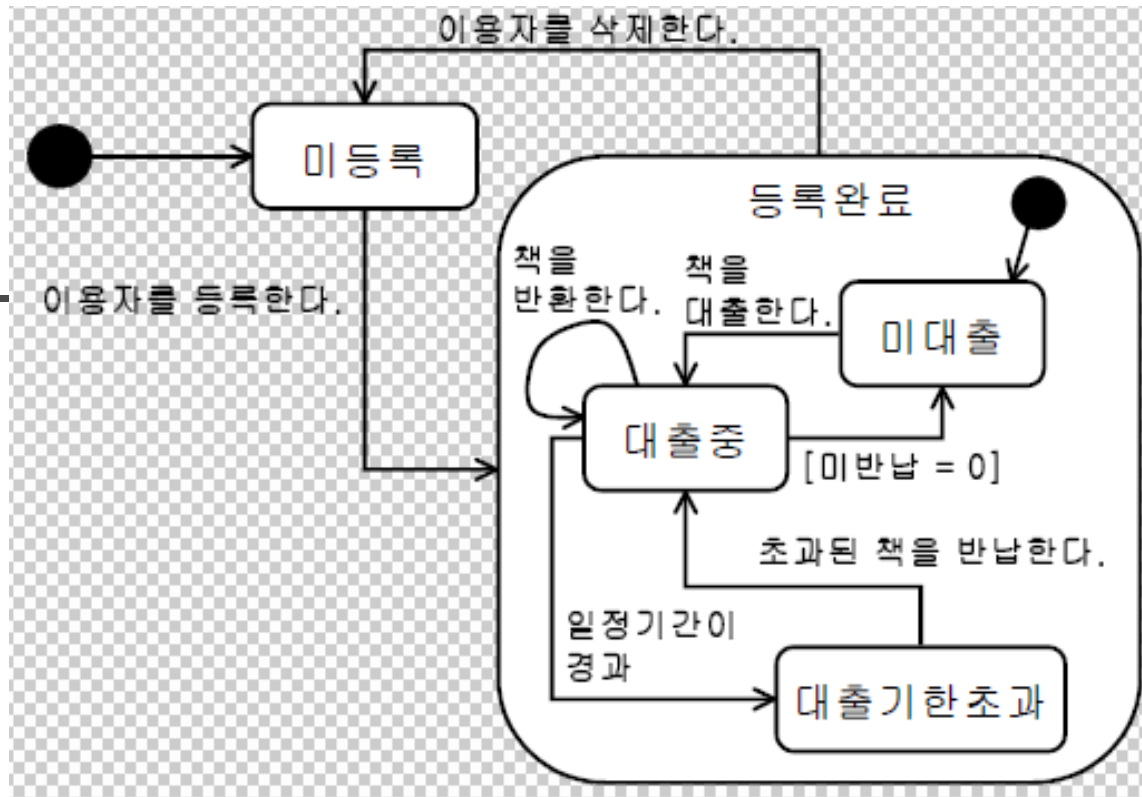


나머지 3개 상태는 “등록완료”상태일때에만 유효 → “등록완료”상태속에 내포



“미대출”, “대출중”, “대출기  
한초과”상태는  
“등록완료”상태에 내포됨!



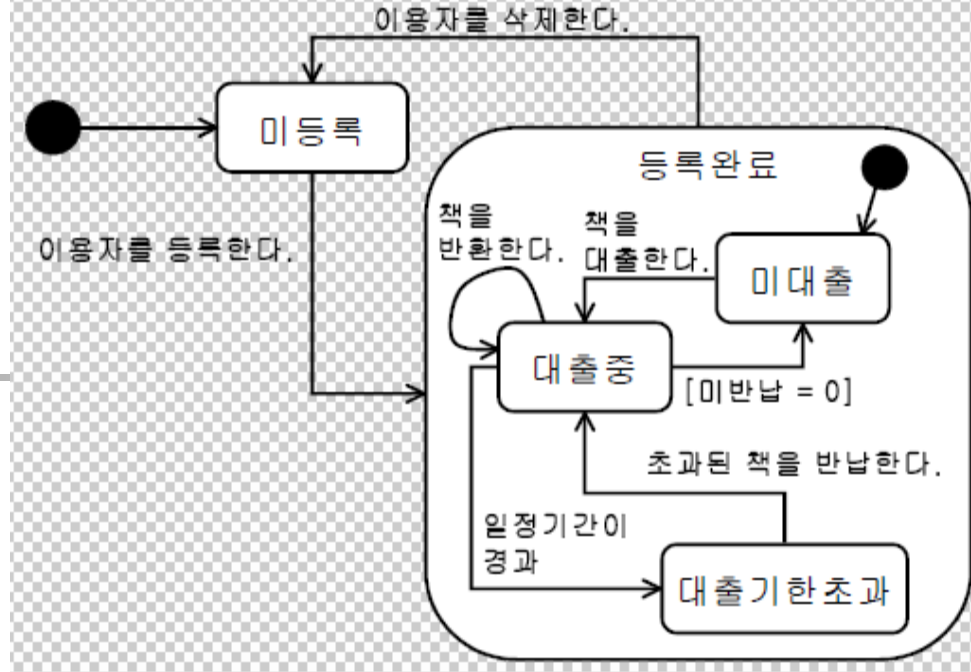


## ■ 중요검토사항

- Event가 UC와 대응?
- 각 상태는 UC의 사전/사후조건과 대응?
- 상태천이를 추적하면 올바른 LifeCycle재현가능?



# Event vs. Use Case



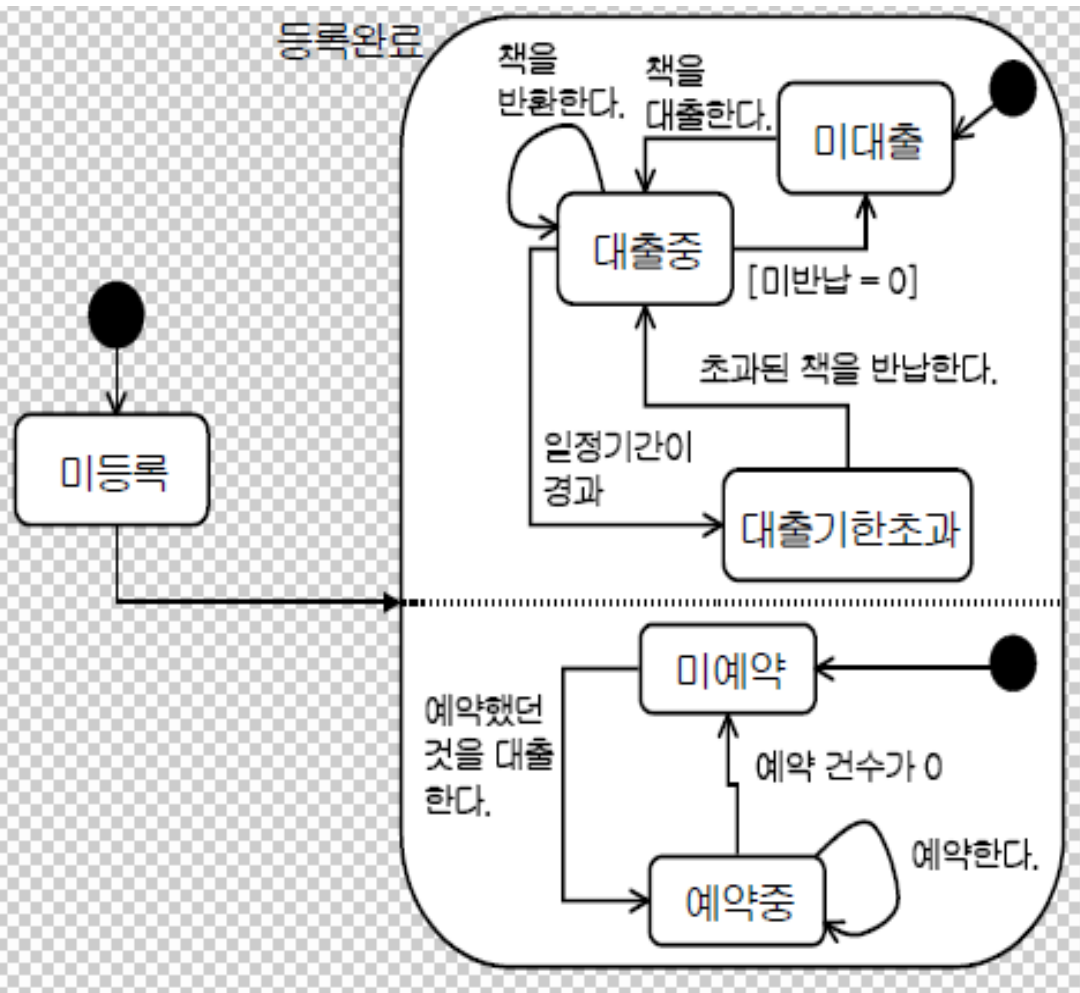
## ■ “이용자”와 관련있는 UC

- 수장자료를 대출한다.
- 수장자료를 반납한다.
- 이용자를 등록한다.
- 이용자를 삭제한다.
- 수장자료를 예약한다.
- 이용자정보를 변경한다.

예약중

미예약

# 병행상태



# SCD의 상태가 UC의 사전/사후 조건과 대응?

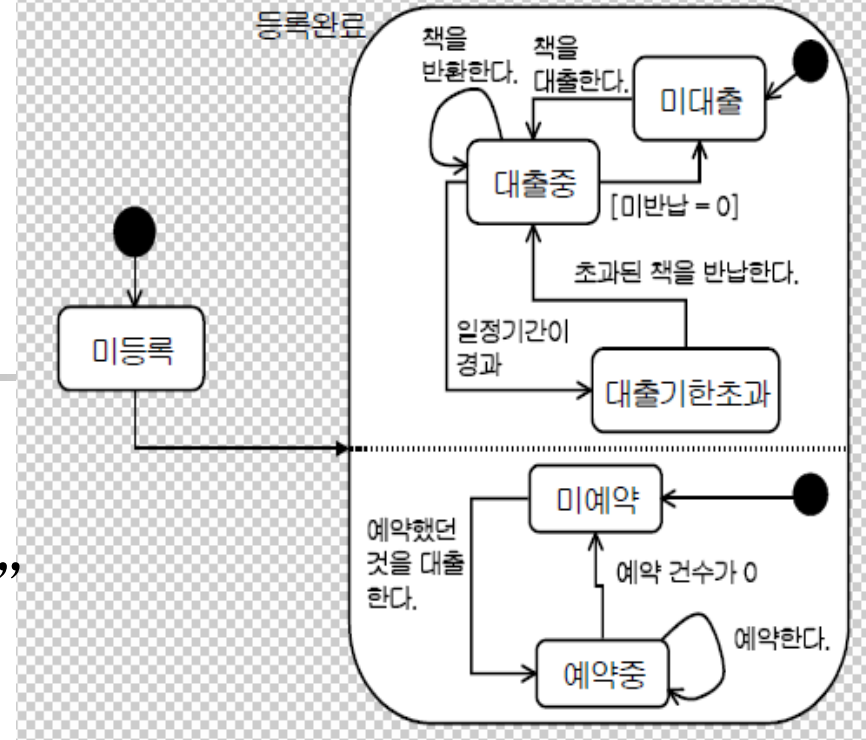
## ■ UC: “수장자료를 대출한다”

### ■ 사전조건

1. 대출하려는 사람이 이용자로서 등록되어 있을 것.
2. 대출하려는 사람에게 반납기한이 초과된 미반납자료가 없을 것.
3. 대출가능한 수량 이내이어야 함.
4. 수장자료가 대출가능해야 함.

### ■ 사후조건

1. 대출자의 대출완료점수가 증가한다.
2. 수장자료는 대출자에게 대출된 상태가 된다.





# SCD의 상태천이를 추적 → 올바른 LifeCycle인가?

---

- 복잡/대규모 시스템인 경우, 수작업으로는 거의 불가능
- UML Tool사용해야
  - SCD를 실제로 동작시키거나, 일시정지 → 상태 확인



# Why SCD?

---

- 객체의 생명주기(상태) 관찰/파악/모델링 ➔ 객체의 성질 명확히 파악에 도움
  - “상태천이표”를 작성하는 경우도 있음.
- SCD자체를, 즉시 실행가능한 SW로 변환가능

# 상태遷移表

- 행
  - 객체가 수신할 가능성이 있는 이벤트
- 열
  - 객체가 갖게되는 상태

이벤트	상태	미대출중	대출중	기한초과
대출한다.		-> 대출중	제한시간이내 -> 대출중	X
반환한다.		-	나머지 == 0 -> 미대출중, Other -> 대출중	나머지 == 0 -> 미대출중, 초과 나머지 > 0 -> 기한초과, Other -> 대출중

조건X → 상태Y



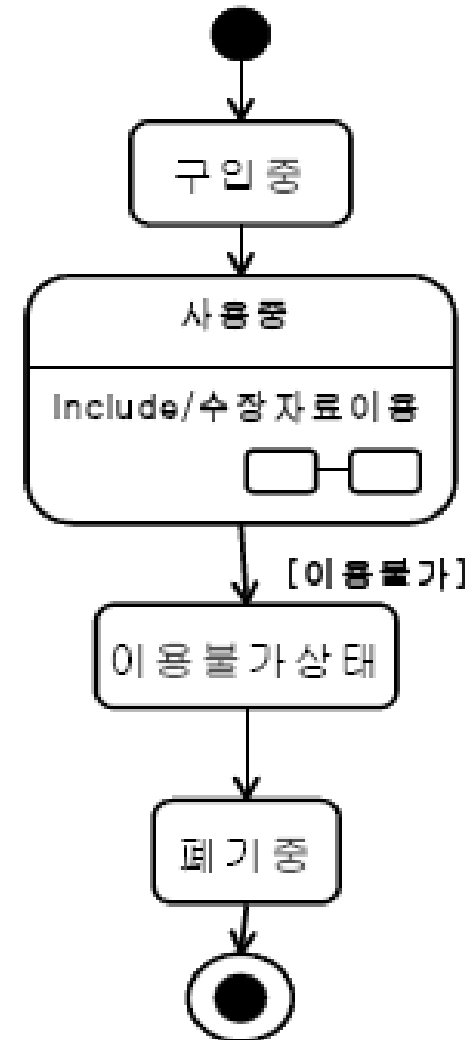
# “대출” & “예약”

---

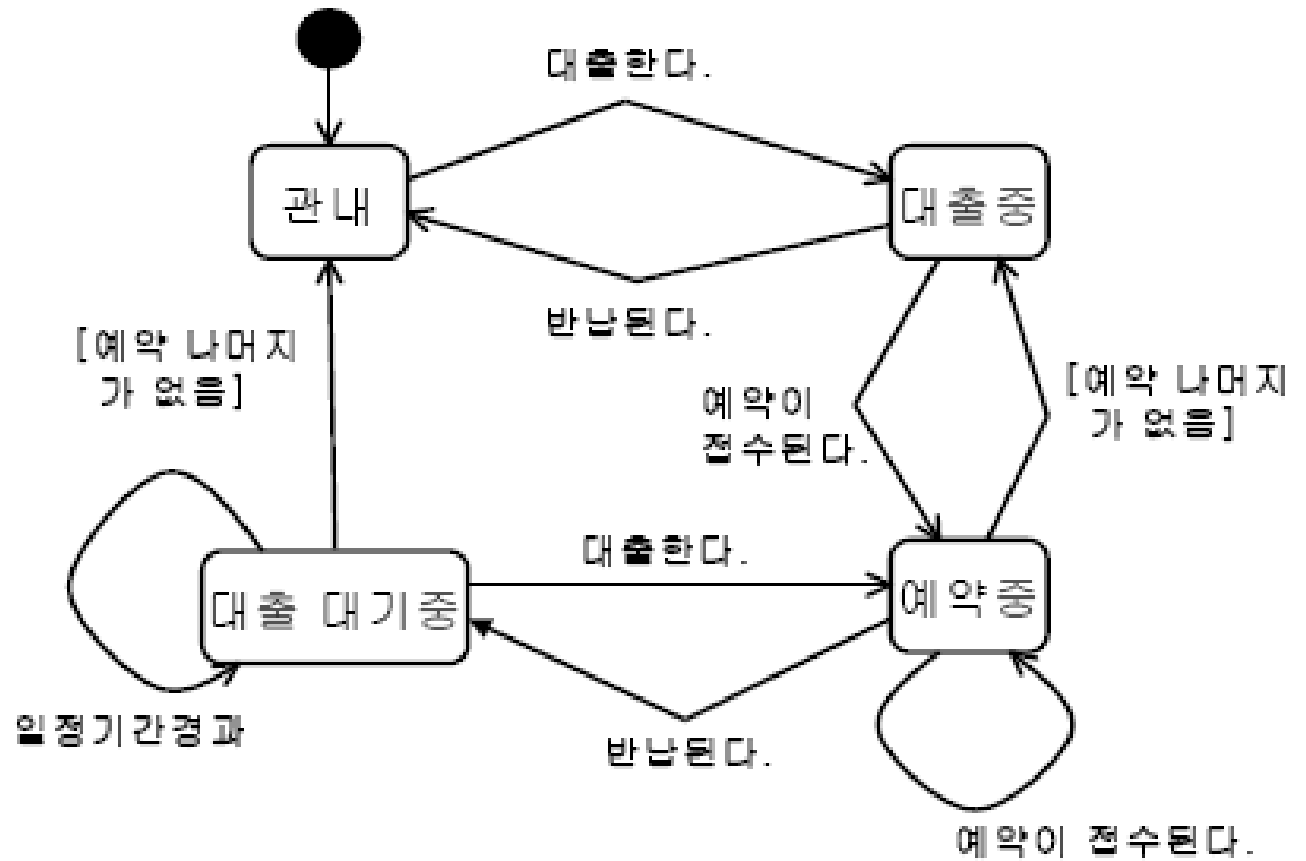
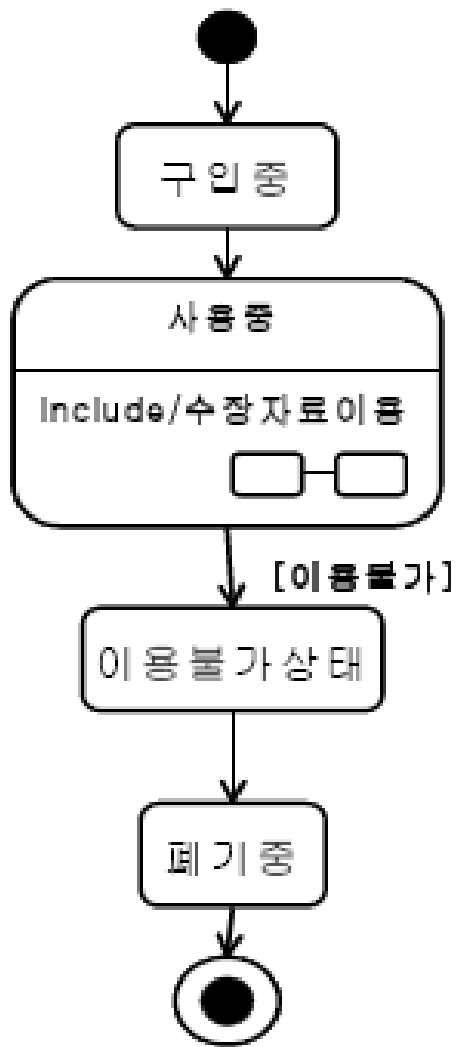
- 각각 대출과 예약을 나타내는 객체
- 대출/예약되는 시점에서 해당 내용이 결정되며, 이후에 변경되지 않음.
- ➔ 시간의 흐름에 따라서 변경되는 상태가 없음.
- ➔ 생명주기 작성 불필요

# “수장자료”의 생명주기

1. (이용자의 요청 또는 무엇인가에 의해) 구입하려고 한다.
2. 구입후 이용된다.
3. 사용하다가 노후/훼손되어 이용 중지
4. 이용중지되면 폐기









# “예약취소”를 추가

---

- 어떤 모델들을 수정/추가?
  - UC Diagram에 “예약을 취소한다” UC를 추가
  - “예약한다” UC를, “통지한다”, “예약한 책을 대출한다”, “예약을 취소한다” 등으로 분해
  - UC Scenario 추가



## 3.7 개념객체의 책임

---

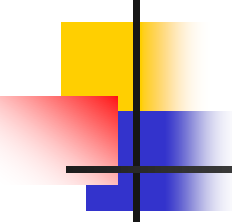
- 각 객체에는 어떤 “역할”이 맡겨짐.
  - “Responsibility”(책임, 책무)라고도 부름.
    - 인지(認知)책임 & 실행(實行)책임
  - 각 역할은 명확해야. 불명확하면, 부작용 파급됨.
- “이용자”객체의 책임은 무엇?
  - “이용자” = 도서관이용하는 사람의 동작수행
    - 이용자의 이름, 연락처 등의 기본정보를 인지
    - 책의 대출, 반납, 예약 등의 동작 수행



# 위임과 협조상대

---

- “대출하고 있는 책의 상세정보”는 어떻게 알아낼 수 있을까?
  - “수장자료”객체에게 문의
- ➔ “다른 객체의 힘을 빌리는 것” = “위임”
- 어떤 책임을 수행하기 위해 함께 작업을 수행하는 객체 = “협조상대”



# “이용자”객체의 책임(협조상대)은?

---

- 이름, 연락처 등의 기본정보를 인지
- 기본정보변경시 통보
- 소장자료 대출/반납(대출, 소장자료)
- 대출한 자료 또는 대출중인 자료에 대해 인지(대출, 소장자료)
- 책 등을 예약, 예약취소(예약, 소장자료)
- 예약한 책 또는 예약중인 책 등에 대해 인지(예약, 소장자료)



# 책임의 갯수

---

- 1개 객체의 책임
  - 20~30개 ➔ 과도, 혼란
  - 수~수십개가 적당



## “수장자료”객체의 책임(협조상대)

---

- 제목, 저자 등의 기본정보를 인지
- 이용자에 의해 대출, 반환된다(이용자, 대출)
- 누구에게 언제까지 대출되고 있는지를 인지(이용자, 대출)
- 이용자에 의해 예약, 예약취소(이용자, 예약)
- 누구에 의해 예약되어 있는지를 인지(이용자, 예약)



# “대출”객체의 책임

---

- “대출”객체 ➔ 한번의 대출을 나타냄

1. 언제 무엇이 누구에게 대출되었는지를 인지하고 있다.
2. 언제가 반납기한인지를 인지하고 있다.
3. 기한을 초과하고 있는지 여부를 인지한다.





# “예약”객체의 책임

---

- “예약”객체 ➔ 한 개의 예약을 나타냄.
1. 언제 무엇을 누가 예약했는지를 인지하고 있다.
  2. 예약에 대한 기한이 경과했는지 여부를 인지한다.



# 문제

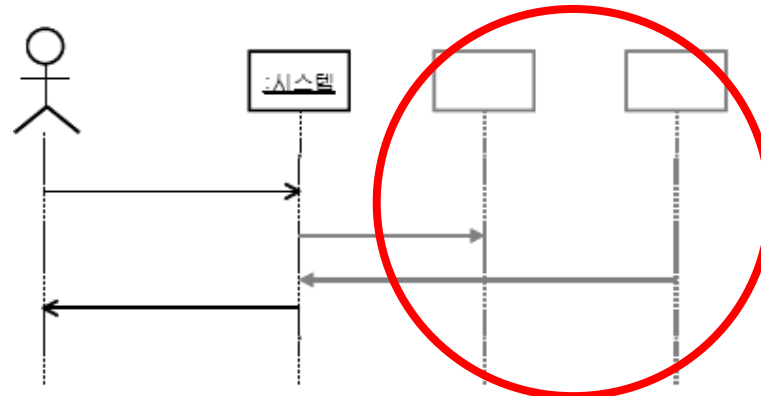
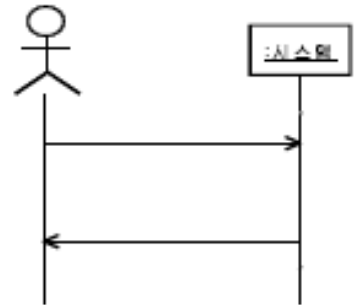
---

- 여러사람들이 예약하고 있던 책이 반납되었을 경우, 다음에는 누구에게 대출될 것인지는 어떤 객체에게 문의하면 될까?
  1. “예약”객체
  2. “이용자”객체
  3. “책”객체
  4. “도서관사서”객체
  5. 답없음.

## 3.8 개념객체와 UC Scenario

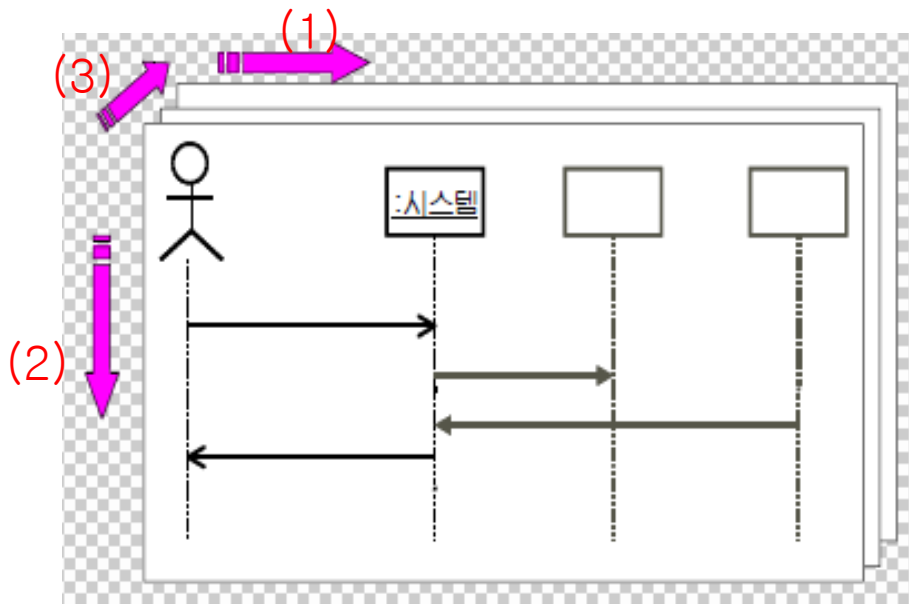
- “도서관관리시스템” 개발 흐름
  - Actor들 사이의 수수작용(WorkFlow) → AD
  - Actor와 시스템간의 수수작용 → UC Scenario + Sequence Diagram

→ SD에 개념객체들을 추가



# 시스템 내부에 존재하는 개념객체들

- 개념객체들을 조합하여 시스템전체가 올바르게 동작하는지를 확인

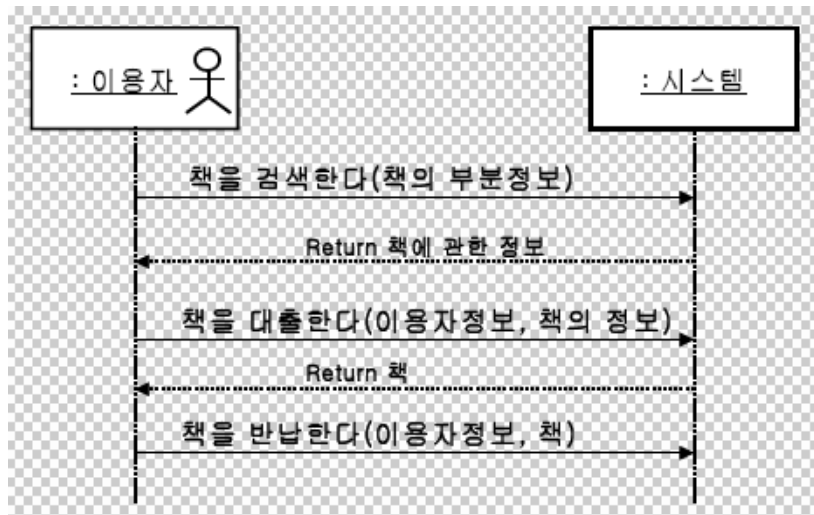


## SD를 중심으로 상세화

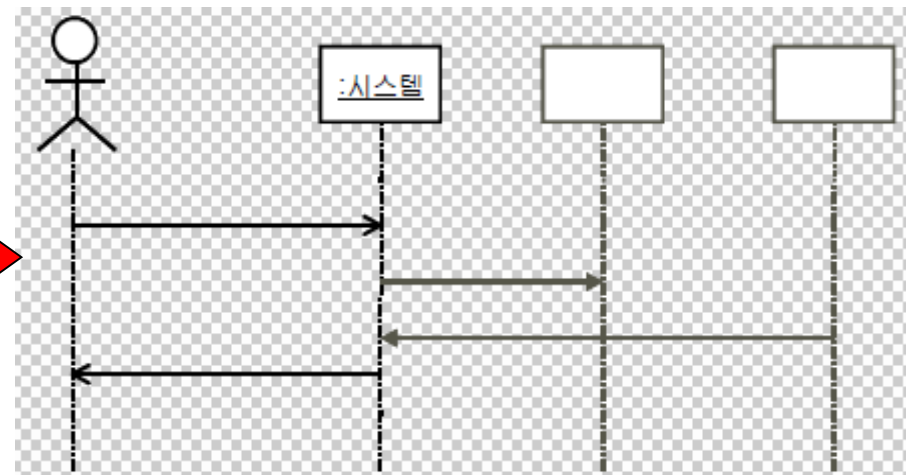
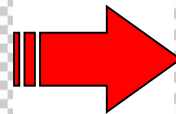
- ① 횡적 확장
  - SD에 객체를 상세화/분열/추가
- ② 종적 확장
  - 객체들간의 수수작용 상세화
- ③ 자체적 확장
  - SD자체를 상세화
  - 보다 다양한 상황/시나리오 고려

## 3.9 개념시나리오

- “개념시나리오” = UC Scenario의 확장판
  - Actor와 시스템간의 수수작용은 불변
  - 시스템 내부를 개념객체들의 집합으로 표현

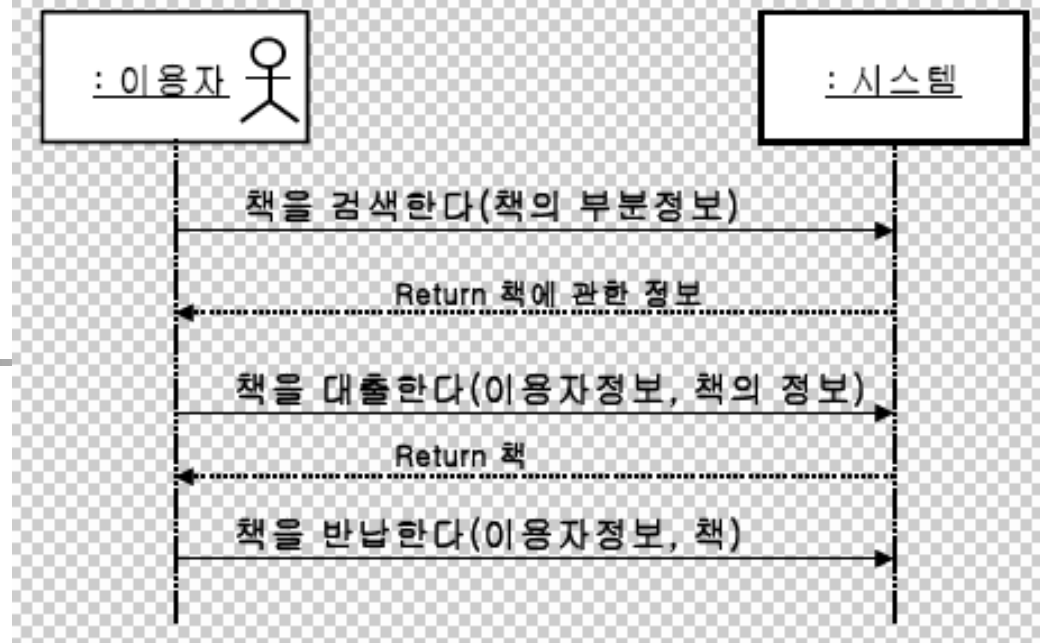


UC Scenario



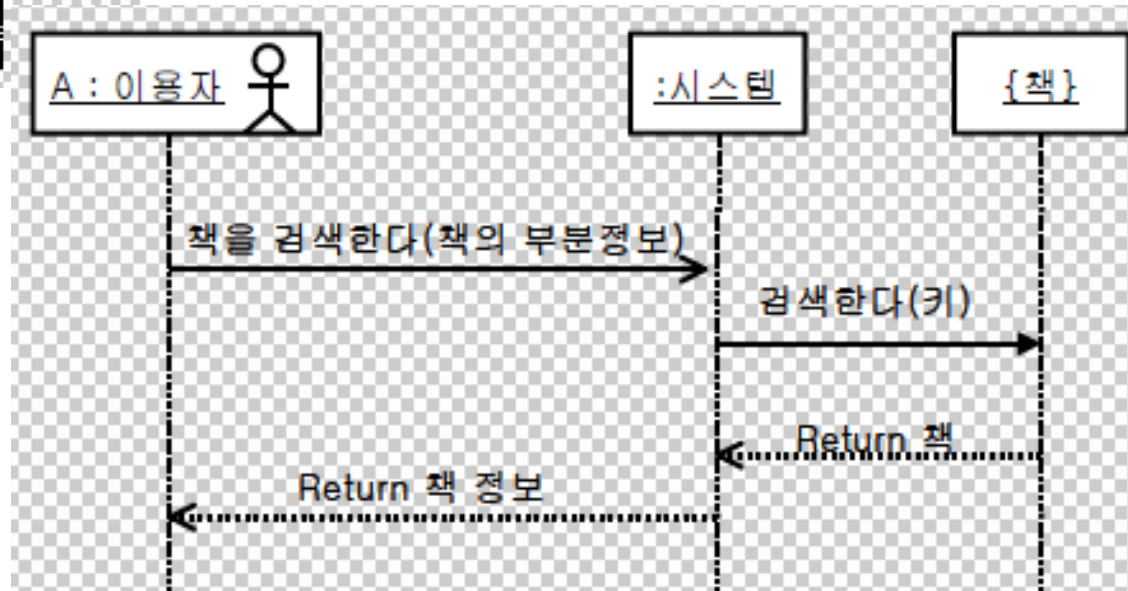
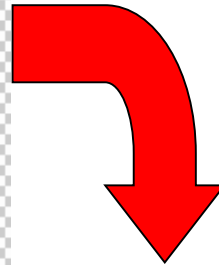
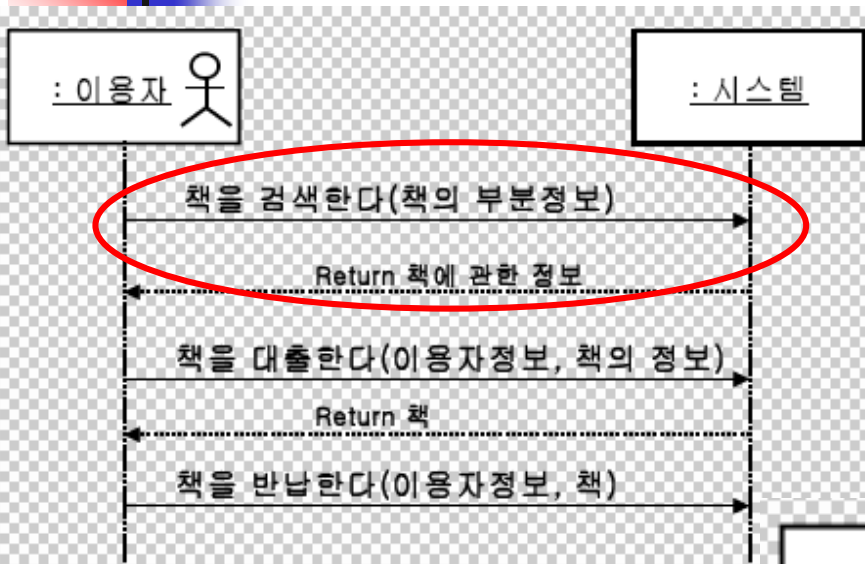
개념 Scenario

# “책을 검색한다” 메시지

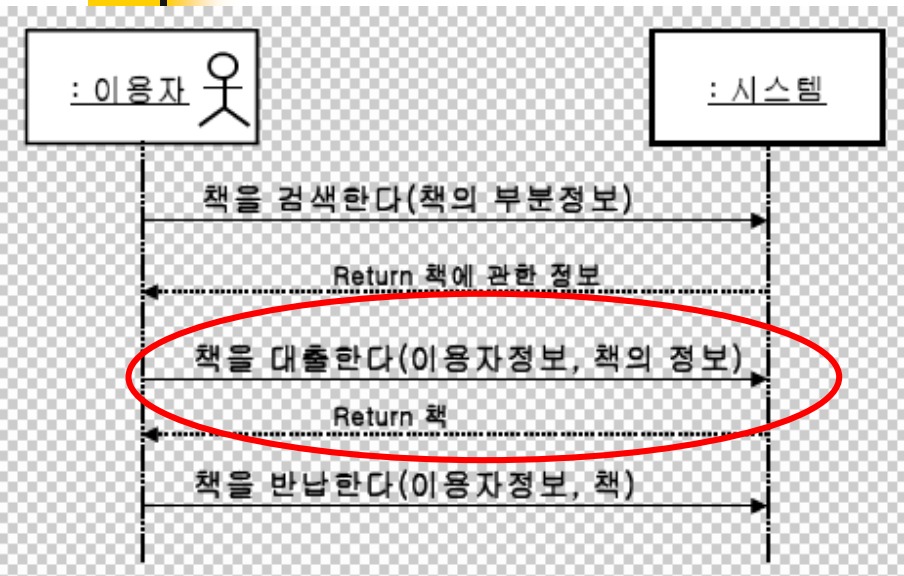


- 시스템내부의 객체들간의 수수작용으로 분해
  - “책을 검색”하기 위해서는, 시스템내부에서 어떤 객체(들)에게 무엇을 의뢰???
  - ➔ “책”객체 (“수장자료”객체)
  - 그러나, “책들”객체는 어떻게 표현?
  - ➔ {책}

# “책을 검색한다” 메시지의 확장

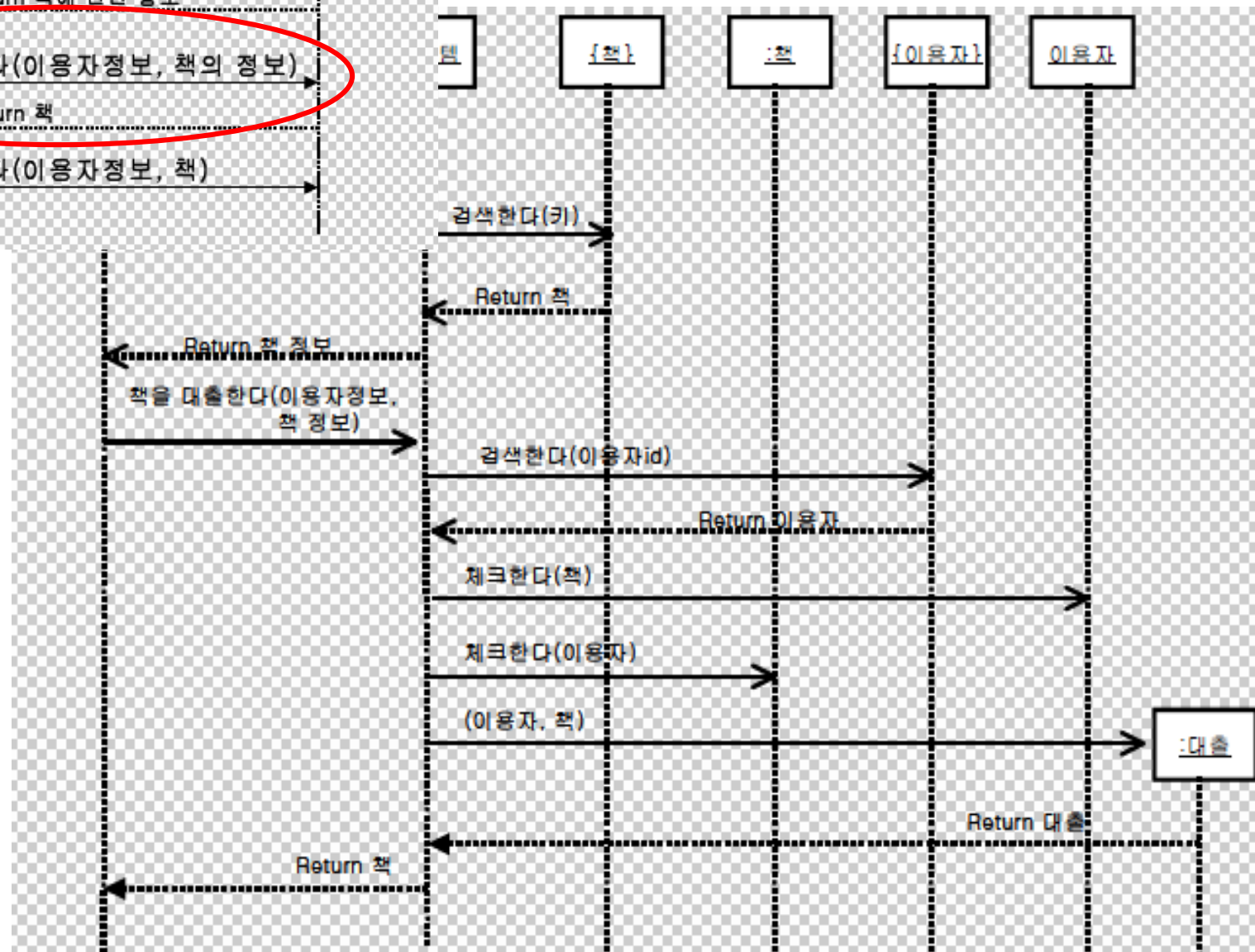
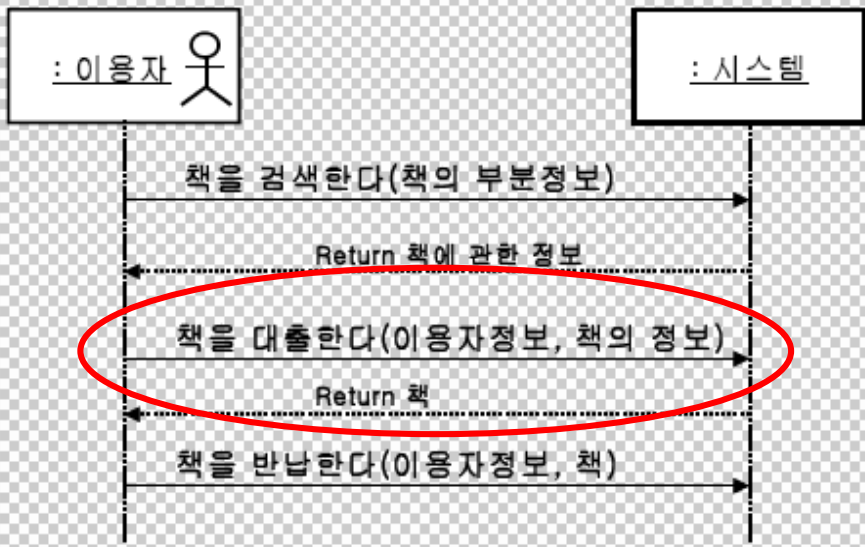


# “책을 대출한다” 메시지의 확장



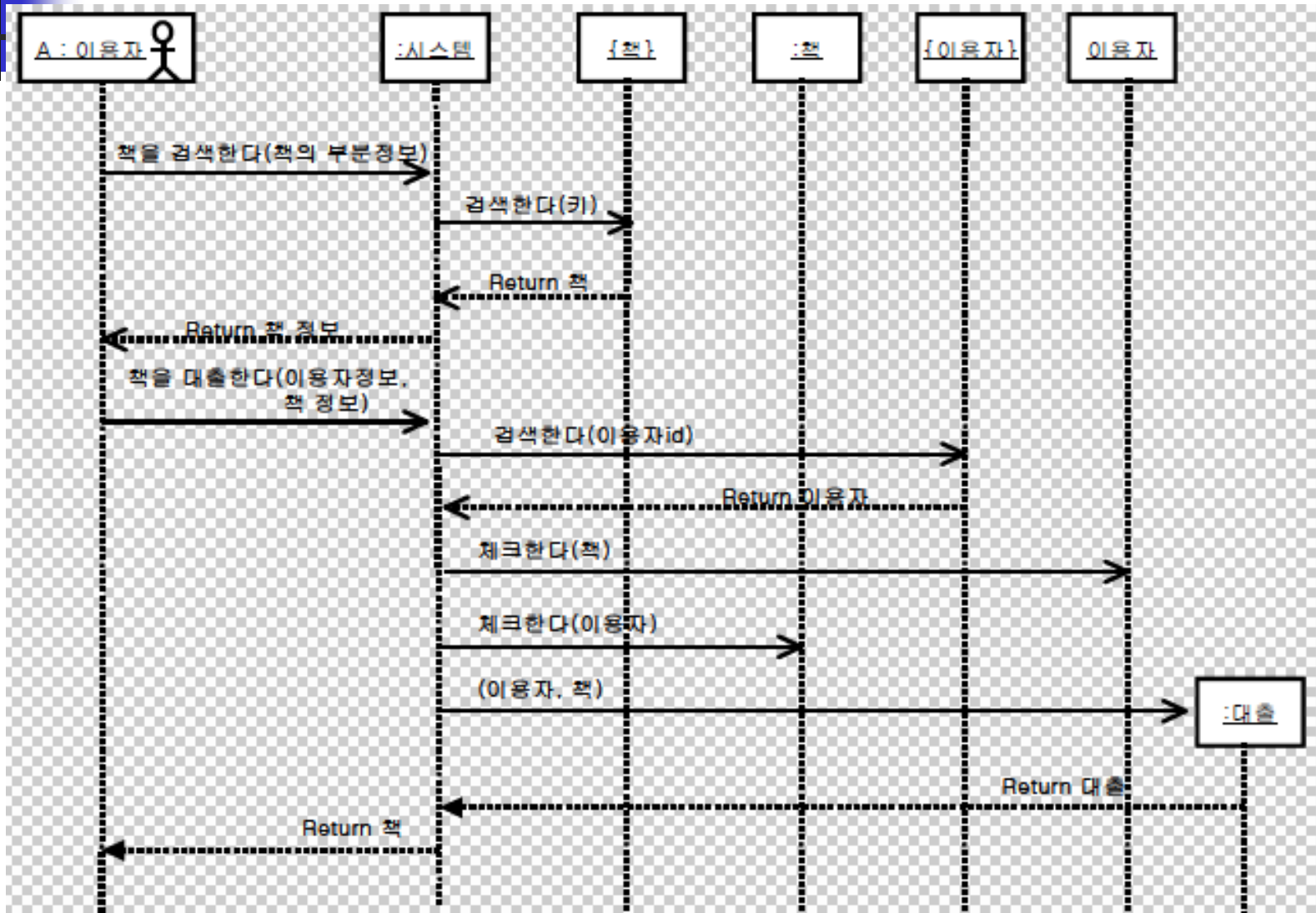
- 사전조건
  - 이용자를 찾아내서 대출가능여부 체크
  - 책을 찾아내서 대출가능여부 체크
- 위 2개조건 모두ok ➔ “대출”개념객체 생성



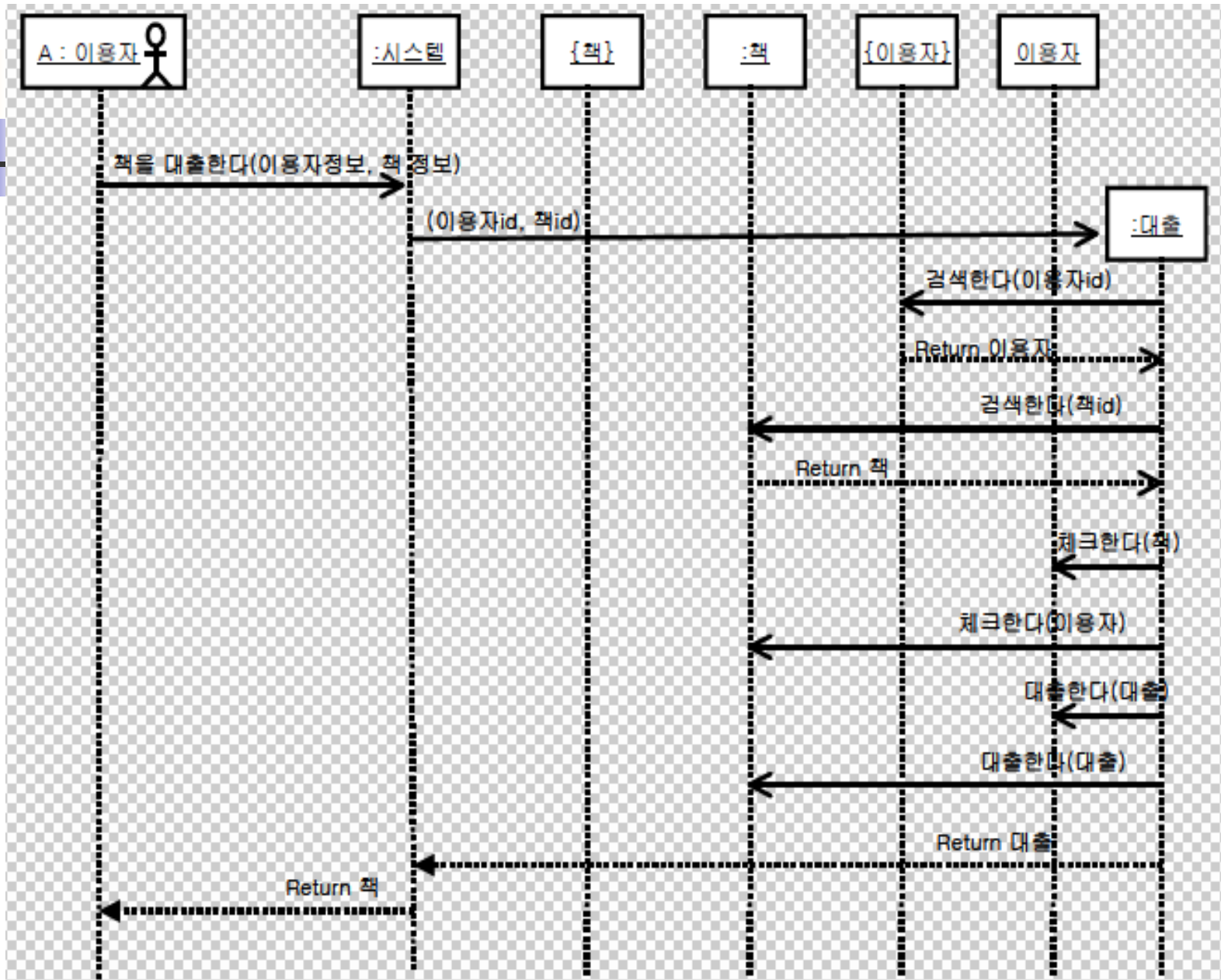


# 책임에 대한 체크

- 각 개념객체들이 책임을 올바르게 수행하고 있나?
  - 시스템이 모든 일을 수행



# 수정판



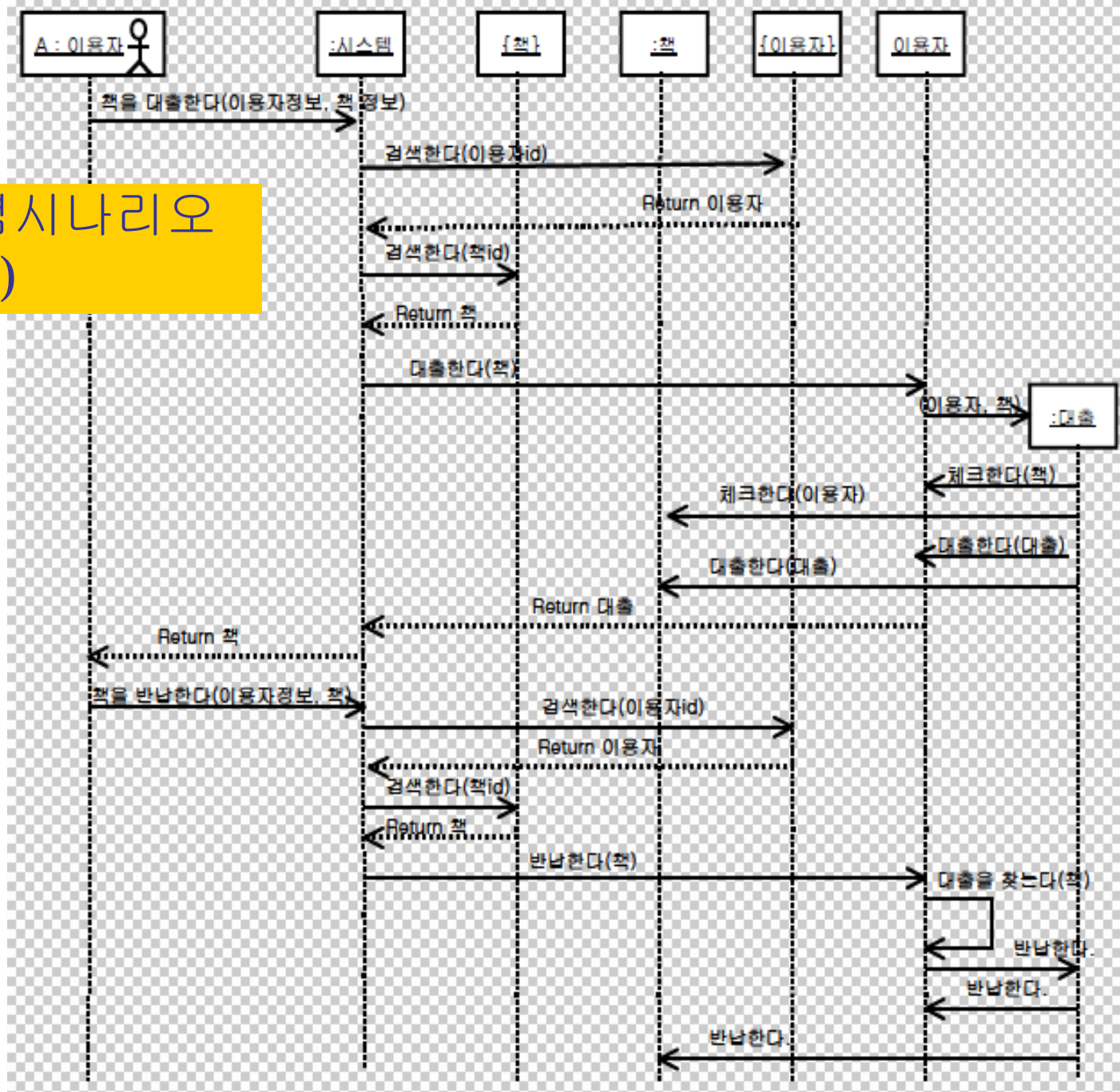


# “반납한다”에 대한 개념시나리오

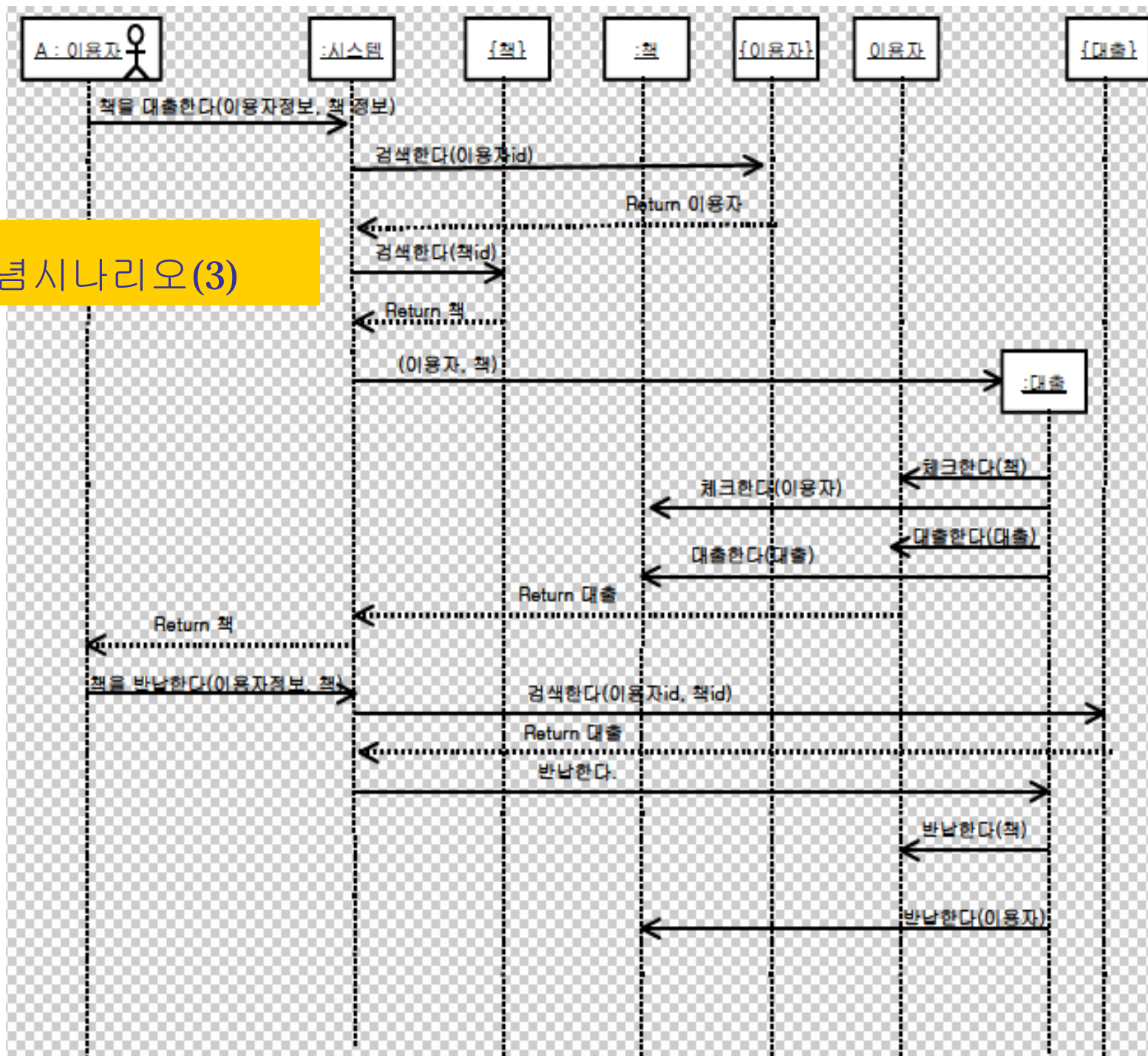
---

- “대출”개념객체를 어떻게 찾아낼까?
  1. 이용자로부터 시작하여 추적하면서 검색
  2. 책으로부터 시작하여 추적하면서 검색
  3. 대출을 직접 검색
  
- 1과 2 방법은 (수정판)“대출한다”개념시나리오와 모순!
  - 앞서 “대출”개념객체를 직접 작성했으므로...

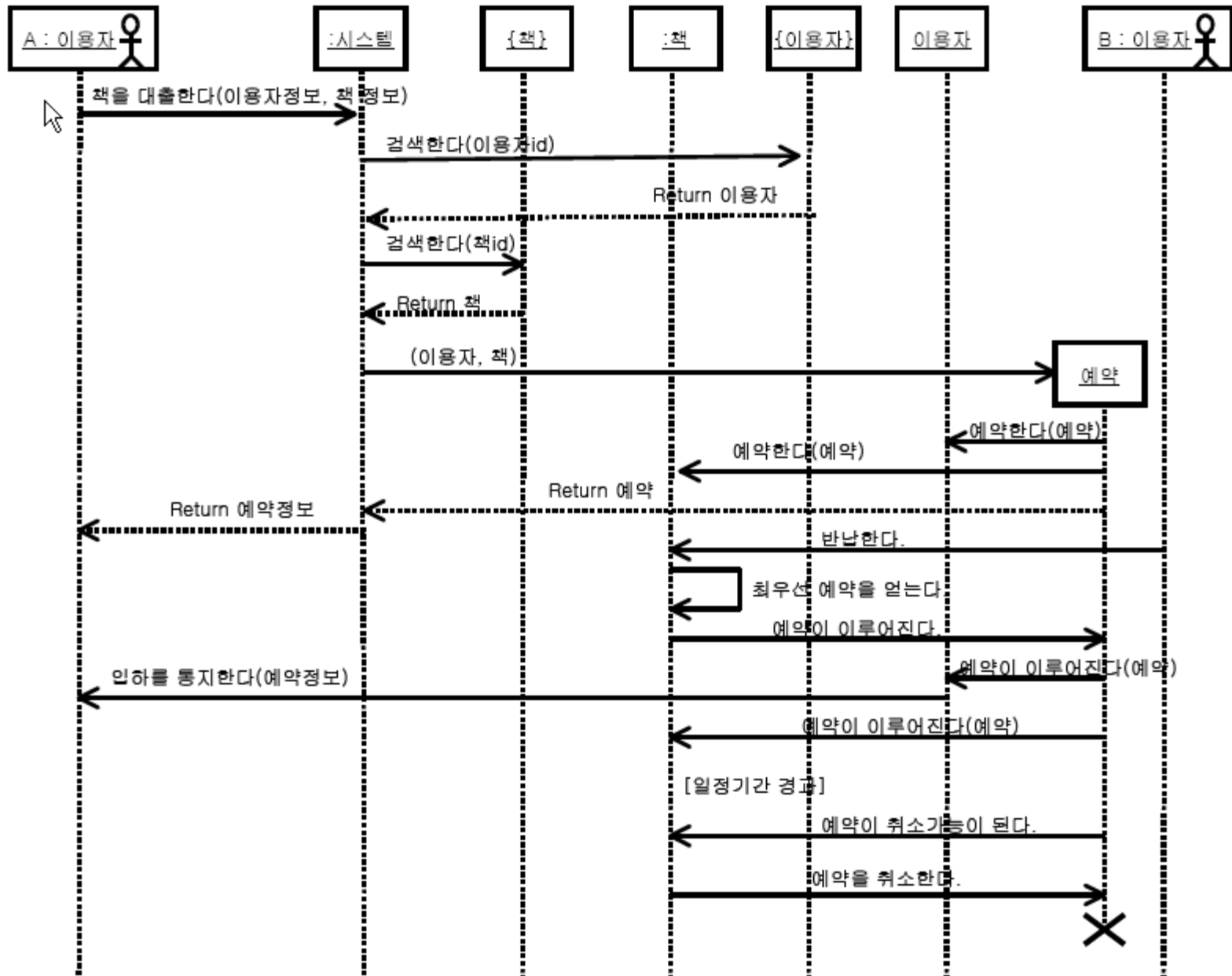
# “반납한다” 개념 시나리오 (1 or 2)



# “반납한다” 개념시나리오(3)



# “예약한다”에 대한 개념시나리오





## 3.10 개념객체의 정적구조

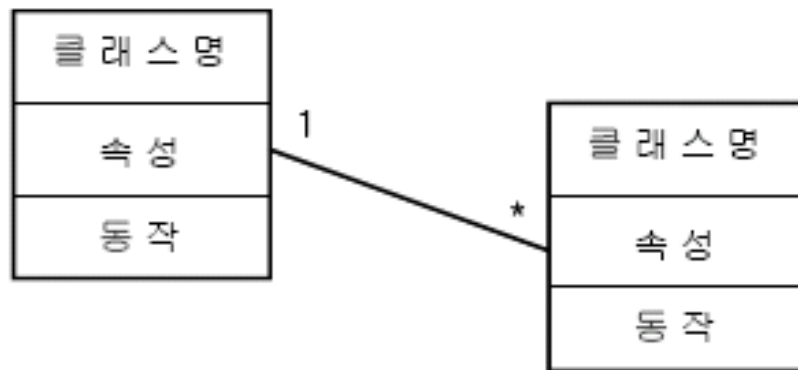
UML 정적구조	CRC
클래스	클래스
내부구조	책임
속성	지식책임
동작	실행책임
관계구조	
관련	협조상대
기타	특별히 없음.

<표 9-1> UML의 정적구조와 CRC

Class Diagram으로 표현



# Class Diagram



<그림 3-62> 정적구조도



## 3.11 개념객체의 속성

---

- 해당 개념객체가 무엇에 대해서 인지해야 하는가를 나타낸 것
- 힌트
  - 개념객체의 **인지책임**
  - 개념객체의 생명주기 상에 있는 **상태**



# “이용자” 개념객체

---

이용자
이름 연락처

- “이용자” 개념객체가 가져야만 되는 정보
  - 이름, 연락처, ... (기본정보)
  - 현재 대출중인 책에 관한 정보(“책” 개념객체와의 관계)
  - 현재 예약중인 책에 관한 정보(상동)
  - 과거에 대출했던 책에 관한 정보(상동)



# 기타

---

수 장 자 료
이 름

대 출
대 출 일 반 납 일

“누가, 무엇을, 언제 대출했나? 언제 반납했나?”에 관한 정보

예 약
예 약 일 입 하 일 취 소 완 료



## 3.12 개념객체의 동작

---

- 해당 개념객체가 어떤 메시지를 수신할 수 있는가에 대한 정보
  - 개념객체의 “실행책임”
  - 개념객체의 생명주기 상의 “이벤트”
  - 시나리오의 “메시지”

# “이용자”개념객체

“이용자”개념객체의 실행책임

- 대출한다
- 대출자격을 체크한다
- 반납한다
- 예약한다

<그림 4-22~26참조>

이용자
이름 연락처

이용자
이름 연락처
예약한다. 대출가능한가? 대출한다. 반납한다.

이용자
이름 연락처
예약한다(예약) 대출가능한가?(수장자료) 대출한다(대출) 반납한다(대출)

매개변수를 추가

# 기타

수장자료
이름

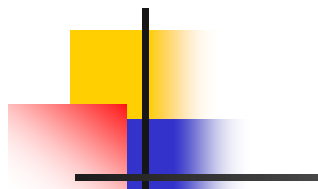
대출
대출일 반납일

예약
예약일 입하일 취소완료

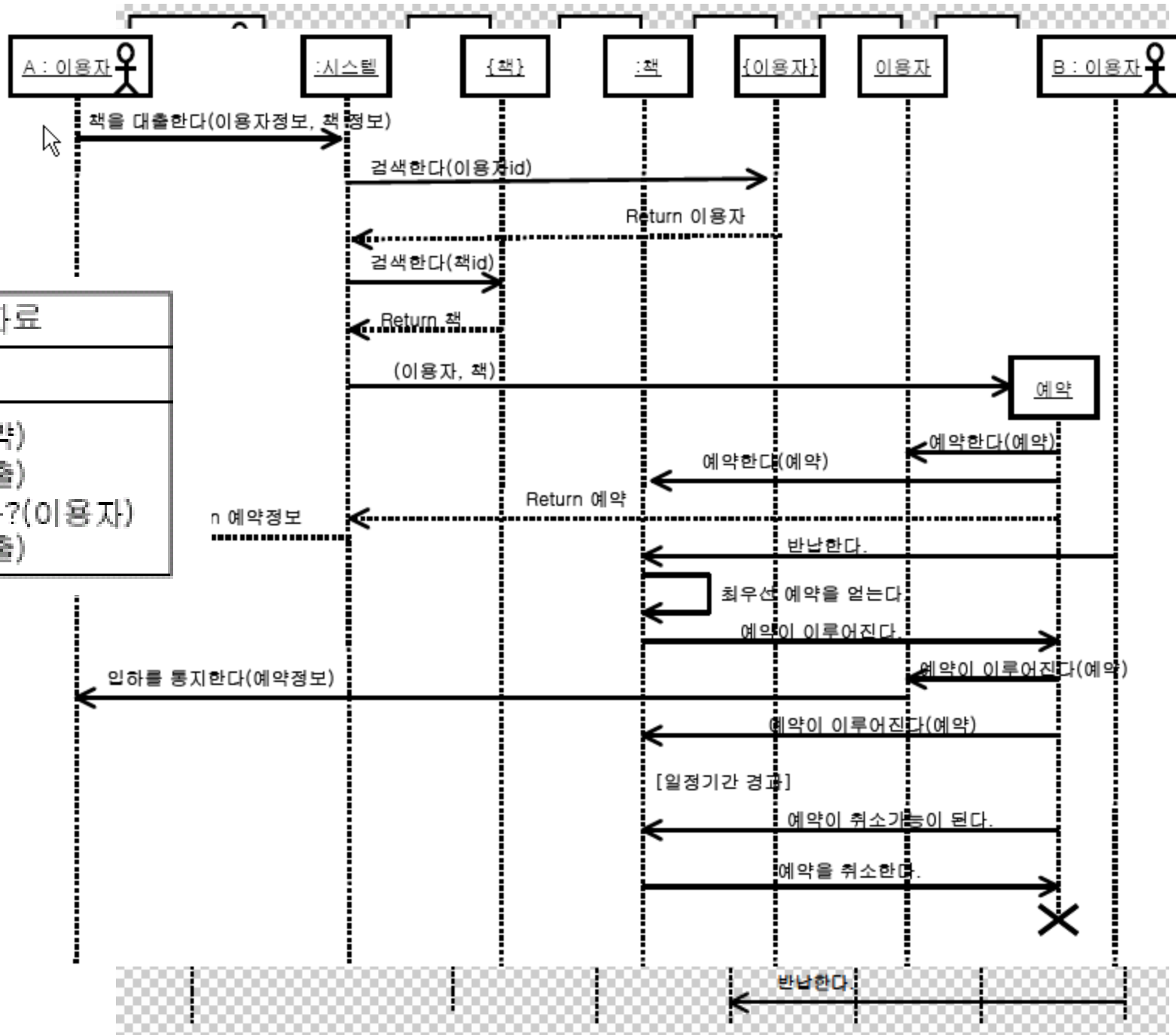
수장자료
이름
예약된다(예약) 대출한다(대출) 대출가능한가?(이용자) 반납된다(대출)

대출
대출일 반납일
반납한다

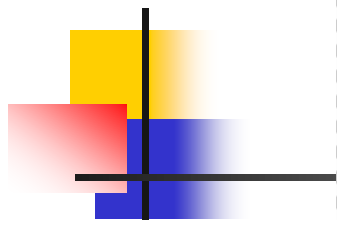
예약
예약일 입하일 취소완료
취소한다 예약이 이루어진다.



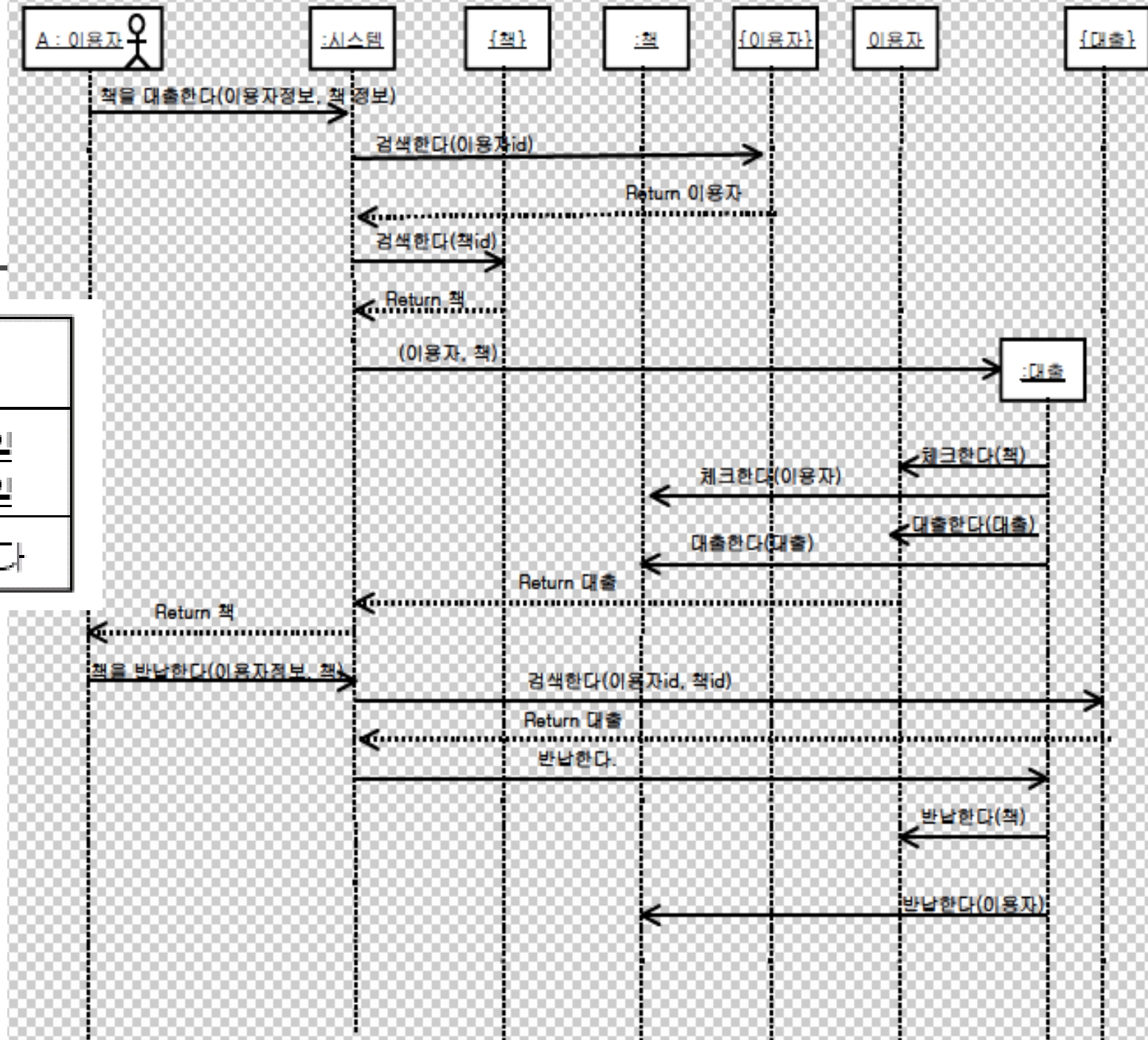
수장자료
이름
예약된다(예약)
대출한다(대출)
대출가능한가?(이용자)
반납된다(대출)

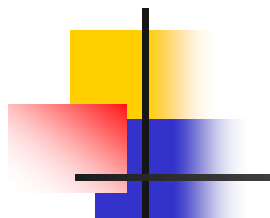




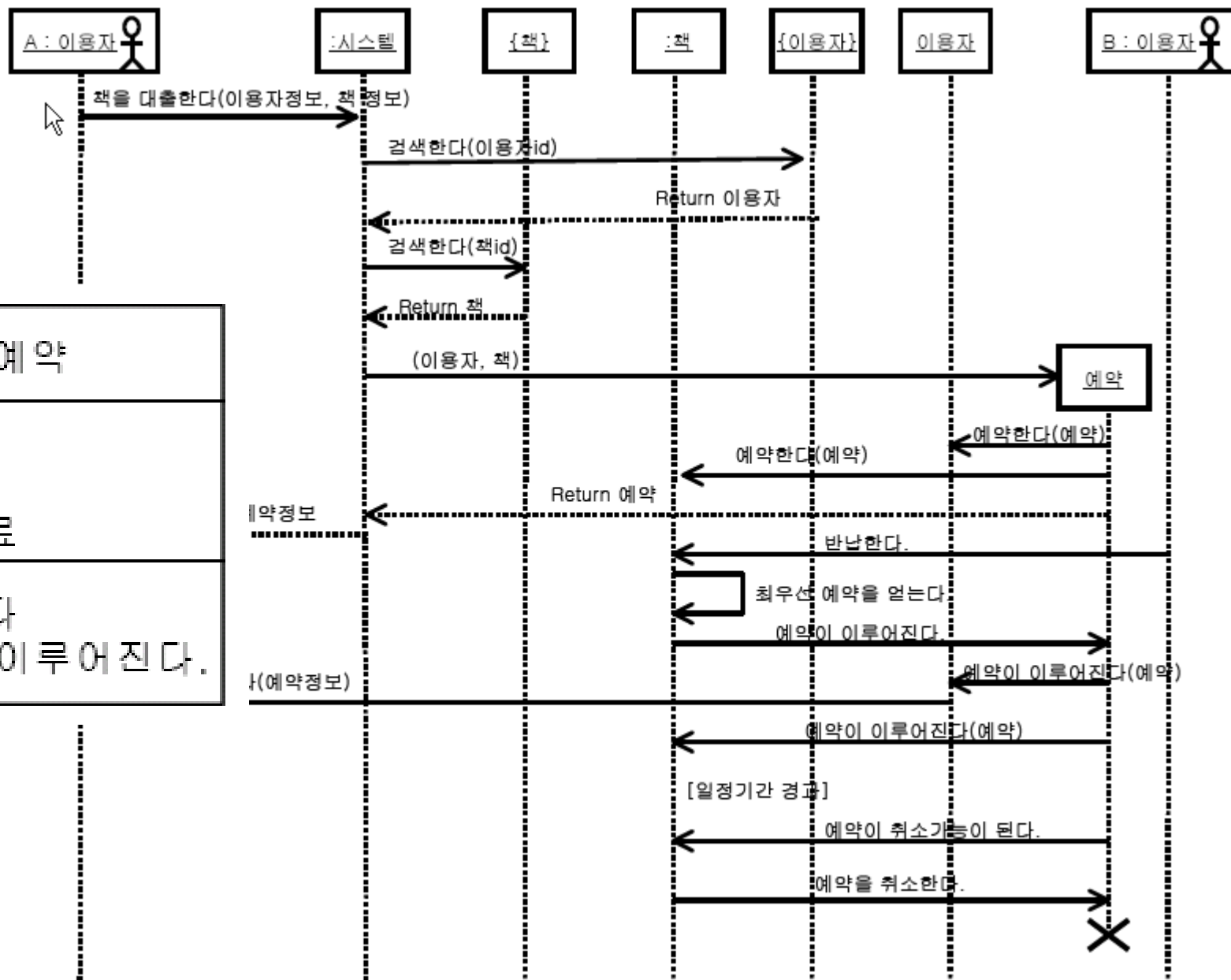


대출
대출인 반납인
반납한다



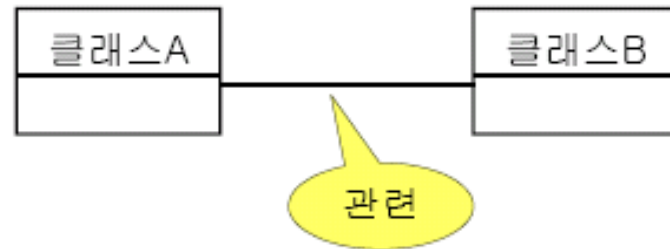


예약
예약일 입하일 취소완료
취소한다 예약이 이루어진다.

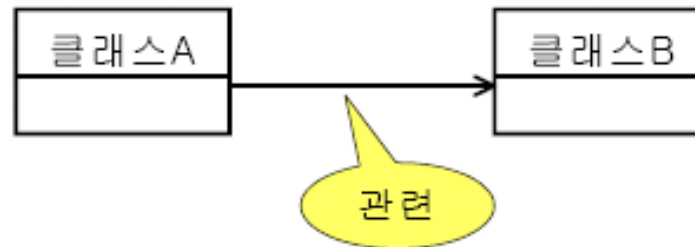


## 3.13 관련

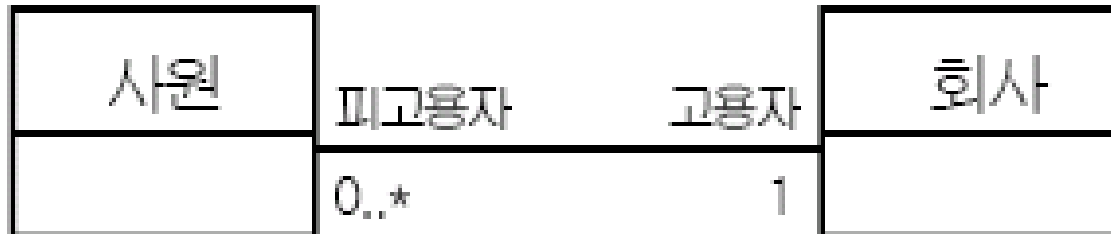
클래스A의 객체와 클래스B의 객체가 서로 “인지”하고 있다.



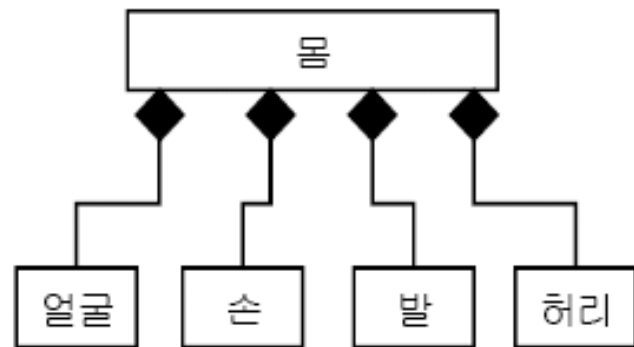
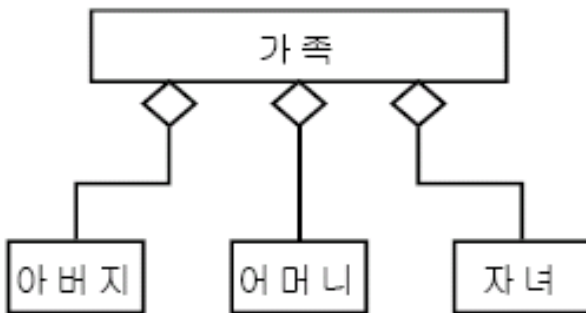
클래스A의 객체는 클래스B의 객체를 “인지”하고 있다.(그 역은 성립하지 않음)



## 기타 부가정보 표현



## 3.14 集約과 合成





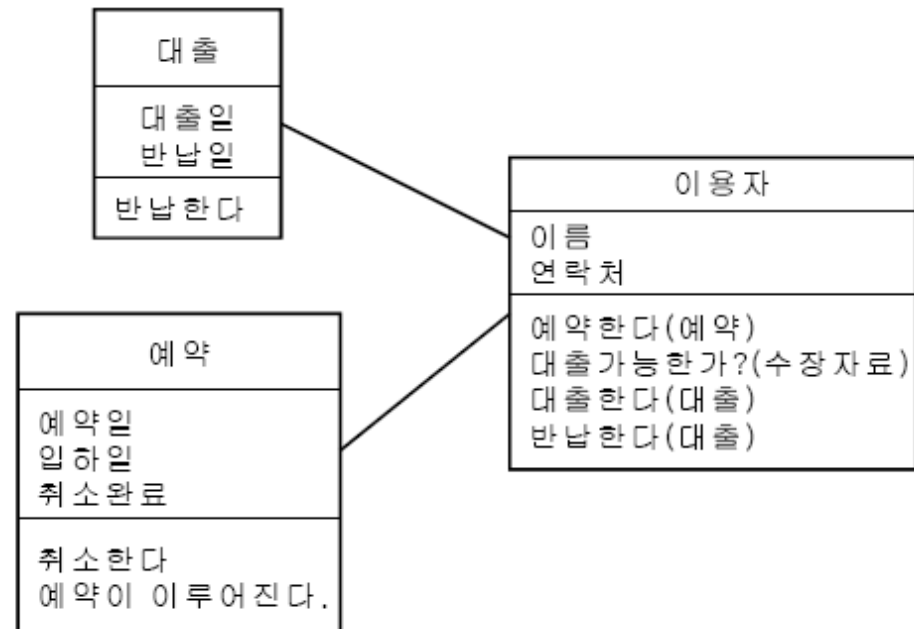
## 3.15 개념객체의 관련관계

---

- “관련관계”의 의미
  - 어떤 객체가 자신의 책임수행을 위해, 다른 객체와 상호수수작용을 하는 것
- 파악방법
  - 개념객체의 실행책임
  - 시나리오의 메시지, 매개변수
  - 개념객체의 생명주기(상태)
  - 개념객체의 속성 (**birthday : Date**)

# Example(“이용자”)

- 이름, 연락처 등의 기본정보(속성)
- 현재 대출하고 있는 책의 정보
  - 이용자 --- 대출
- 현재 예약하고 있는 책의 정보
  - 이용자 -- 예약
- 과거 대출했던 책의 정보
  - 이용자 -- 대출





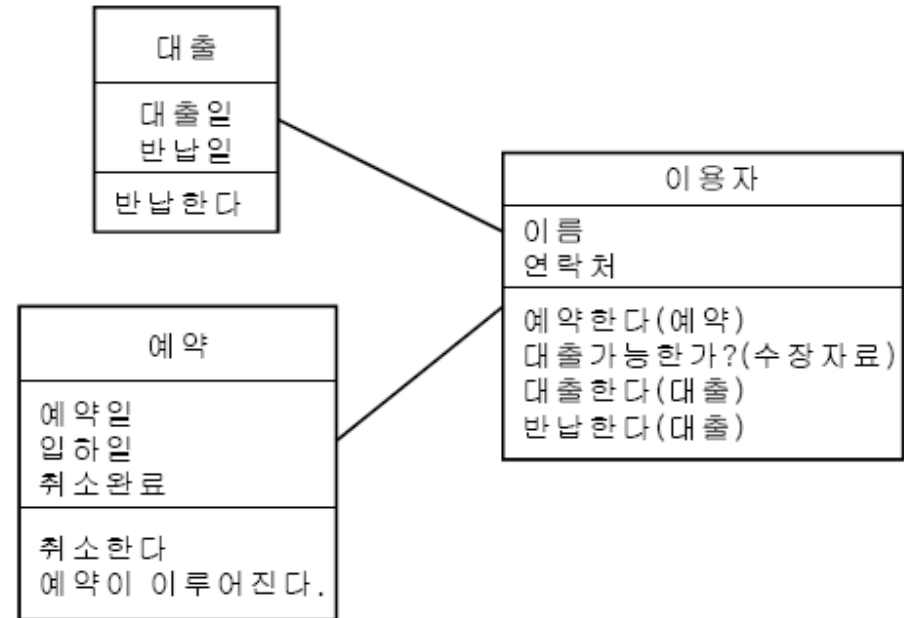
# 관련관계 파악시 고려사항

---

- 관련관계의 이름 또는 역할명
  - 관련관계는 무엇을 나타내고 있는가?
- 방향성
  - 어느쪽에서 어느쪽을 인지하는가?
- 다중도
  - 몇대몇의 관계인가?

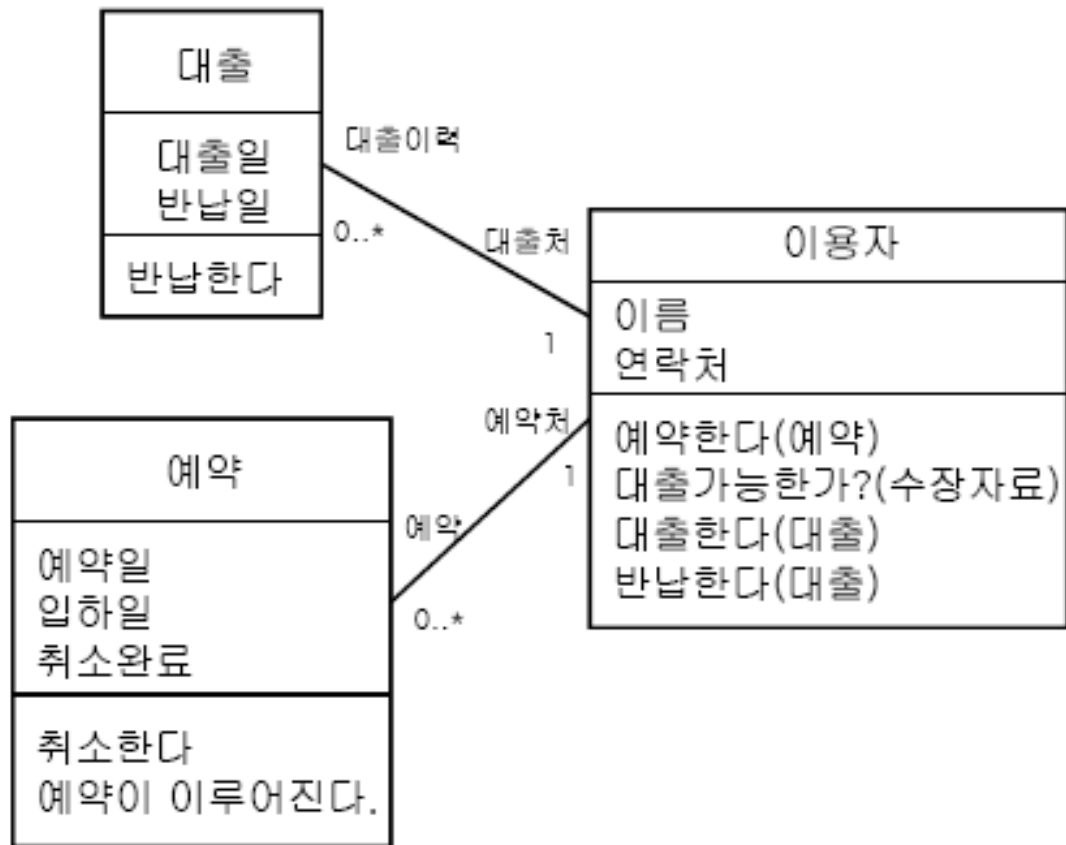


# 이용자 vs. 대출



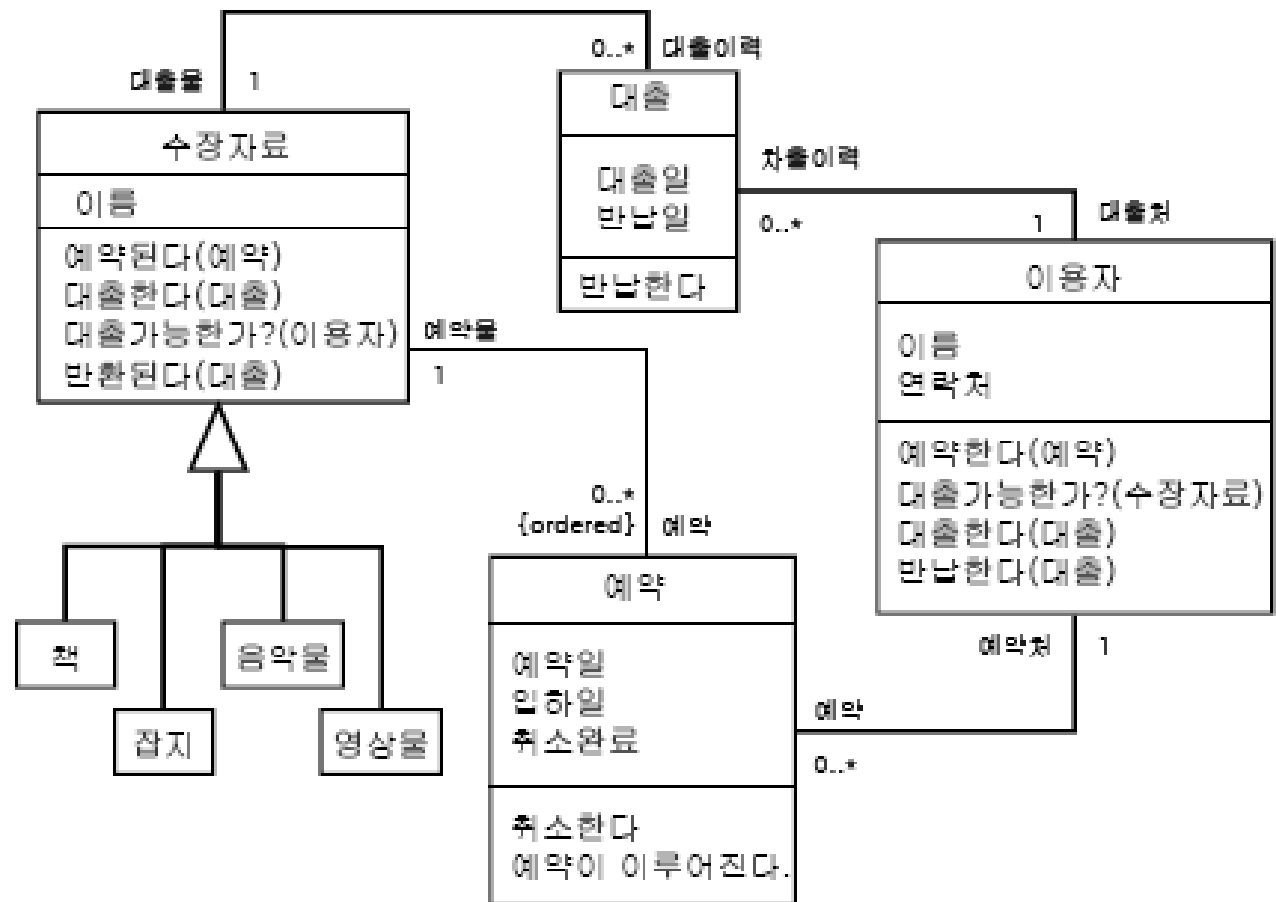
- 이용자에게 있어서 대출은,
  - 이용자가 지금까지 대출했던 수장자료를 나타냄.
  - 역할명 = “대출이력”
- 대출에게 있어서 이용자는?
  - 역할명 = “대출처” 또는 “대출인”
- 방향성 = “양방향”
- 다중도 = ???

# 관련관계 = 상호수수작용통로

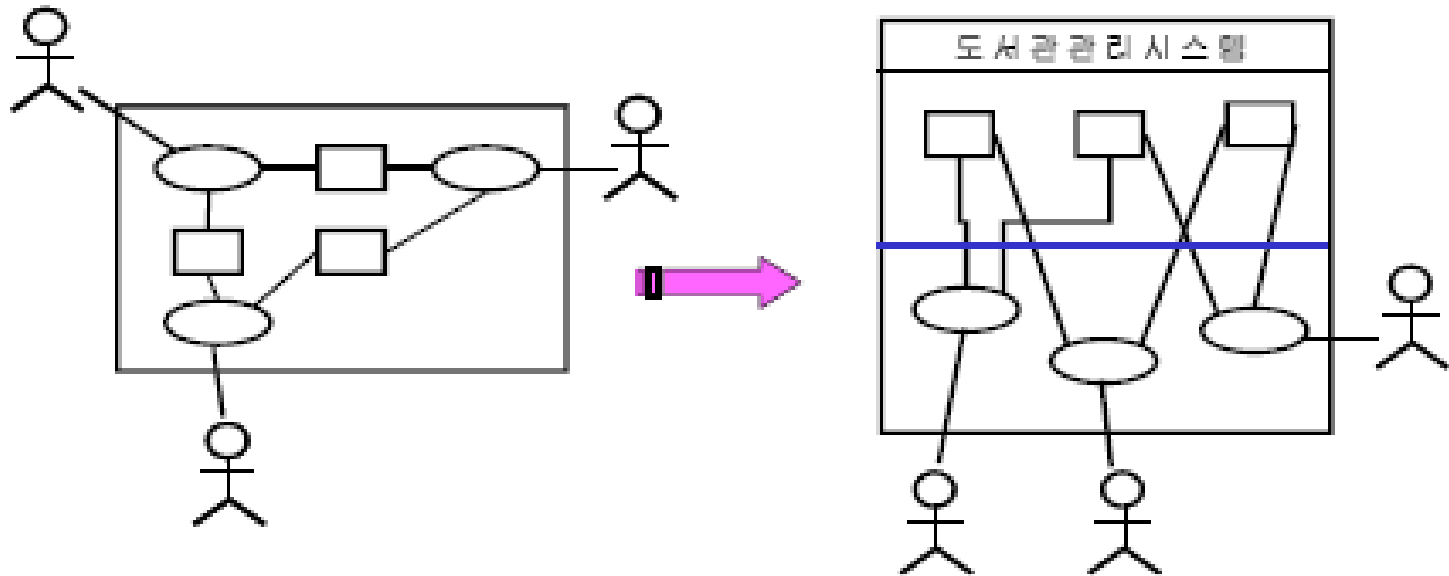


# 문제

- “수장자료”와 “대출”, “예약”들사이의 관련관계는?



## 3.16 정리..정리..총정리..



“시스템객체”

- 속성 = 개념객체들
- 동작 = Use Cases

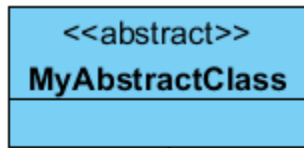


## 3.17 개념모델링의 종료

---

- 지금까지는,
  - 개발 시스템을 외부에서 살펴보면서 바깥쪽의 형태를 구축 ➔ “개념모델링”
  - 미고려사항 ➔ 4장에서 언급예정
    - 어떻게 시스템을 만들것인가?
    - 시스템의 내부를 어떤 구조로 만들것인가?
- 개념모델링은 계속되어야 한다...쭈~욱!

# Class Diagram vs. Java Implementation



```
public abstract class MyAbstractClass{ ... }
```

```
public class MyDerivedClass
    extends MyAbstractClass{
```

```
    public int att;
```

```
    private static int numberOfObjects;
```

```
    protected double totalPrice;
```

```
    public void myFunction(ReferencedClass r){
```

```
        ....
```

```
    }
```

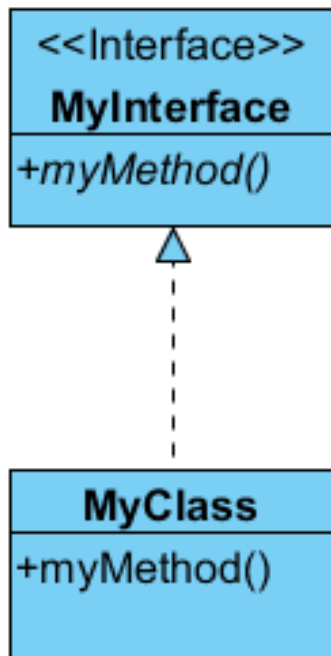
```
    public static double operation(){
```

```
        ....
```

```
    }
```

```
}
```

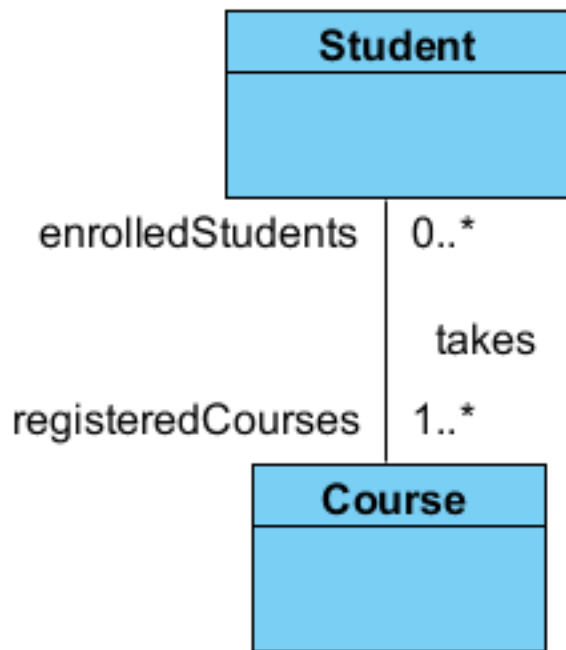
# Class Diagram vs. Java Implementation



```
public interface MyInterface{
    ...
    public void myMethod();
    ...
}
```

```
public class MyClass implements MyInterface{
    ...
    public void myMethod(){
        ....
    }
    ....
}
```

# Class Diagram vs. Java Implementation

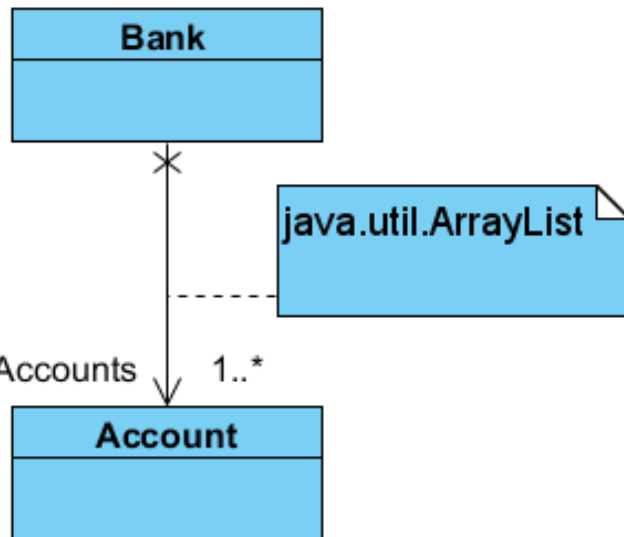


```
public class Student{  
    ...  
    Course[] registeredCourses;  
    ...  
}
```

```
public class Course{  
    Student[] enrolledStudents;  
  
}
```



# Class Diagram vs. Java Implementation



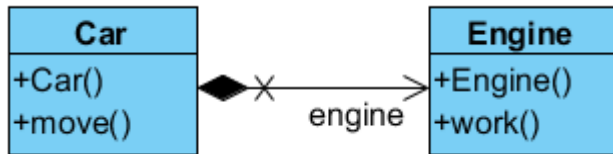
```
import java.util.ArrayList;

public class Bank{
    ...
    public ArrayList<Account> registeredAccounts;

    public Bank(){
        registeredAccounts = new ArrayList<Account>();
    }
    ...
}

public class Account{
    ...
    ....
}
```

# Composition vs. Aggregation

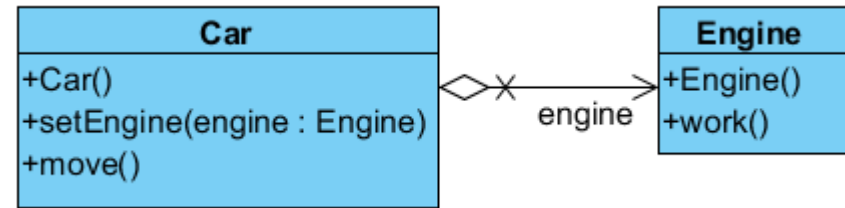


```
// Composition //
public class Car{
    ...
    private Engine engine;

    public Car(){
        engine = new Engine();
    }
    ...
    public void move(){
        engine.work();
    }
}

public class Engine{
    public Engine(){ .... }

    public work(){ .... }
}
```

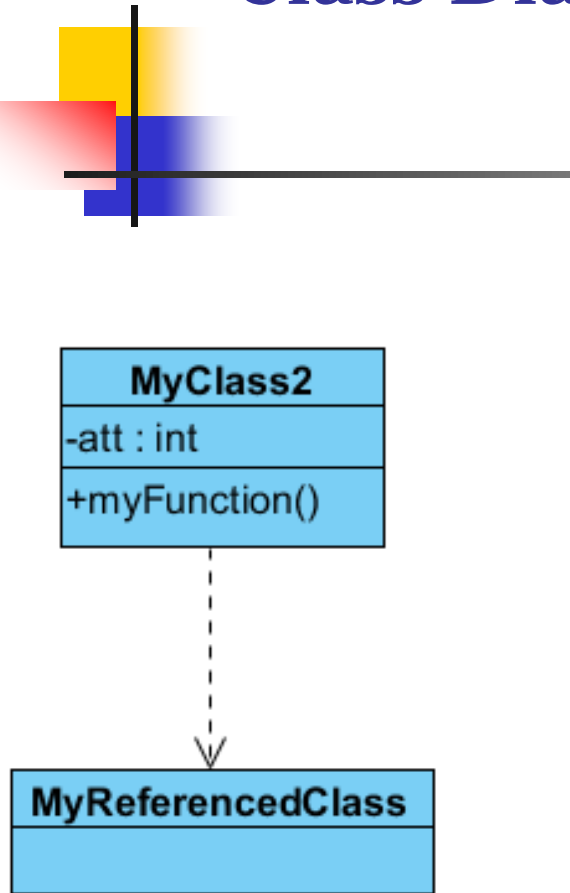


```
// Aggregation //
public class Car{
    ...
    private Engine engine;
    public Car(){ ... }
    public setEngine(Engine engine){
        this.engine = engine;
    }
    ...
    public void move(){
        if(engine != null)
            engine.work();
    }
}

public class Engine{
    public Engine(){ .... }

    public work(){ .... }
}
```

# Class Diagram vs. Java Implementation



```
public class MyClass2{
    ...
    private int att;

    public void myFunction1(MyReferencedClass r){ ... }

    public MyReferencedClass myFunction2( ... ){ ... }

    public void myFunction3( ... ){
        MyReferencedClass m;
        ....
    }
}

public class MyReferencedClass{
    ...
    ....
}
```