PLANTUML 실습

동의대학교 컴퓨터소프트웨어공학과 이종민교수

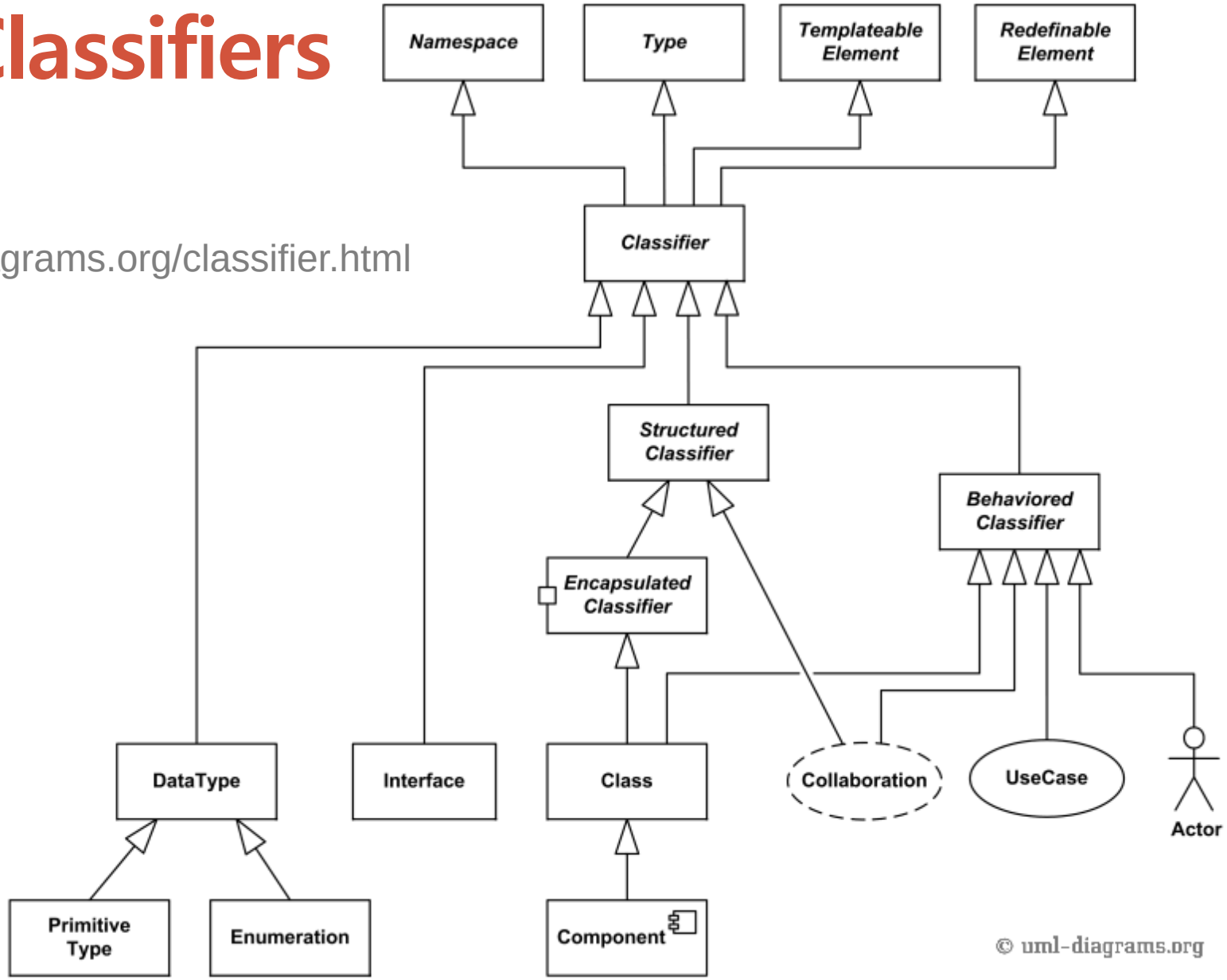
PlantUML 소개

- 텍스트 기반으로 UML 다이어그램을 생성하는 오픈 소스 도구
  - 다양한 다이어그램 지원
  - 쉬운 문법
  - 다양한 출력 형식 지원
- Jar 파일 다운로드
  - https://plantuml.com/ko/download
- 사용법

  $ java –jar plantuml-mit-1.2025.2.jar [cd-01.txt]
- Lanuguage Refernce Guide
  - https://plantuml.com/ko/guide (한글판, 영어판 등)

UML 2.5 Classifiers

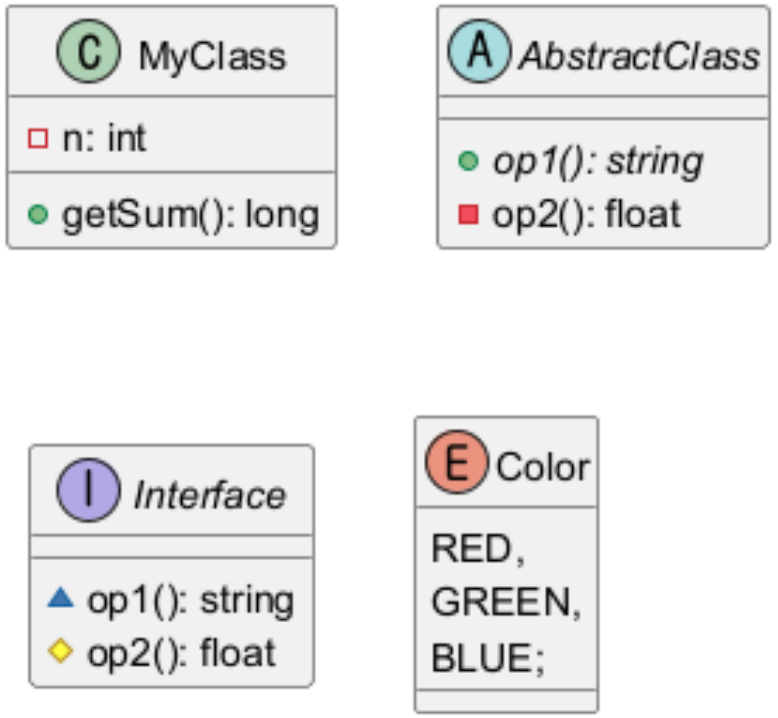
cf. https://www.uml-diagrams.org/classifier.html



클래스 다이어그램 기본 요소

- 클래스, 추상 클래스, 인터페이스, 열거형(enum),

[UML 방식]



[Java 방식]

MyClass
private int n
public long getSum()

AbstractClass
public string op1()
private float op2()

Interface
string op1()
protected op2()

Color
RED(0),
GREEN(1),
BLUE(2);
Attributes
private int n
Opertations
public Color(int n)
public int getValue();

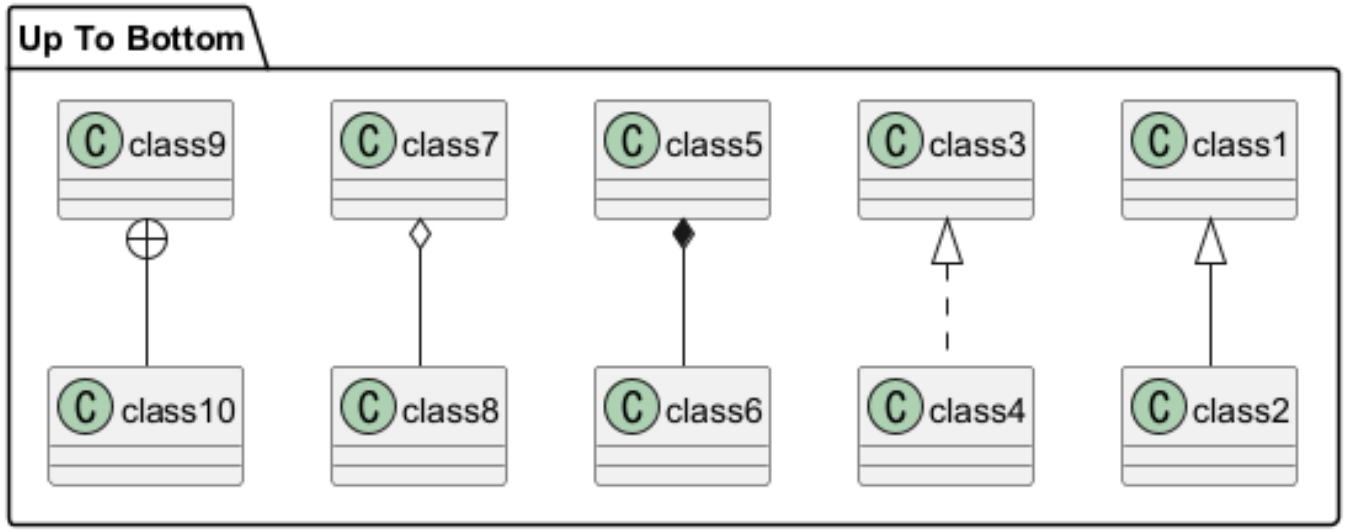
PlantUML 코드 샘플: UML 방식

```
@startuml
' 클래스 (한 줄 주석)
class MyClass {
  -n: int                  /' attribute '/
  +getSum(): long  /' operation '/
}

/' 추상 클래스를
  설명 (여러 줄 주석. 줄 뒤에도 가능)
'/
abstract class AbstractClass {
  +op1(): string {abstract}
  -op2(): float
}

interface Interface {  /' 인터페이스 '/
  ~op1(): string
  #op2(): float
}

' 열거형
enum Color {
  RED,
  GREEN,
  BLUE;
}
@enduml
```

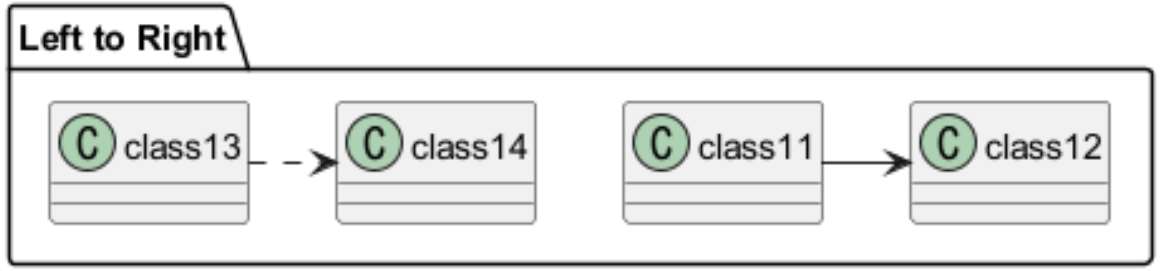
PlantUML 코드 샘플: Java 방식

```
@startuml
' 클래스
class MyClass {
  private int n
  public long getSum()
}

/' 추상 클래스를
  설명
'/
abstract class AbstractClass {
  {abstract} public string op1()
  private float op2()
}

interface Interface {  /' 인터페이스 '/
  string op1()
  protected op2()
}

' 열거형
enum Color {enum Color {
  RED(0),
  GREEN(1),
  BLUE(2);
   /' "-- 설명 --" 또는 "--"만 표시 '/
  -- Attributes --
  private int n
  -- Opertations --
  public Color(int n)
  public int getValue();
}
@enduml
```

클래스 관계



내부 클래스 집합 합성 실체화(구현) 일반화



의존성 연관

클래스 관계

유형	기호	목적
일반화(generalization)	<|--	클래스 상속
구현 (implementation)	<|..	클래스에 의한 인터페이스의 실현
합성(composition)	*--	부분이 전체 없이는 존재할 수 없음 (독점적 소유권)
집합 (aggregation)	o--	부분이 전체와 독립적으로 존재할 수 있음
내부 클래스(inner class)	+--	클래스 내부에 정의되는 다른 클래스 모델링
연관 (association)	-->	객체가 다른 객체를 사용함
의존성 (dependency)	..>	더 약한 형태의 의존성

PlantUML 코드 샘플: 관계

```
@startuml

package "Up To Bottom" {
  class1 <|-- class2
  class3 <|.. class4
  class5 *-- class6
  class7 o-- class8
  class9 +-- class10
}

package "Left to Right" {
  class11 -right-> class12
  class13 .right.> class14
}

@enduml
```

애트리뷰트, 오퍼레이션 가시화

Character	Icon for field	Icon for method	Visibility
-			private
#			protected
~			package private
+			public

애트리뷰트

- *cf.* https://www.uml-diagrams.org/property.html
- 정의 (informal)
  - property ::= [visibility] ['/'] property-name [':' property-type] ['[' multiplicity ']'] ['=' default-value] [property-modifiers]
  - visibility ::= '+' | '~' | '#' | '-'
  - property-modifiers ::= '{' property-modifier [',' property-modifier] * '}'
  - property-modifier ::= 'id' | 'readOnly' | 'ordered' | ('seq' | 'sequence') | 'unique' | 'nonunique' | 'union' | 'redefines' property-name | 'subsets' property-name | property-constraint
- Forward slash '/' means that the property is derived.

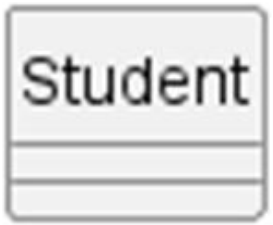
- 예제 → 대응하는 자바 코드
  - **+ name : String → public String name;**
  - **- n : int → private int n;**
  - **# animalType : Animal → protected Animal animalType;**

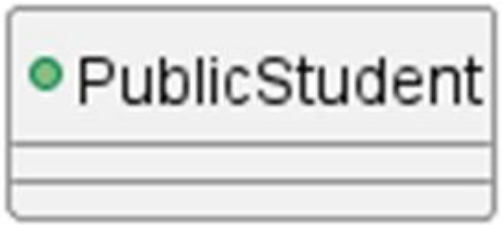
- 자바: 인스턴스 변수 또는 필드(field)
  - *cf.* https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html
- C++: 멤버 변수 (member variable)

오퍼레이션

- *cf.* http://www.uml-diagrams.org/operation.html
- 정의 (formal): ***operation*** ::= [***visibility***] ***signature*** [***oper-properties***]
  - ***visibility*** ::= '+' | '-' | '#' | '~'
  - ***signature*** ::= ***operation-name*** '(' [***parameter-list***] ')' [':' ***return-spec***]
  - ***parameter-list*** ::= ***parameter*** [',' ***parameter***]*
  - ***parameter*** ::= [***direction***] ***parm-name*** ':' ***type-expression*** ['[' ***multiplicity*** ']'] ['=' ***default***] [***parm-properties***]
  - ***direction*** ::= 'in' | 'out' | 'inout'
  - ***return-spec*** ::= [***return-type***] ['[' ***multiplicity*** ']']
- 예제 → 대응하는 코드
  - **+ getName() : String → public String getName() { return name; }**
  - **+ setN(in n: int) : void → public void setN(int n) { this.n = n; }**
  - **# getAnimalType() : Animal → protected Animal getAnimalType() { ? }**
- 자바: 메소드 (method) / C++: 멤버 함수 (member function)

실습 01: 클래스





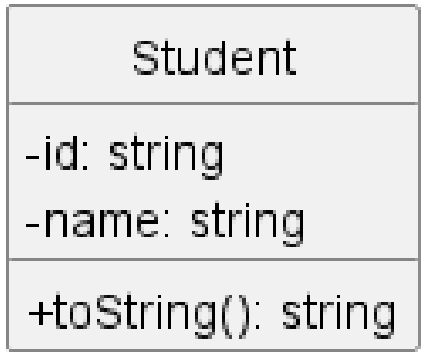

```
class Student {
}
```

```
public class PublicStudent {
}
```

PlantUML 코드: cd-01.txt

```
@startuml /'삭제 금지'/
' left to right direction
' skinparam classAttributeIconSize 0
' 표시된 구간에서만 답안 작성 가능
'--------------------------

class Student {
}


+class PublicStudent {
}

'--------------------------
hide circle  /' 삭제 금지'/
@enduml /'삭제 금지'/
```

실습 02: 애트리뷰트와 오퍼레이션



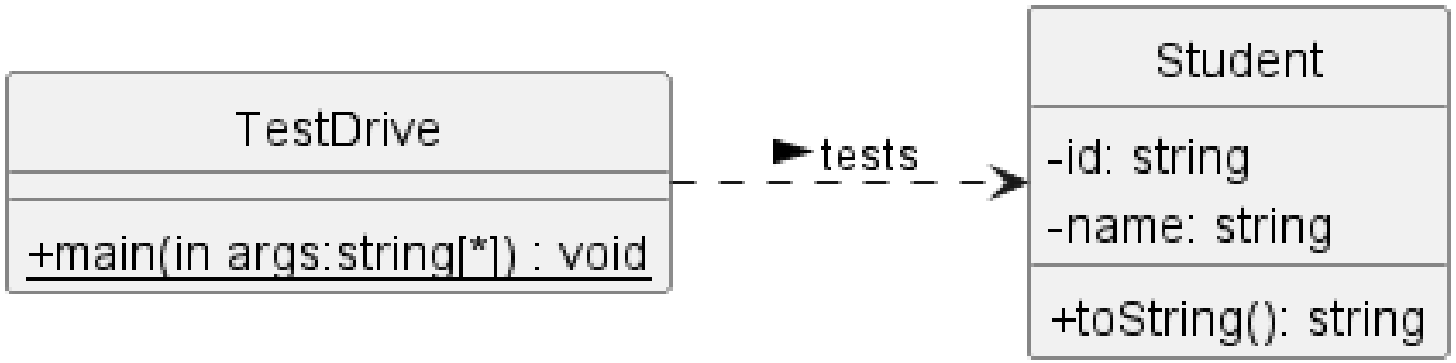

```
public class Student {
    private String id;
    private String name;

    public String toString() { … }
}
```

PlantUML 코드: cd-02.txt

```
@startuml /'삭제 금지'/
' left to right direction
skinparam classAttributeIconSize 0
' 표시된 구간에서만 답안 작성 가능
'---------------------------

class Student {
  -id: string        /' string 또는 String 모두 가능 '/
  -name: string
  +toString(): string
}


'---------------------------
hide circle  /' 삭제 금지'/
@enduml /'삭제 금지'/
```

실습03: 클래스 관계

- TestDrive 클래스에는 정적 메서드 main()이 있음.
- TestDrive 클래스의 main()에서 Student 인스턴스를 생성하여 실행하기 때문에 의존 관계를 사용하여 모델링함.
- 의존 관계의 "tests"는 설명이며, ▶는 의존 관계의 방향을 나타냄.

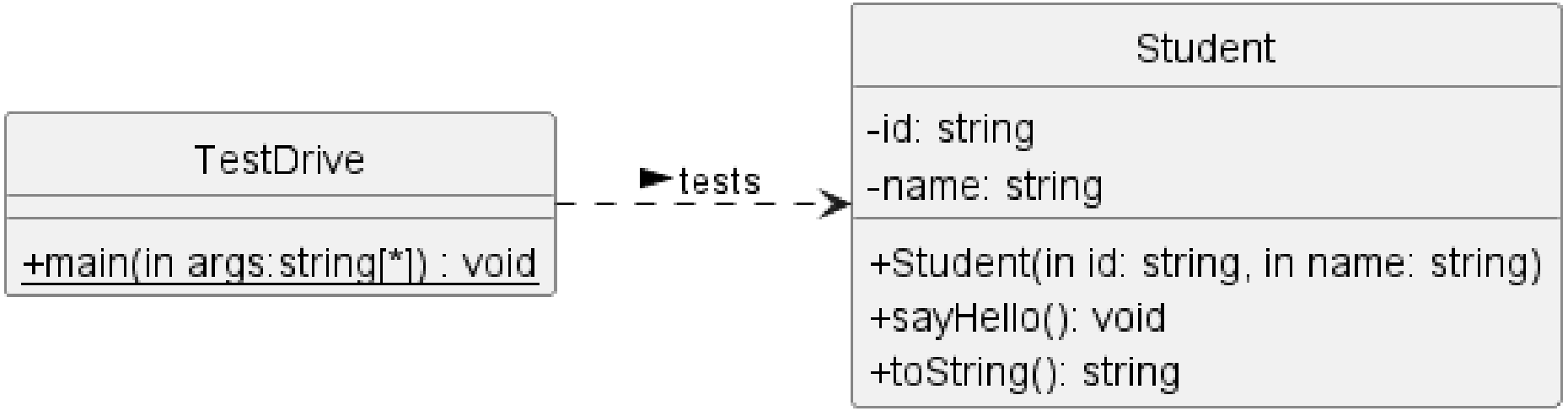


PlantUML 코드: cd-03.txt

```
@startuml /'삭제 금지'/
left to right direction
skinparam classAttributeIconSize 0
' 표시된 구간에서만 답안 작성 가능
'---------------------------

class Student {
  -id: string
  -name: string
  +toString(): string
}

class TestDrive {
  +main(in args: string[*]) : void {static}
}


TestDrive ..> Student : tests >

'---------------------------
hide circle  /' 삭제 금지'/
@enduml /'삭제 금지'/
```

실습04: 클래스 관계 (Java 실습)

- Java 실습을 위하여 실습03의 클래스 다이어그램에 Student 생성자와 sayHello() 메서드가 추가된 사례

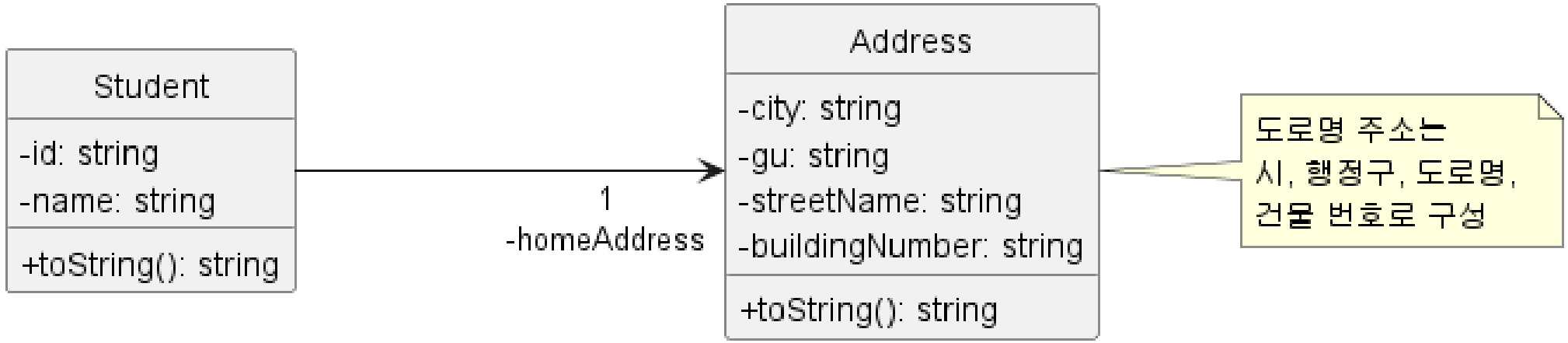


PlantUML 코드: cd-04.txt

```
@startuml /'삭제 금지'/
left to right direction
skinparam classAttributeIconSize 0
' 표시된 구간에서만 답안 작성 가능
'---------------------------

class Student {
  -id: string
  -name: string

  +Student(in id: string, in name: string)  /' 생성자 '/
  +sayHello(): void
  +toString(): string
}

class TestDrive {
  +main(in args: string[*]) : void {static}
}

'  ..> 기호는 의존성(depedency)을 나타냄.
TestDrive ..> Student : tests >

'---------------------------
hide circle  /' 삭제 금지'/
@enduml /'삭제 금지'/
```

실습05: 연관

- 연관의 클래스의 애트리뷰트로 구현됨.



- Java 코드 예

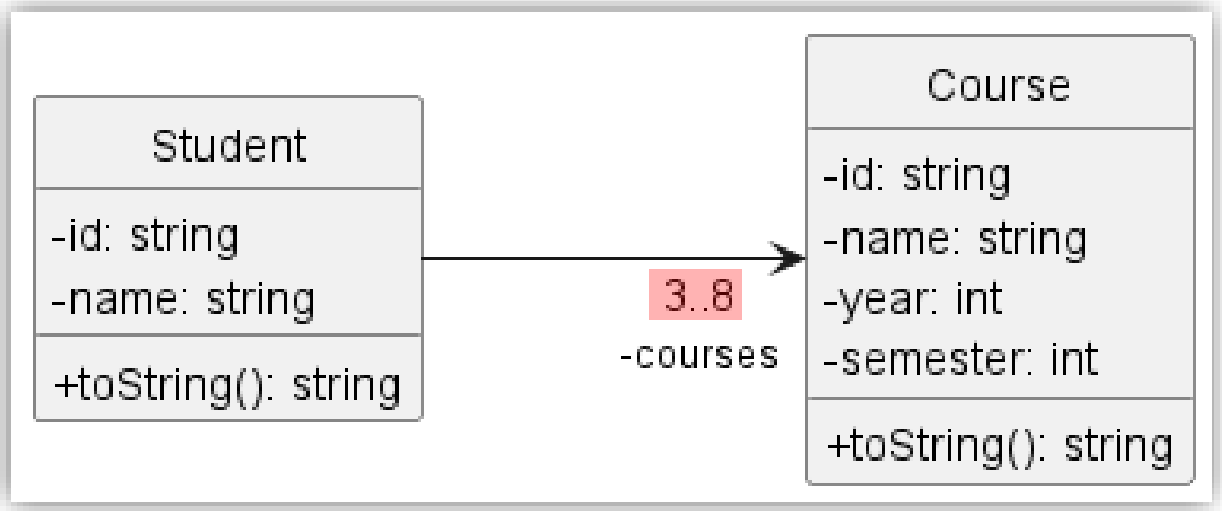
```
class Student {
  // 이전 생략
  private Address homeAddress;
  // 이하 생략
}
```

PlantUML 코드: cd-05.txt

```
@startuml /'삭제 금지'/
left to right direction
skinparam classAttributeIconSize 0
' 표시된 구간에서만 답안 작성 가능
'---------------------------

class Student {
  -id: string
  -name: string

  +toString(): string
}

class Address {
  -city: string
  -gu: string
  -streetName: string
  -buildingNumber: string

  +toString(): string
}

note right of Address
도로명 주소는
시, 행정구,
도로명, 건물 번호로 구성
end note


Student ----> "1₩n-homeAddress" Address


'---------------------------
hide circle  /' 삭제 금지'/
@enduml /'삭제 금지'/
```

실습06: 다중성 (Multiplicity)

- 연관에 의해 연결된 객체의 개수 또는 **애트리뷰트, 매개변수의 개수**를 의미
- 지정하지 않을 경우에는 일반적으로 하나를 의미
- 2개 이상일 경우 이를 적절히 다룰 수 있도록 배열이나 컬렉션 클래스를 사용해야 함.
- 다중성 예
  - * : 0개 이상
  - 1..* : 1개 이상
  - 10: 10개
  - min..max : min ~ max 개
  - 0,2..5,9..*: 없거나 2~5개, 9개 이상



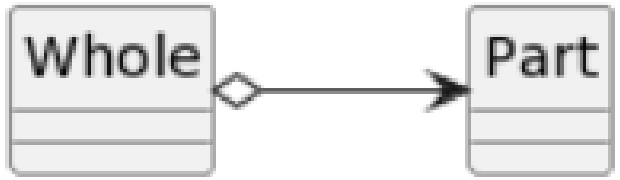
PlantUML 코드: cd-06.txt

```
@startuml /'삭제 금지'/
left to right direction
skinparam classAttributeIconSize 0
' 표시된 구간에서만 답안 작성 가능
'--------------------------

class Student {
  -id: string
  -name: string
  '-----------------
  +toString(): string
}

class Course {
  -id: string
  -name: string
  -year: int
  -semester: int
  '-----------------
  +toString(): string
}

Student ---> "3..8₩n-courses" Course

'--------------------------
hide circle  /' 삭제 금지'/
@enduml /'삭제 금지'/
```

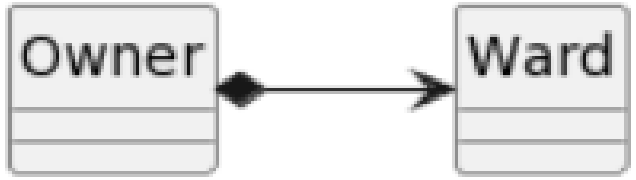
실습07: 집합과 합성

- 연관의 특수한 형태로 독점적 소유권(exclusive ownership) 유무에 따라서 구분

집합



합성

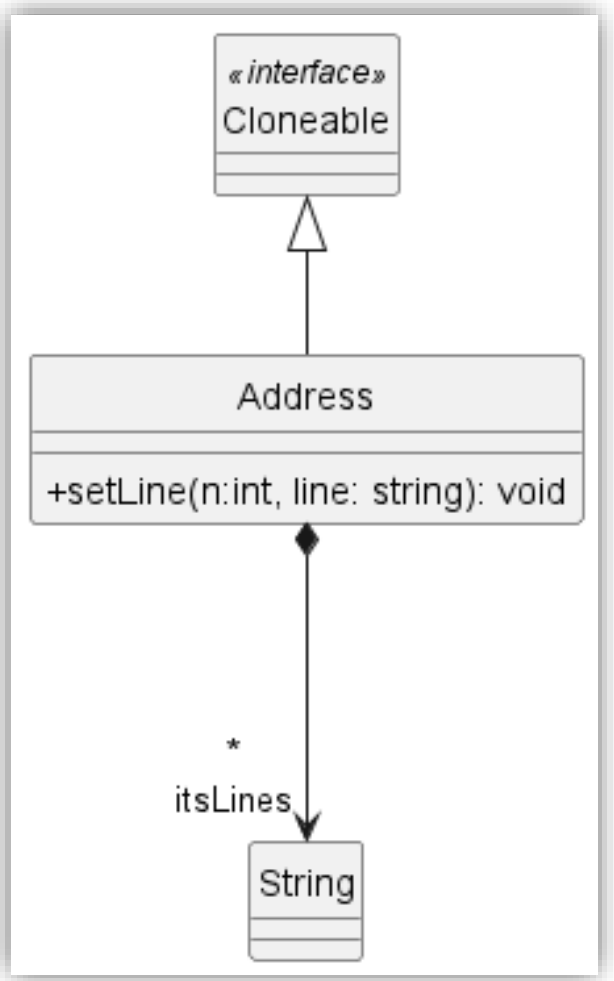


PlantUML 코드: cd-07.txt

```
@startuml /'삭제 금지'/
left to right direction
' 표시된 구간에서만 답안 작성 가능
'--------------------------
class Whole
class Part

Whole o--> Part  /' 집합 '/


class Owner
class Ward

Owner *--> Ward  / '  합성 '/


'--------------------------
hide circle  /' 삭제 금지'/
@enduml /'삭제 금지'/
```

실습08: 합성-Deep Copy

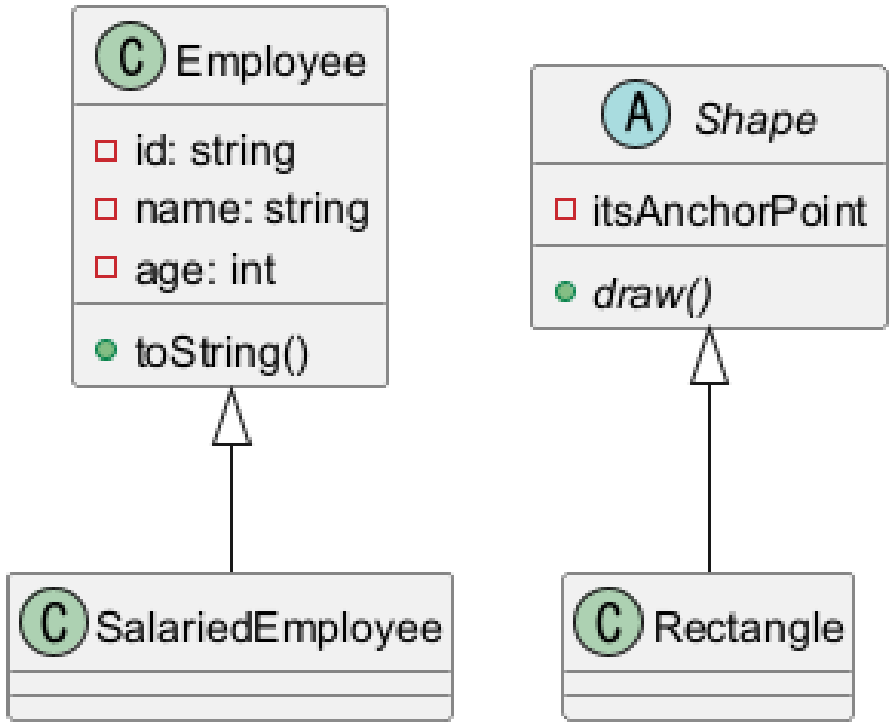
- Address 객체 복사할 때 Vector로 구현된 itsLines 인스턴스 변수도 같이 복사하여 생성해야 함.



PlantUML 코드: cd-08.txt

```
@startuml /'삭제 금지'/
'left to right direction
skinparam classAttributeIconSize 0
' 표시된 구간에서만 답안 작성 가능
'---------------------------

class Cloneable <<interface>>  /' 또는 interface Cloneable '/
class Address {
  +setLine(n:int, line: string): void
}
class String

Cloneable <|-- Address
Address *---> "*₩nitsLines" String



'---------------------------
hide circle  /' 삭제 금지'/
@enduml /'삭제 금지'/
```

실습09: 상속과 인터페이스

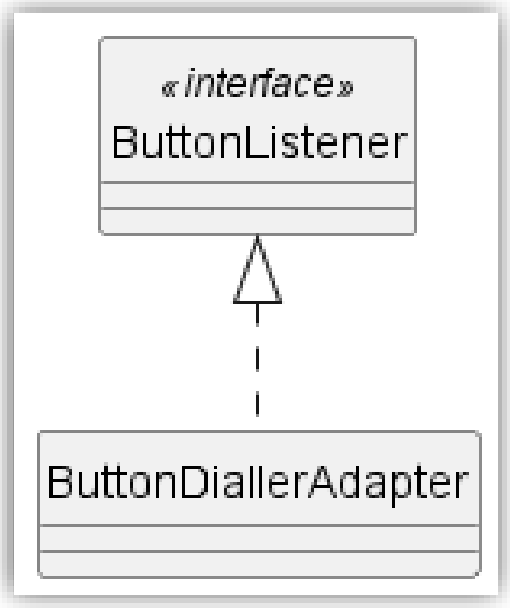


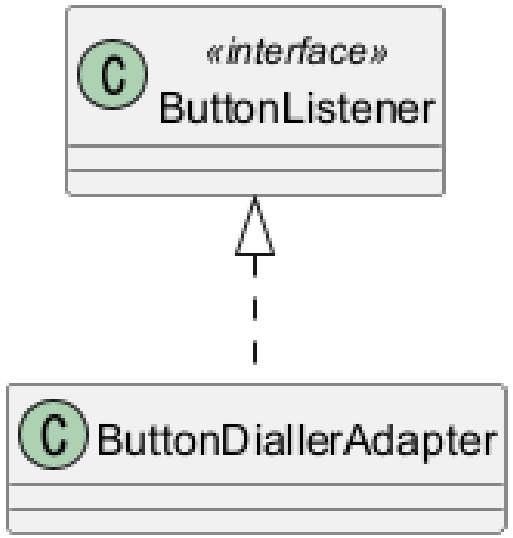
PlantUML 코드: cd-09.txt

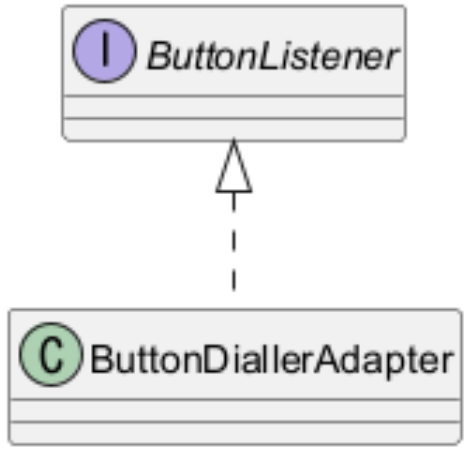
```
@startuml

class Employee {
  -id: string
  -name: string
  -age: int
  +toString()
}
' class SalariedEmployee

Employee <|-- SalariedEmployee

abstract class Shape {
  -itsAnchorPoint: Point2D[2..*]
  +draw() {abstract}
}
class Rectangle {
  +draw()
}

Shape <|-- Rectangle


@enduml
```

실습10: 실체화 의존성 또는 구현 관계

- 인터페이스는 interface 또는 class … <<interface>> 형식으로 사용 가능





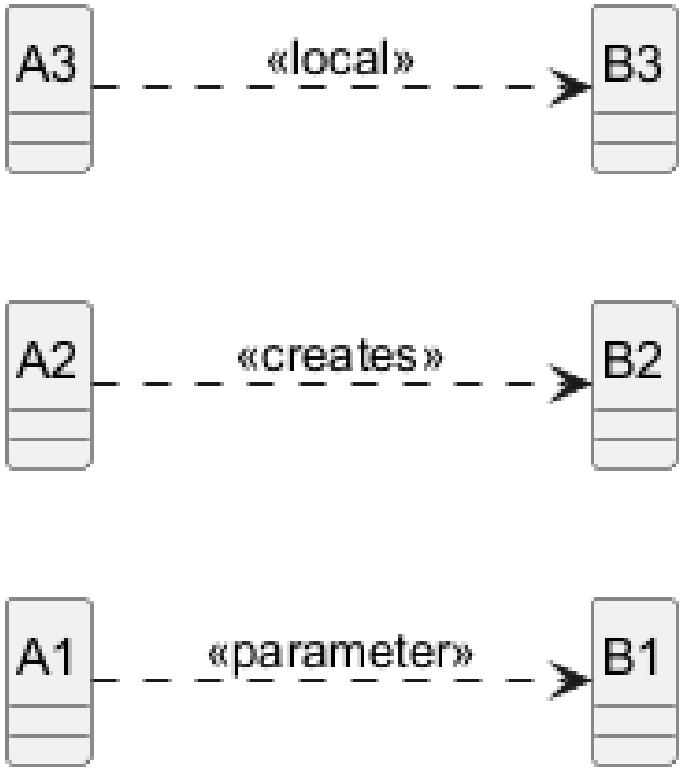


PlantUML 코드: cd-10.txt

```
@startuml /'삭제 금지'/
'left to right direction
skinparam classAttributeIconSize 0
'
' 표시된 구간에서만 답안 작성 가능
'--------------------------


‘ interface ButtonListener
class ButtonListener <<interface>>

ButtonListener <|.. ButtonDiallerAdapter

'--------------------------
‘ hide circle  /' 삭제 금지'/
@enduml /'삭제 금지'/
```

실습11: 의존 관계

- 매개변수 의존성 : ..>~ : <<parameter>>
- 생성 의존성 : ..> ~ : <<creates>>
- 지역 의존성 : ..> ~ : <<local>>



PlantUML 코드: cd-11.txt

```
@startuml /'삭제 금지'/

class A1

' 의존 관계 유형을 명시적으로 표현할 때만 사용
A1 ..> B1 : <<parameter>>
A2 ..> B2 : <<creates>>
A3 ..> B3 : <<local>>


@enduml /'삭제 금지'/
```

의존성 코드 예 (1/2)

[매개변수 의존성]

```
class A1 {
  public op(B1 b) {
  // 중략
  }
}

class B1 { /* 생략 */ }
```

[생성 의존성]

```
class A2 {
  public B2 makeB2() {
    return new B2();
  }
}

class B2 { /* 생략 */ }
```

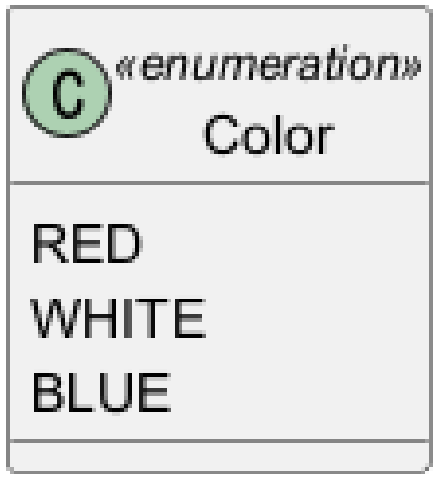
의존성 코드 예 (2/2)

[지역 의존성]

```
class A3 {
  public void foo() {
    B3 b = new B();
    // use b
  }
}

class B3 { /* 생략 */ }
```

실습12: 열거형

- 기호 값(symbolic value) 이외에는 어떠한 속성도 갖지 않는 고정된 값의 집합을 보이기 위하여 사용
- UML에서는 ≪enumeration≫ 키워드를 사용하여 표시




```
public enum Color {
  RED, WHITE, BULE;

   public String toString() {
      switch(this) {
         case RED:
            return “red”;
         case WHITE:
            return “white”;
         case BLUE:
            return “blue”;
         default:
            return “no color”;
   }
}
```

```
public enum Color {
  RED(“red”),
  WHITE(“white”),
  BLUE(“blue”);

  private String value;

  private Color(String value) {
    this.value = value;
  }
  public String toString() {
    return value;
  }
}
```

PlantUML 코드: cd-12.txt

```
@startuml /'삭제 금지'/

class Color <<enumeration>> {
  RED
  WHITE
  BLUE
}


enum MyColor {
  RED,
  WHITE,
  BLUE;
}

@enduml /'삭제 금지'/
```

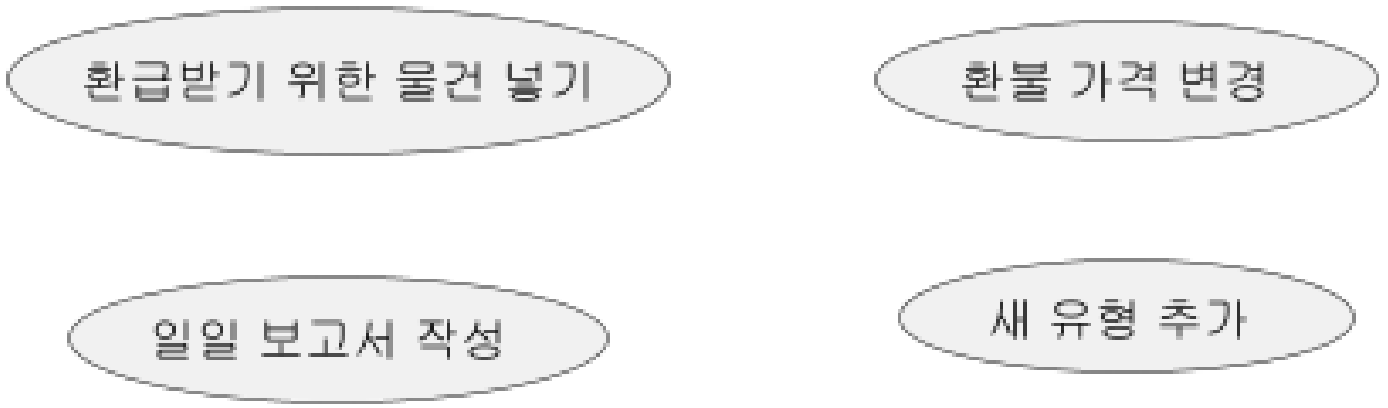
```
# python
from enum import Enum

class Color(Enum):
  RED = “red”
  WHITE = “white”
  BLUE = “blue”

c = Color.RED
print(c, c.name, c.value)
```

기타 다이어그램

유스 케이스 다이어그램

- 시스템과 사용자(액터) 간의 상호작용을 시각적으로 표현하는 도구
- **유스케이스 다이어그램의 주요 구성 요소:**
  - **액터(Actor):** 시스템과 상호작용하는 외부 요소(사용자, 다른 시스템 등). 시스템의 기능을 사용하는 역할을 수행하며, 시스템 외부에 존재
  - **유스케이스(Use Case):** 시스템이 제공하는 기능 또는 서비스. 액터가 시스템을 통해 수행하는 특정 목표를 의미하며, 타원 모양으로 표현
  - **관계(Relationship):** 액터와 유스케이스, 유스케이스 간의 관계
    - **연관 관계(Association):** 액터와 유스케이스 간의 상호작용 표현
    - **일반화 관계(Generalization):** 액터 또는 유스케이스 간의 일반화/특수화 관계 표현
    - **포함 관계(Include):** 하나의 유스케이스가 다른 유스케이스의 기능을 포함하는 경우 사용
    - **확장 관계(Extend):** 하나의 유스케이스가 다른 유스케이스의 기능을 확장하는 경우 사용

실습01: 유스케이스

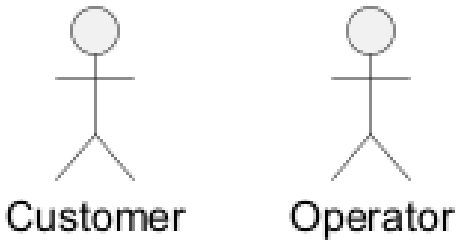


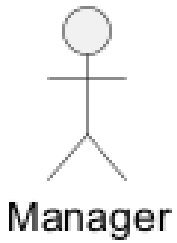
PlantUML 코드: uc-01.txt

```
@startuml
left to right direction

(환급받기 위한 물건 넣기) as UC1
(일일 보고서 작성) as UC2
usecase "환불 가격 변경" as UC3
usecase "새 유형 추가" as UC4

@enduml
```

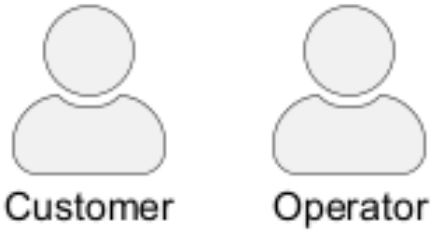
실습02: 액터

Stickman (기본)

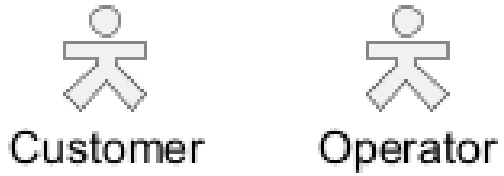




awesome



Hollow

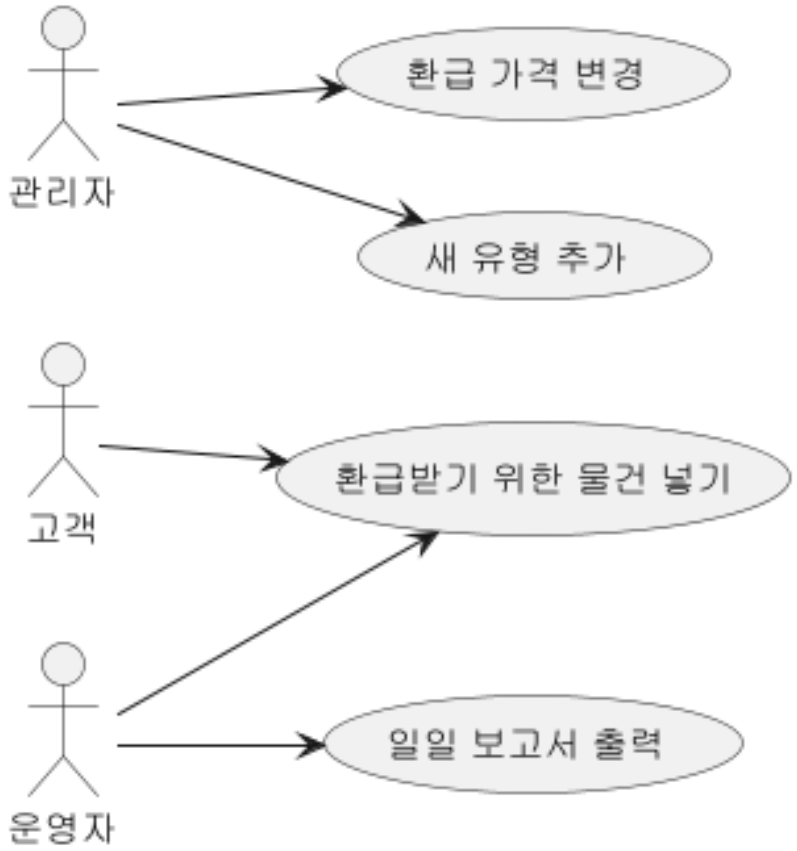


PlantUML 코드: uc-02.txt

```
@startuml
' skinparam actorStyle awesome
' skinparam actorStyle Hollow

:Customer: as A1
actor Operator as A2
actor Manager as A3

@enduml
```

실습03: 액터와 유스케이스 관계

- 액터와 유스케이스 관계는 연관(association)으로 모델링

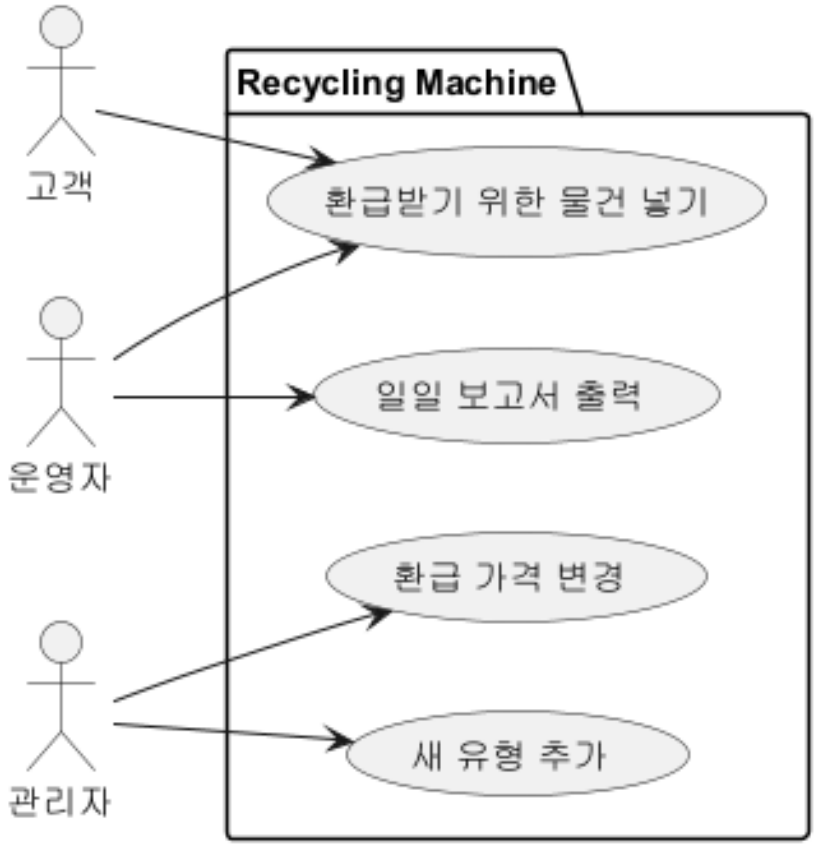


PlantUML 코드: uc-03.txt

```
@startuml
left to right direction


actor 고객 as A1
:운영자: as A2
:관리자: as A3

usecase "환급받기 위한 물건 넣기" as UC1
usecase "일일 보고서 출력" as UC2
usecase "환급 가격 변경" as UC3
(새 유형 추가) as UC4

A1 --> UC1
A2 --> UC1
A2 --> UC2
A3 --> UC3
A3 --> UC4

@enduml
```

실습04: 시스템 범위

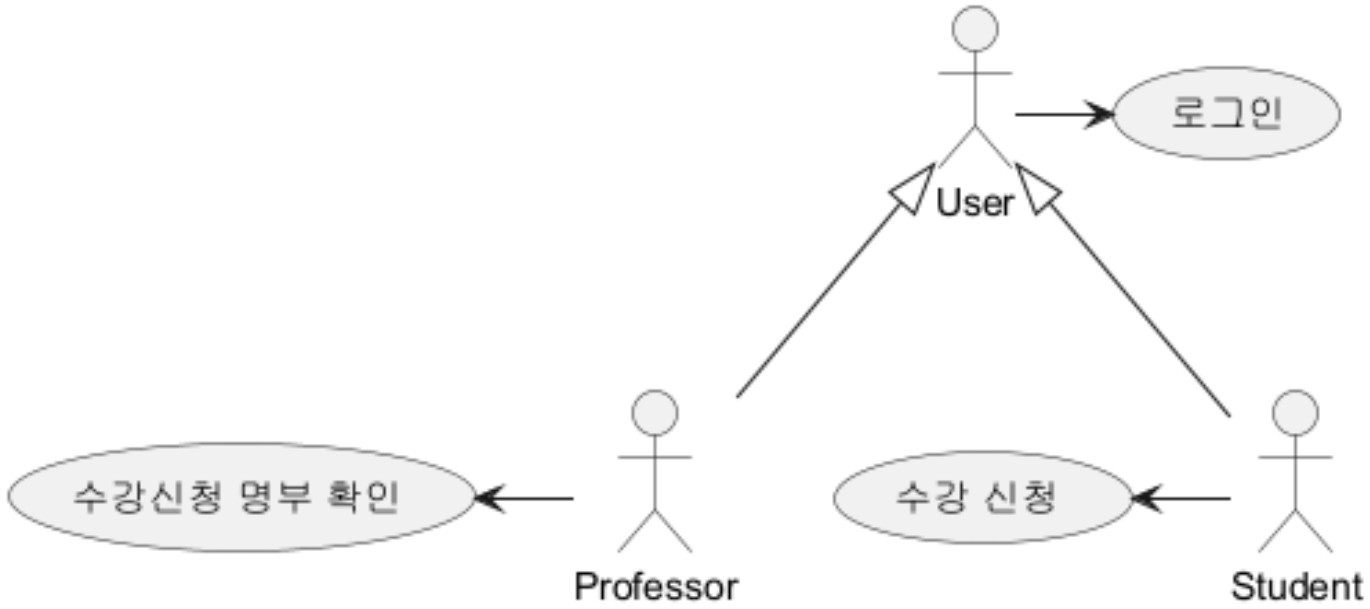
- 개발 범위에 해당하며 boundary box 형태로 표현



PlantUML 코드: uc-04.txt

```
@startuml
left to right direction


actor 고객 as A1
:운영자: as A2
:관리자: as A3

package "Recycling Machine" {
    usecase "환급받기 위한 물건 넣기" as UC1
    usecase "일일 보고서 출력" as UC2
    usecase "환급 가격 변경" as UC3
    (새 유형 추가) as UC4
}

A1 --> UC1
A2 --> UC1
A2 --> UC2
A3 --> UC3
A3 --> UC4

@enduml
```

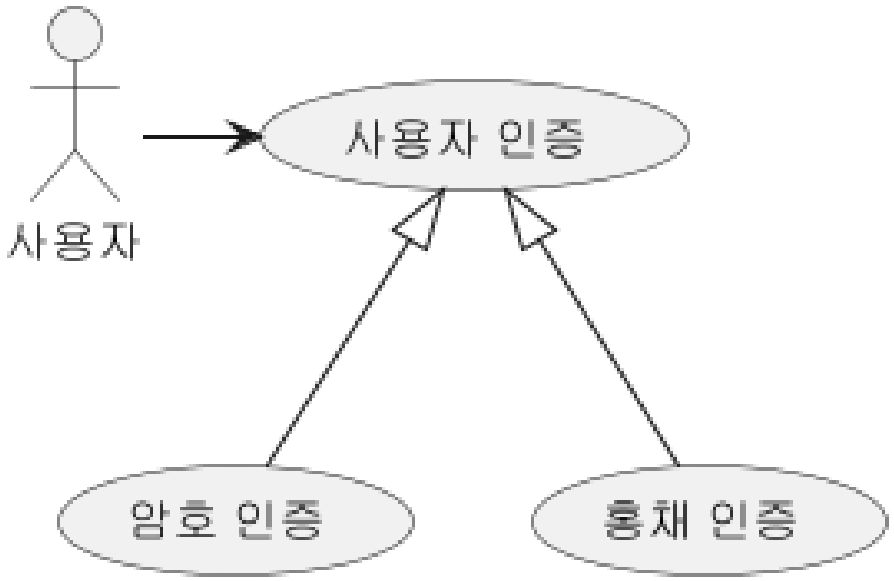
실습05: 액터 일반화



PlantUML 코드: uc-05.txt

```
@startuml
' left to right direction
top to bottom direction

actor User as A1
actor Student as A2
actor Professor as A3

A1 <|-- A2
A1 <|-- A3

(로그인) as UC1
(수강 신청) as UC2
(수강신청 명부 확인) as UC3

A1 -right-> UC1
A2 -left--> UC2
A3 -l--> UC3


@enduml
```

실습06: 유스케이스 일반화

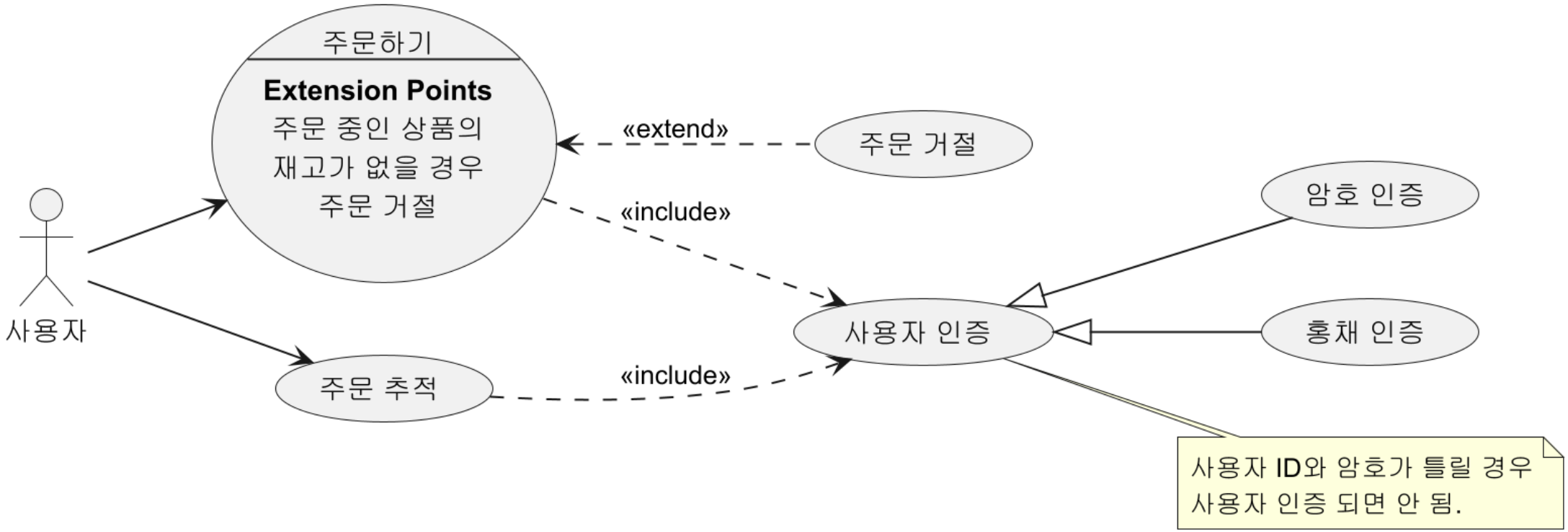


PlantUML 코드: uc-06.txt

```
@startuml

actor 사용자 as A1

(사용자 인증) as UC1
(암호 인증) as UC2
(홍채 인증) as UC3

UC1 <|-- UC2
UC1 <|-- UC3

/' --> 대신 ->으로 헤도 화살표 방향이 좌우로 됨. '/
A1 -right-> UC1

@enduml
```

실습07: 포함, 확장 관계

- 포함 관계: **주문 추적** UC 실행할 때 반드시 **사용자 인증** UC 사용함을 의미
- 확장 관계: **주문하기** UC 실행할 때 **주문 거절** UC는 실행될 수도 있고 아닐 수도 있음.

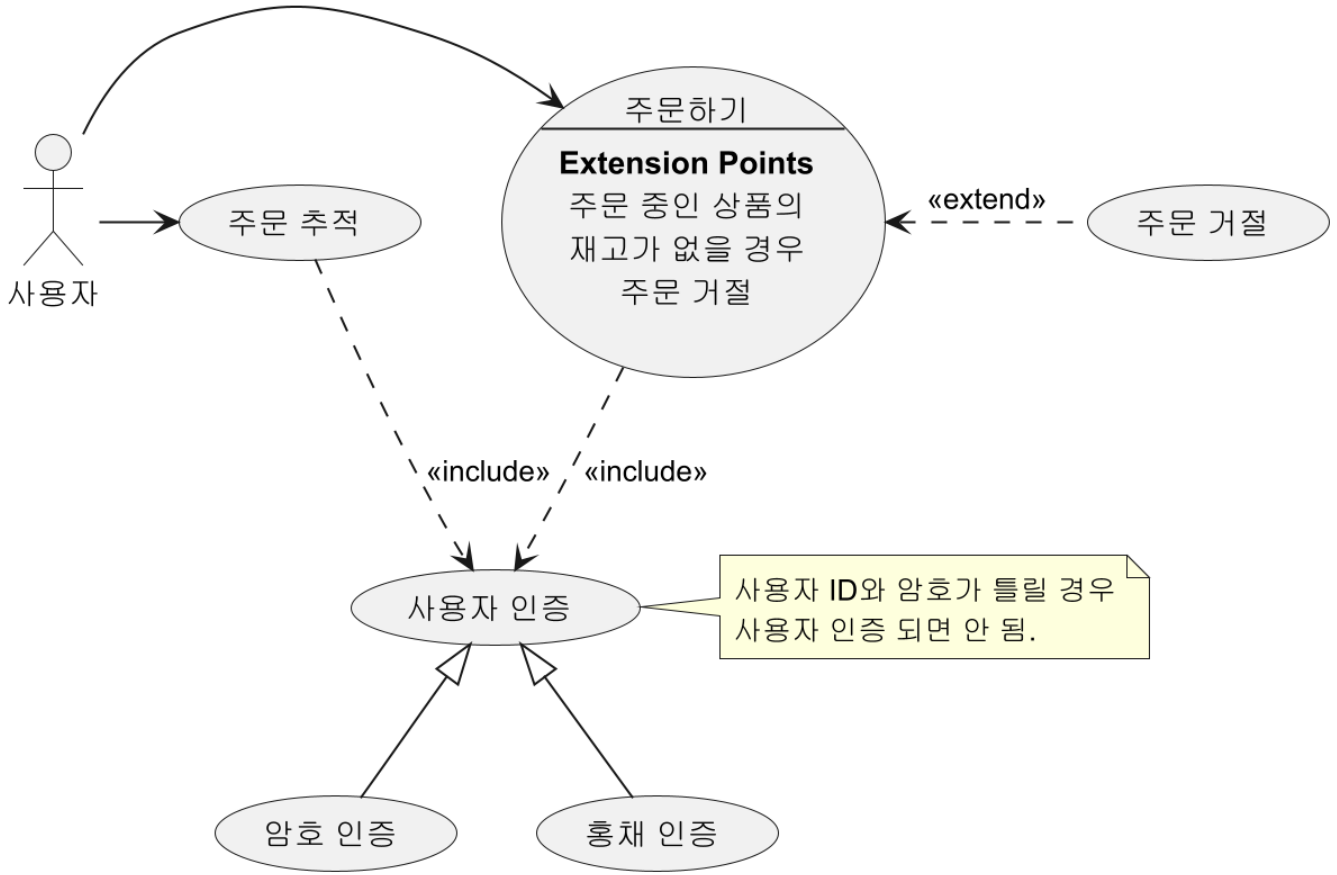


PlantUML 코드: uc-07.txt

```
@startuml

left to right direction

actor 사용자 as A1

' (사용자 인증) as UC1
UC1 as (사용자 인증)
(암호 인증) as UC2
(홍채 인증) as UC3

UC1 <|-down- UC2
UC1 <|-d- UC3

usecase UC4 as "주문하기
--
**Extension Points**
주문 중인 상품의
재고가 없을 경우
주문 거절"

(주문 거절) as UC5
(주문 추적) as UC6

A1 --> UC4
A1 --> UC6
```

```
UC4 ..> UC1 : <<include>>
UC6 ..> UC1 : <<include>>
UC4 <.. UC5 : <<extend>>

note as N1
사용자 ID와 암호가 틀릴 경우
사용자 인증 되면 안 됨.
end note

UC1 -- N1

@enduml
```

실습07-1: 포함, 확장 관계 (default direction)



PlantUML 코드: uc-07-1.txt

```
@startuml

actor 사용자 as A1

' (사용자 인증) as UC1
UC1 as (사용자 인증)
(암호 인증) as UC2
(홍채 인증) as UC3

UC1 <|-- UC2
UC1 <|-- UC3

usecase UC4 as "주문하기
--
**Extension Points**
주문 중인 상품의
재고가 없을 경우
주문 거절"

(주문 거절) as UC5
(주문 추적) as UC6

A1 -r-> UC4
A1 -r-> UC6
```

```
UC4 ..> UC1 : <<include>>
UC6 ..> UC1 : <<include>>
UC4 <.r. UC5 : <<extend>>

note as N1
사용자 ID와 암호가 틀릴 경우
사용자 인증 되면 안 됨.
end note

UC1 .r. N1

@enduml
```

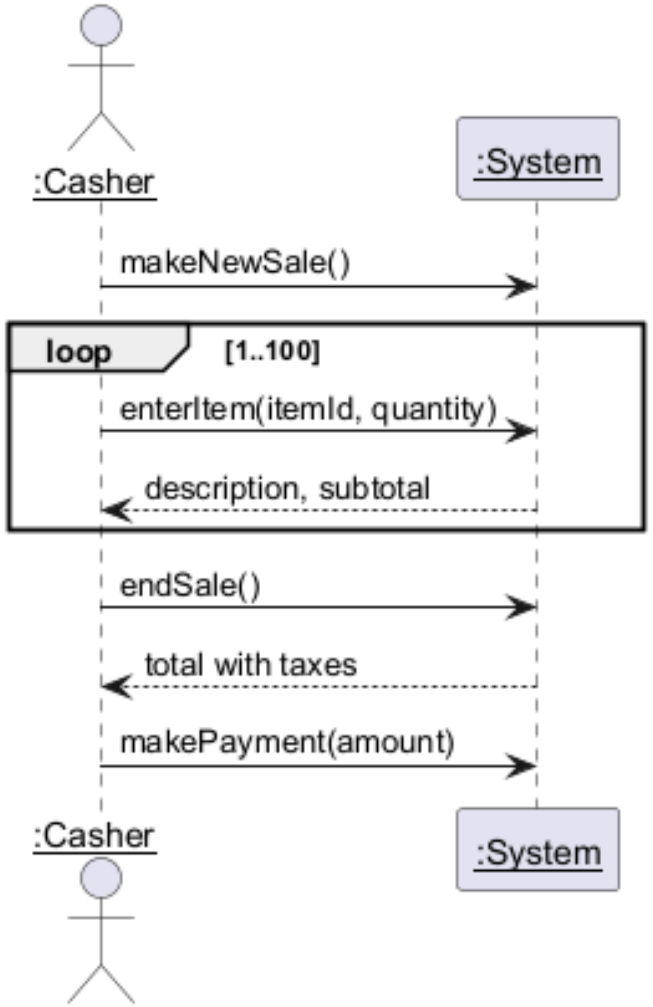
순차 다이어그램

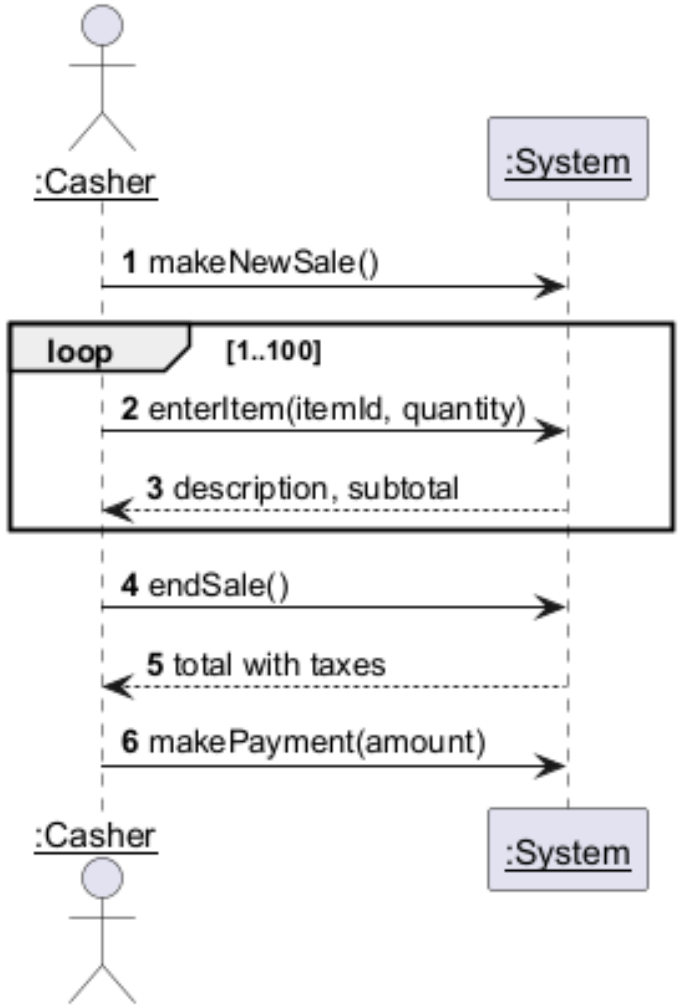
순차 다이어그램

- 객체간의 상호 작용을 시간 순서로 나열하여 보여주는 다이어그램
- 객체 명명 방법
  - 객체 이름: **SW공학**
  - 객체 이름과 클래스: **SW공학:교과목**
  - 클래스 이름: **:교과목** (익명 객체는 그 클래스의 모든 인스턴스를 의미)

- 활용 방안
  - 시스템 순차 다이어그램 (System Sequence Diagram)
    - 액터로부터 시스템(**:System**)에게 전달되는 이벤트를 보여줌
    - 분석 단계에서 시스템 오퍼레이션을 파악하여 설계 단계의 통신 다이어그램 그릴 때 시작점으로 사용
  - 객체 간 메시지 흐름 분석 (완벽한 오퍼레이션 형태를 사용하지 않아도 됨)
  - 설계 단계에서 구현에 필요한 클래스의 메서드 찾기 (클래스 유형: 경계, 제어, 엔티티 클래스 등)

실습01: 시스템 순차 다이어그램 1



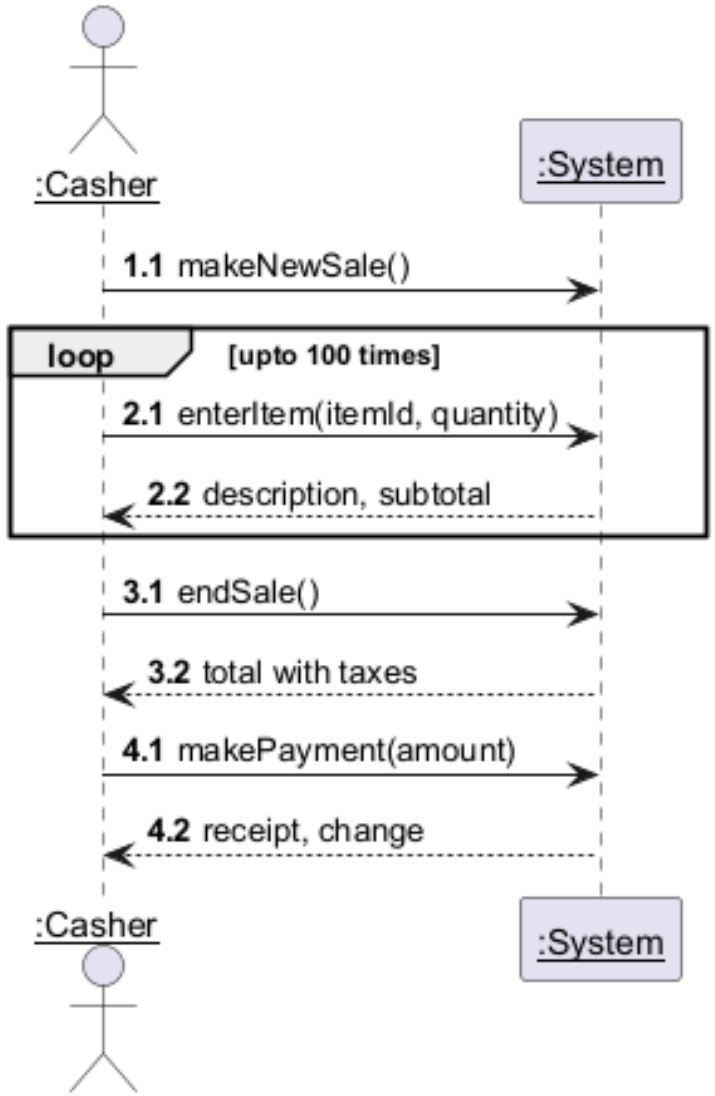


PlantUML 코드: sd-01.txt

```
@startuml

actor "<u>:Casher" as A1
participant "<u>:System" as S

' autonumber 1
A1 -> S : makeNewSale()

loop 1..100
    A1 -> S : enterItem(itemId, quantity)  /' request '/
    A1 <-- S : description, subtotal       /' response '/
end

A1 -> S : endSale()
' A1 <-- S : total with taxes
return total with taxes

A1 -> S : makePayment(amount)

@enduml
```

실습02: 시스템 순차 다이어그램 2

- 오퍼레이션 소번호 생성



PlantUML 코드: sd-02.txt

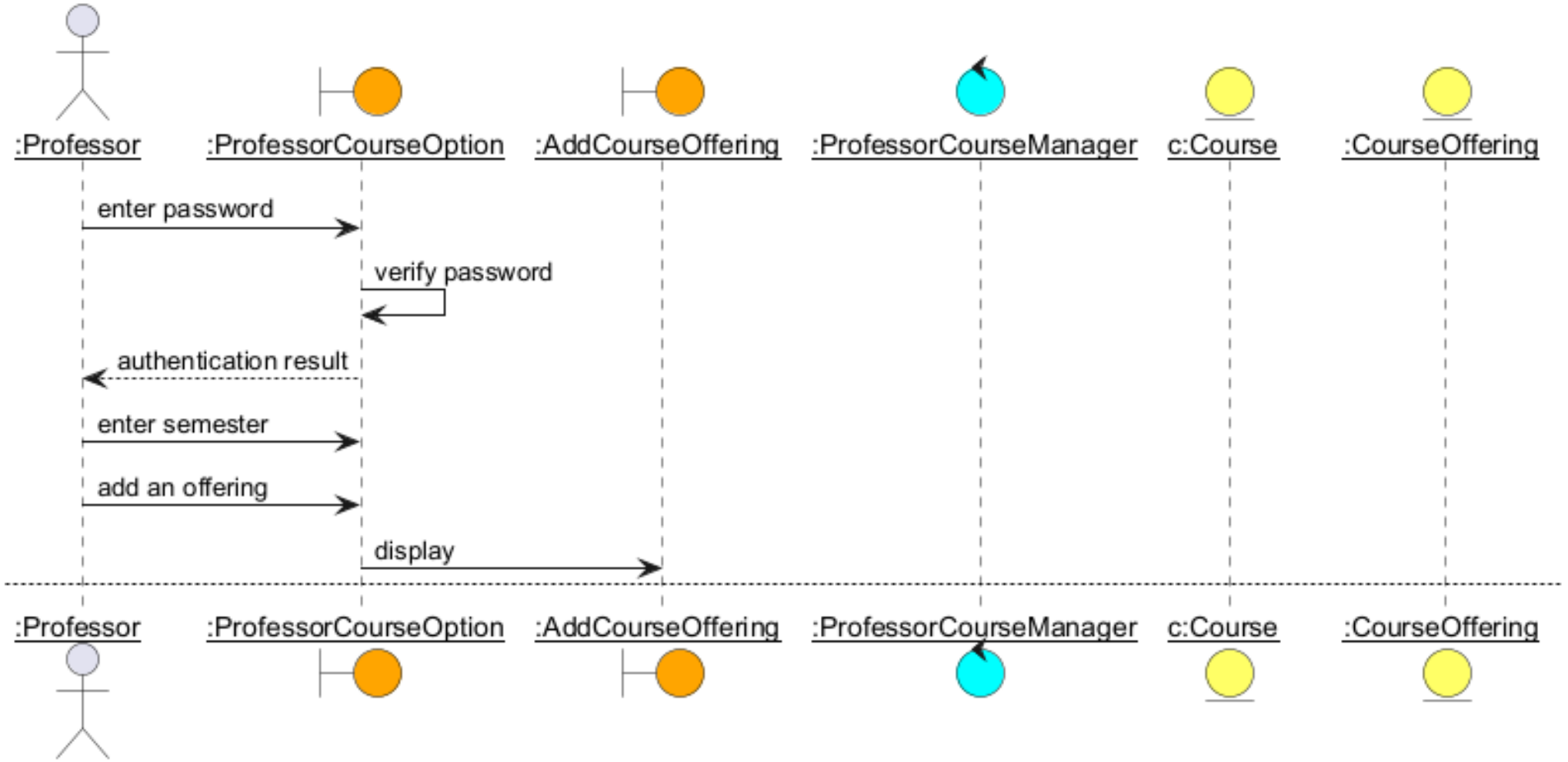
```
@startuml

actor "<u>:Casher" as A1
participant "<u>:System" as S

autonumber 1.1  /' 1.1 or 1.1.1 '/
A1 -> S : makeNewSale()

autonumber inc A
loop upto 100 times
    A1 -> S : enterItem(itemId, quantity)
    return description, subtotal
end

autonumber inc A
A1 -> S : endSale()
return total with taxes

autonumber inc A
A1 -> S : makePayment(amount)
return receipt, change

@enduml
```

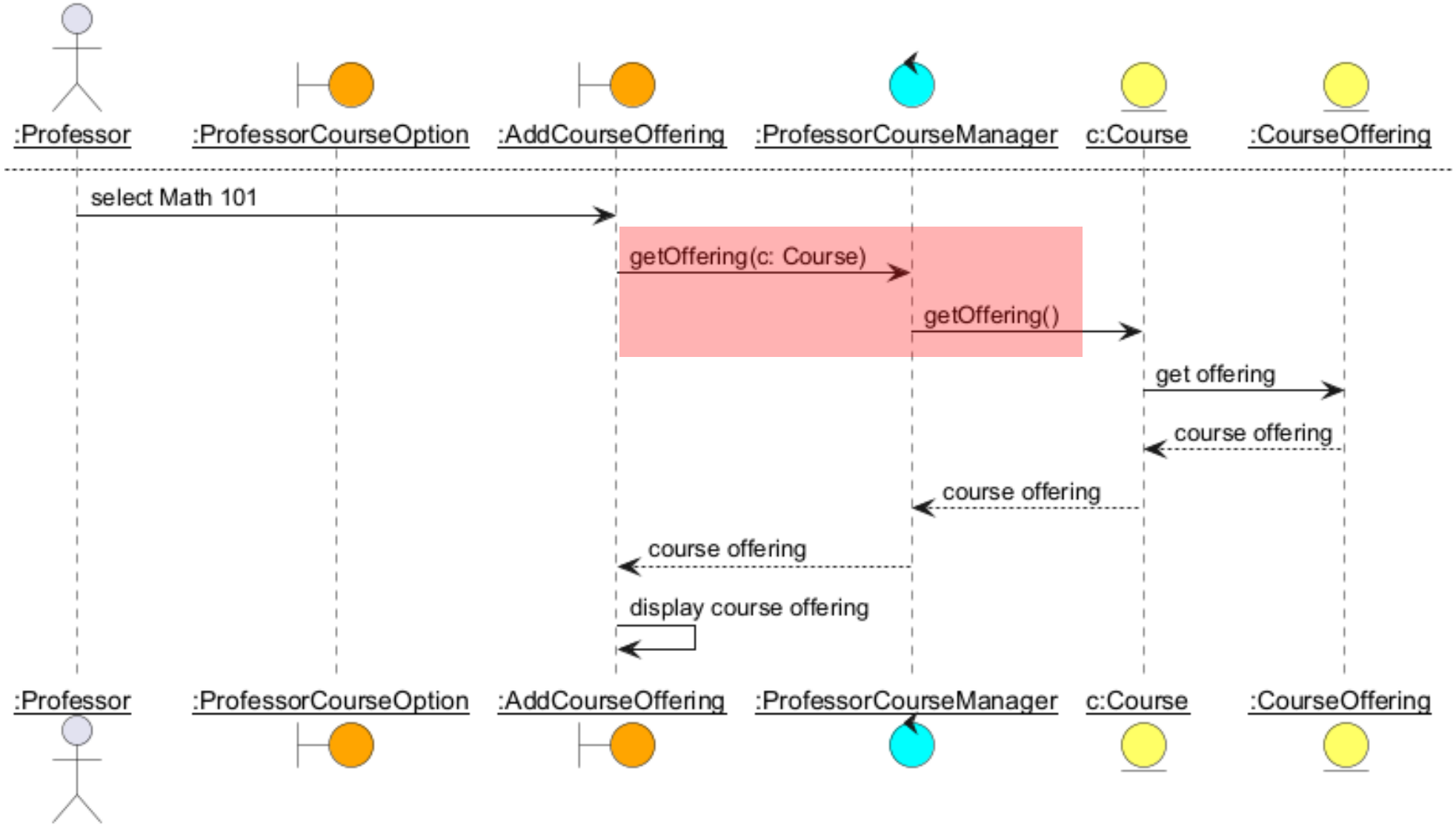
실습03: 경계, 제어, 엔티티 클래스

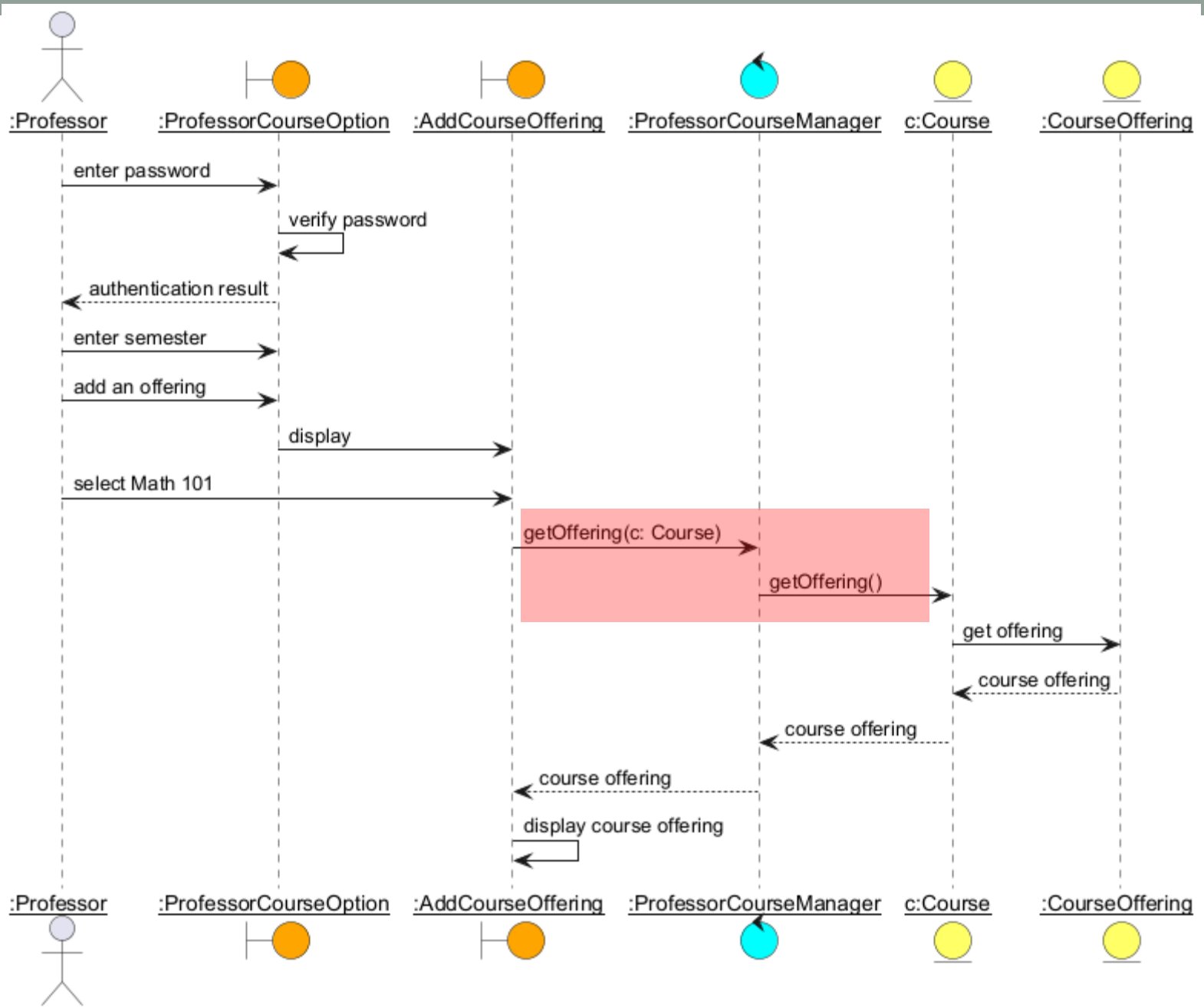
- **Boundary 클래스 (경계 클래스)**
  - **역할**
    - 시스템과 외부 액터 간 상호작용 담당
    - 사용자 인터페이스, 입력/출력 처리
  - **특징**
    - 외부 세계와의 경계 표현
    - 사용자 인터페이스 로직 포함
  - **예시**: 로그인 화면, 사용자 정보 입력 폼, 결과 보고서 화면
- **Control 클래스 (제어 클래스)**
  - **역할**
    - boundary, entity 클래스 간 상호작용 관리
    - 시스템 핵심 비즈니스 로직 수행
    - 시스템 흐름 제어
  - **특징**:
    - 핵심 로직 구현
    - boundary 클래스 요청 처리, entity 클래스 조작
    - 여러 클래스 간 협력 관리
  - **예시**: 주문 처리 로직, 회원 가입 로직, 데이터 검색 로직

- **Entity 클래스 (엔티티 클래스)**
  - **역할**
    - 시스템 핵심 데이터 표현
    - 영구적 데이터 저장 및 관리
  - **특징**
    - 핵심 정보 표현
    - 데이터 저장/검색 기능 제공
    - 비즈니스 규칙에 따른 데이터 관리
  - **예시**: 사용자 정보, 상품 정보, 주문 정보

- 순차 다이어그램은 주고 받는 메시지가 많으면 한 화면에 그리기 어려우므로 적절하게 화면을 분할하여 사용해야 함. PlantUML에서는 newpage를 입력하여 화면 분할할 수 있음.


:Professor
:ProfessorCourseOption
:AddCourseOffering
:ProfessorCourseManager
c:Course
:CourseOffering
enter password
verify password
authentication result
enter semester
add an offering
display
:Professor
:ProfessorCourseOption
:AddCourseOffering
:ProfessorCourseManager
c:Course
:CourseOffering

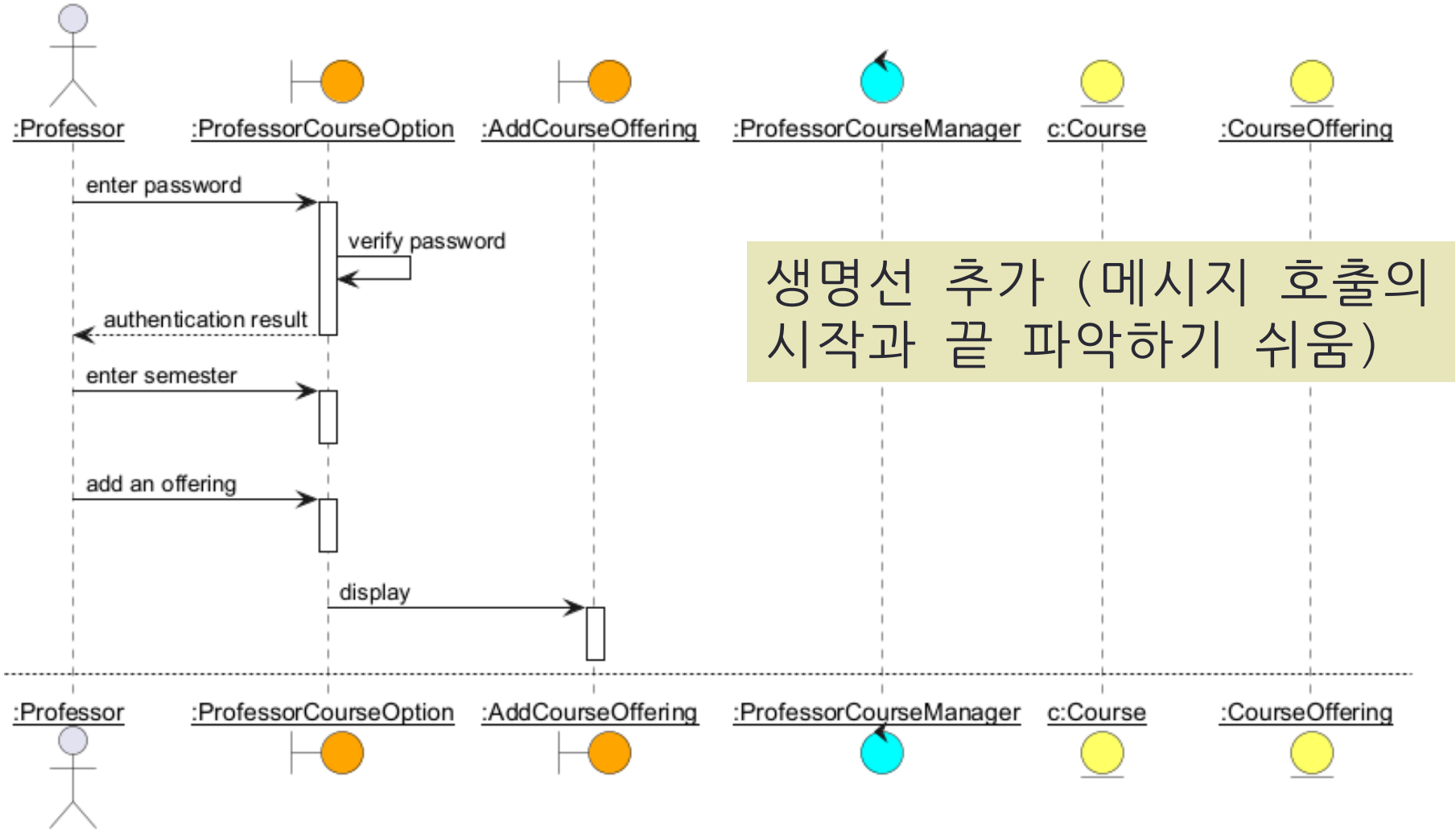

:Professor
:ProfessorCourseOption
:AddCourseOffering
:ProfessorCourseManager
c:Course
:CourseOffering
select Math 101
getOffering(c: Course)
getOffering()
get offering
course offering
course offering
course offering
display course offering
:Professor
:ProfessorCourseOption
:AddCourseOffering
:ProfessorCourseManager
c:Course
:CourseOffering

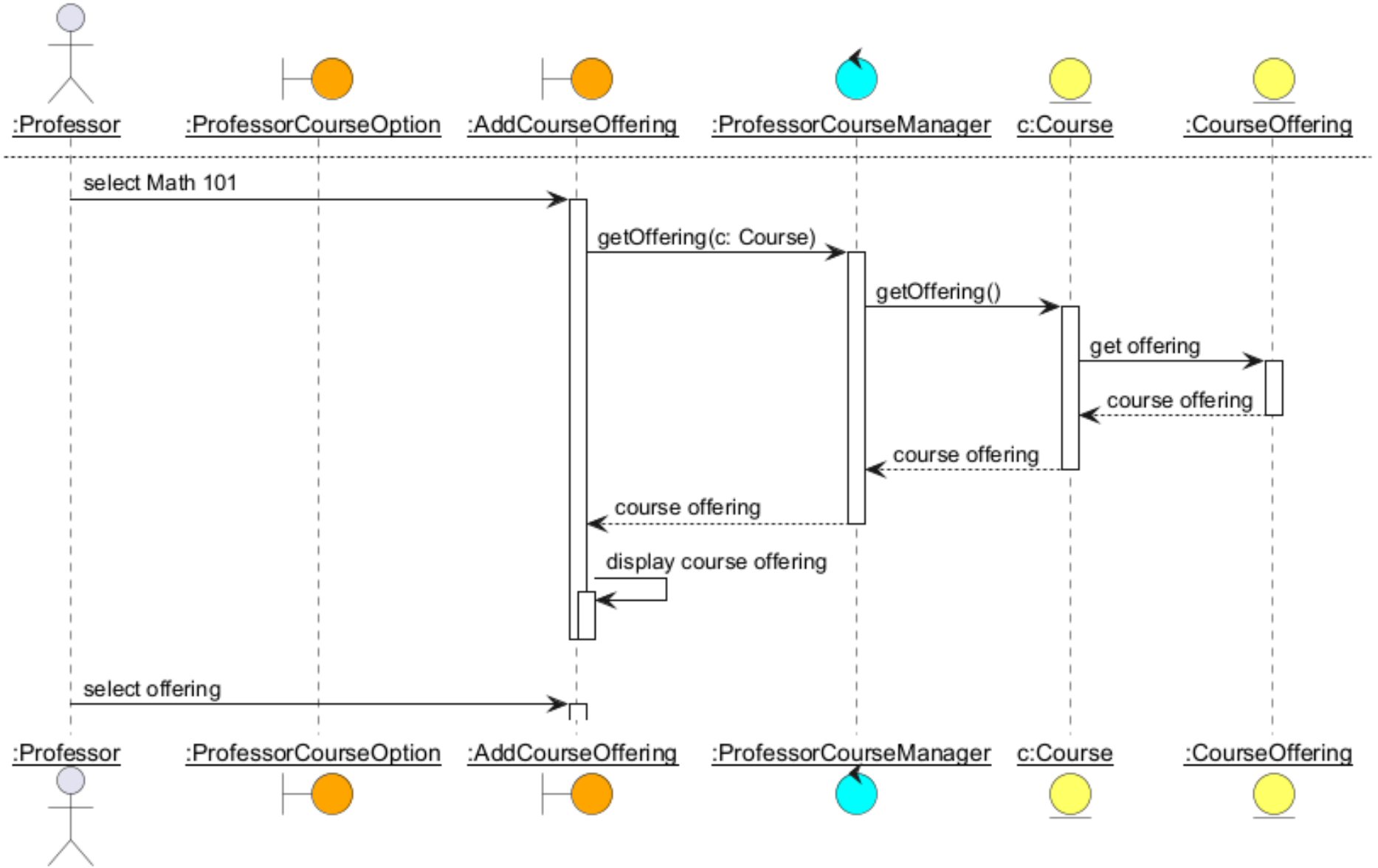

:Professor
:ProfessorCourseOption
:AddCourseOffering
:ProfessorCourseManager
c:Course
:CourseOffering
enter password
verify password
authentication result
enter semester
add an offering
display
select Math 101
getOffering(c: Course)
getOffering()
get offering
course offering
course offering
course offering
display course offering
:Professor
:ProfessorCourseOption
:AddCourseOffering
:ProfessorCourseManager
c:Course
:CourseOffering

PlantUML 코드: sd-03.txt

```
@startuml

actor "<u>:Professor" as A1
boundary "<u>:ProfessorCourseOption" as B1  #orange
boundary "<u>:AddCourseOffering" as B2 #orange

control "<u>:ProfessorCourseManager" as C1  #cyan
entity "<u>c:Course" as E1  #FFFF66
entity "<u>:CourseOffering" as E2  #FFFF66
A1 -> B1 : enter password
B1 -> B1 : verify password
A1 <-- B1 : authentication result

A1 -> B1 : enter semester
A1 -> B1 : add an offering
B1 -> B2 : display

newpage  /' 페이지 분할 '/
A1 -> B2 : select Math 101
B2 -> C1 : getOffering(c: Course)
C1 -> E1 : getOffering()
E1 -> E2 : get offering
E2 --> E1 : course offering
E1 --> C1 : course offering
C1 -> B2 : course offering
B2 -> B2 : display course offering

@enduml
```

실습03-2: sd-03 다시 그리기



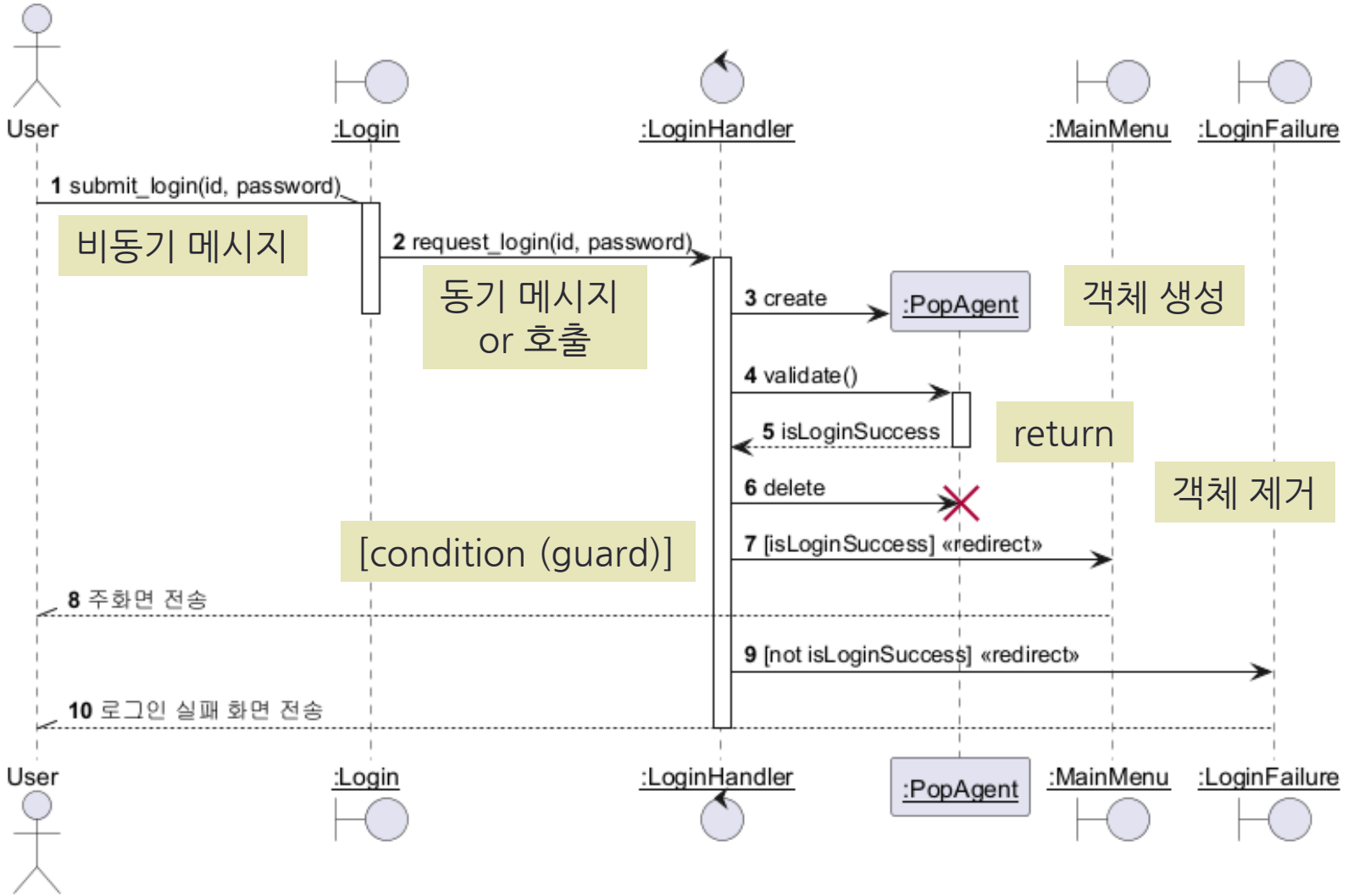

:Professor
:ProfessorCourseOption
:AddCourseOffering
:ProfessorCourseManager
c:Course
:CourseOffering
select Math 101
getOffering(c: Course)
getOffering()
get offering
course offering
course offering
course offering
display course offering
select offering
:Professor
:ProfessorCourseOption
:AddCourseOffering
:ProfessorCourseManager
c:Course
:CourseOffering

PlantUML 코드: sd-03-2.txt

```
@startuml

actor "<u>:Professor" as A1
boundary "<u>:ProfessorCourseOption" as B1  #orange
boundary "<u>:AddCourseOffering" as B2 #orange

control "<u>:ProfessorCourseManager" as C1  #cyan
entity "<u>c:Course" as E1  #FFFF66
entity "<u>:CourseOffering" as E2  #FFFF66

A1 -> B1 ++ : enter password
B1 -> B1 : verify password
A1 <-- B1 -- : authentication result
A1 -> B1 ++: enter semester
deactivate

A1 -> B1 ++: add an offering
deactivate

B1 -> B2 ++: display
deactivate

newpage /' 페이지 분할 '/
A1 -> B2 ++: select Math 101
B2 -> C1 ++ : getOffering(c: Course)
C1 -> E1 ++: getOffering()
E1 -> E2 ++: get offering
return course offering
return course offering
return course offering
```

```
B2 -> B2 ++: display course offering
deactivate
deactivate

A1 -> B2 ++: select offering

@enduml
```

실습04: 생명선(Lifeline)



PlantUML 코드: sd-04.txt

```
@startuml

actor User as A1
boundary "<u>:Login" as PL
control "<u>:LoginHandler" as PLH
participant "<u>:PopAgent" as PPA
boundary "<u>:MainMenu" as PMM
boundary "<u>:LoginFailure" as PLF

autonumber 1
A1 -\\ PL : submit_login(id, password)
activate PL
```

- ++ 대상활성화 (추가로색을따를수있습니다.)
- -- 원본비활성화
- ** 대상인스턴스생성
- !! 대상인스턴스파괴

```
' autonumber 1.1
PL -> PLH : request_login(id, password)
activate PLH

PLH -> PPA ** : create

deactivate PL

PLH -> PPA : validate()
activate PPA
return isLoginSuccess
' PLH <-- PPA : isLoginSuccess 와 동일함.
' return이 더 간단함.
PLH -> PPA !! : delete
```

```
PLH -> PMM : [isLoginSuccess] <<redirect>>
PMM --\\ A1 :

PLH -> PLF : [not isLoginSuccess] <<redirect>>

PLF --\\ A1 :
deactivate PLH

@enduml
```

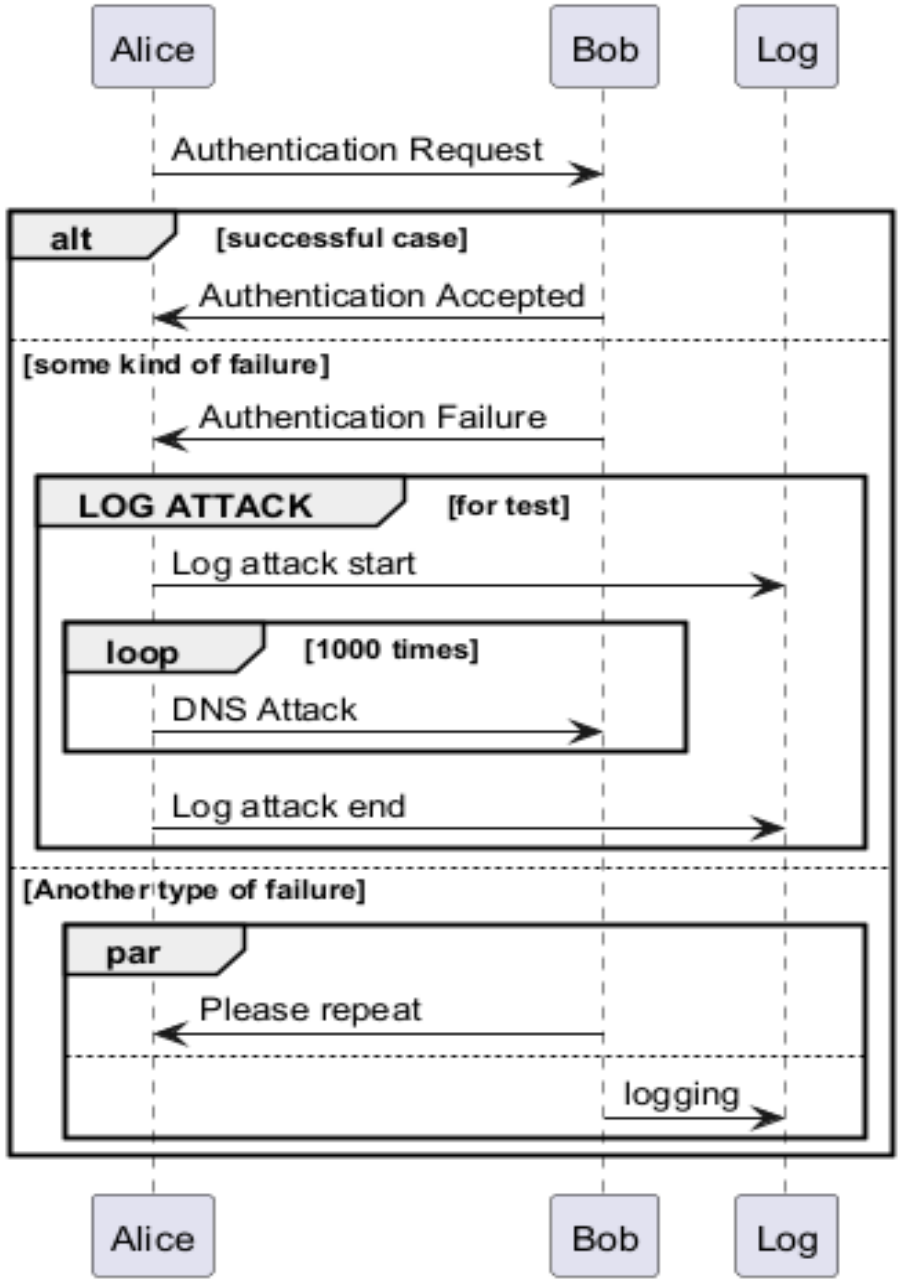
PlantUML 코드: sd-04-2.txt

```
@startuml

actor User as A1
boundary "<u>:Login" as PL
control "<u>:LoginHandler" as PLH
participant "<u>:PopAgent" as PPA
boundary "<u>:MainMenu" as PMM
boundary "<u>:LoginFailure" as PLF

autonumber 1
A1 -\\ PL : submit_login(id, password)

' autonumber 1.1
PL -> PLH ++ : request_login(id, password)

PLH -> PPA ** : create

deactivate PL

PLH -> PPA ++: validate()
return isLoginSuccess

PLH -> PPA !! : delete

PLH -> PMM : [isLoginSuccess] <<redirect>>
PMM --\\ A1 : 주화면 전송

PLH -> PLF -- : [not isLoginSuccess]
<<redirect>>

PLF --\\ A1 : 로그인 실패 화면 전송

@enduml
```

```
PLH -> PMM : [isLoginSuccess] <<redirect>>
PMM --\\ A1 :

PLH -> PLF : [not isLoginSuccess] <<redirect>>

PLF --\\ A1 :
deactivate PLH

@enduml
```

실습05: 메시지 그룹

- 메시지를 그룹으로 만들어 관리 가능
- 사용 가능한 키워드
  - alt/else
  - opt
  - loop
  - par
  - break
  - critical



PlantUML 코드: sd-05.txt

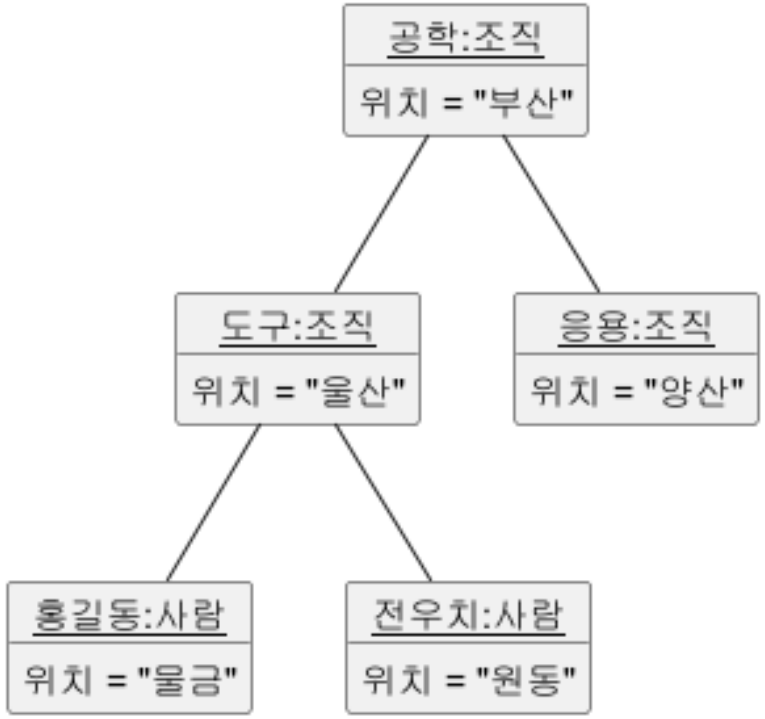
```
@startuml
Alice -> Bob: Authentication Request

alt successful case
    Bob -> Alice: Authentication Accepted

else some kind of failure
    Bob -> Alice: Authentication Failure

group LOG ATTACK [for test]
    Alice -> Log : Log attack start

    loop 1000 times
        Alice -> Bob: DNS Attack
    end

    Alice -> Log : Log attack end
end
```

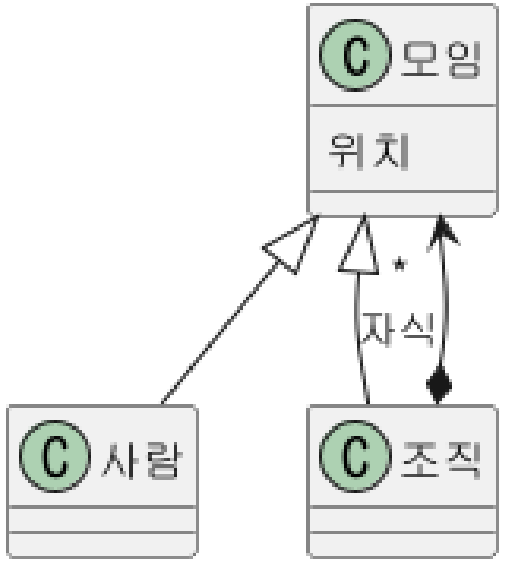
```
else Another type of failure
    par
        Bob -> Alice: Please repeat
    else
       Bob -> Log : logging
    end
end
@enduml
```

통신 다이어그램

실습01: 객체 다이어그램

- 클래스의 실체인 객체와 그 객체 간의 관계 표현
- 실행 중 시스템의 정적인 snap shot 시각화
- 특정 시점의 클래스 다이어그램 상황
- 주로 시스템 설계와 디버깅에서 특정 시점의 데이터 상태를 파악하는 데 유용



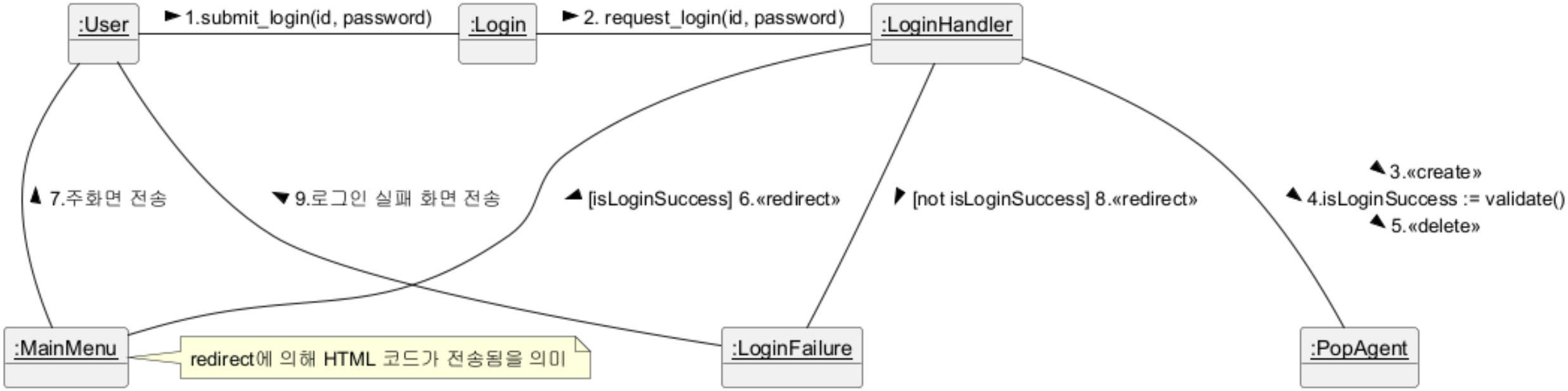


PlantUML 코드: od-01.txt

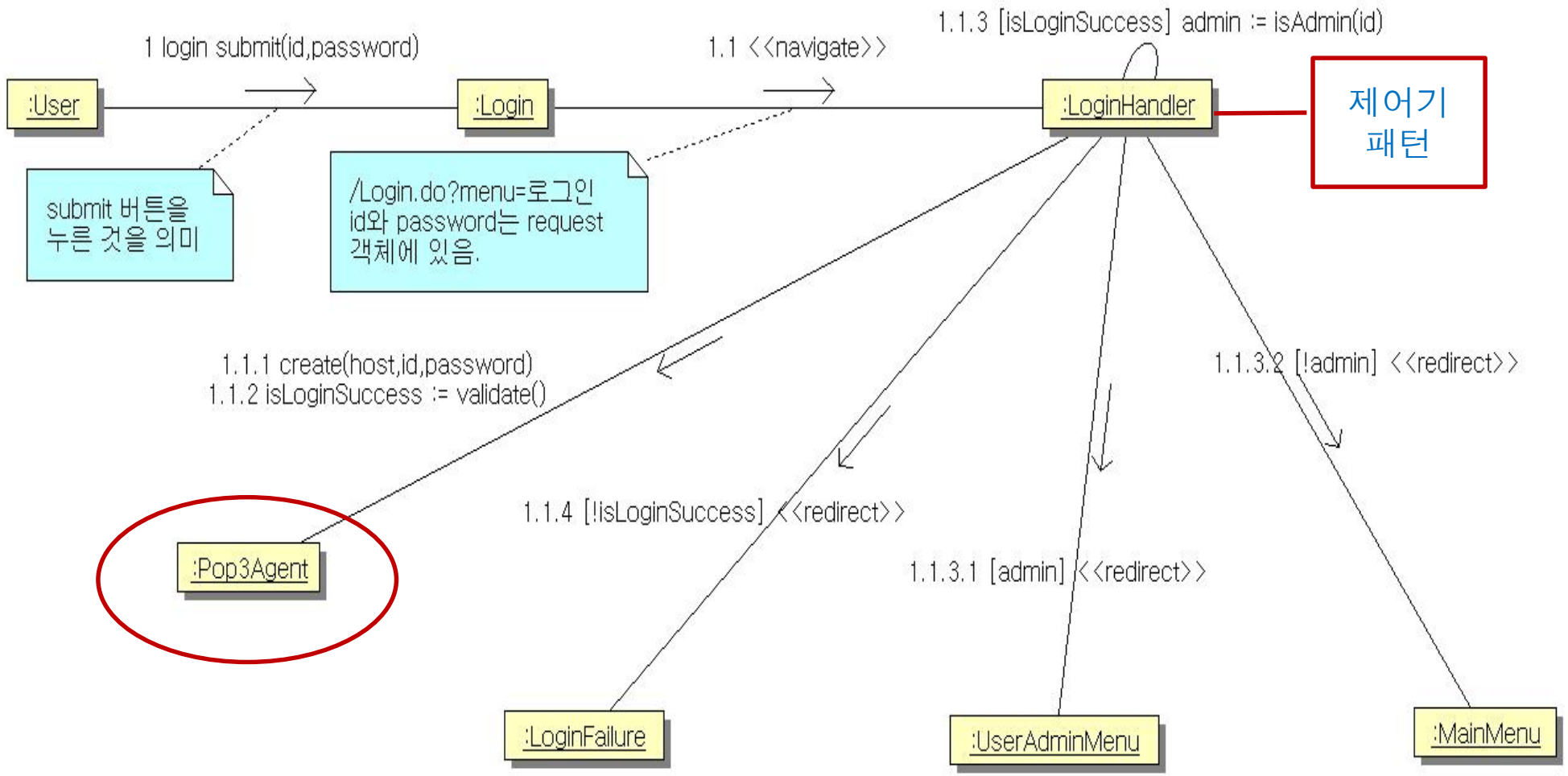
```
@startuml

object "__공학:조직__" as 01
01 : 위치 = "부산"

object "<u>도구:조직" as 02
02 : 위치 = "울산"

object "<u>응용:조직" as 03
03 : 위치 = "양산"

object "__홍길동:사람__" as 04
04 : 위치 = "물금"

object "__전우치:사람__" as 05
05 : 위치 = "원동"

01 -- 02
01 -- 03

02 -- 04
02 -- 05

@enduml
```

실습02: 순차 다이어그램 실습04

- 통신 다이어그램은 순차 다이어그램과 동일한 정보 표현 가능



(참고) boumI로 그린 통신 다이어그램



PlantUML 코드: od-02.txt

```
@startuml

object "<u>:User" as User
object "<u>:Login" as Login
Object "<u>:LoginHandler" as LH
Object "<u>:PopAgent" as PopAgent
Object "<u>:MainMenu" as MainMenu
Object "<u>:LoginFailure" as LoginFailure

User -r-- Login : 1.submit_login(id, password) >
Login -r-- LH : 2. request_login(id, password) >

LH --- PopAgent : 3.<<create>> >\n4.isLoginSuccess := validate() >\n5.<<delete>> >
LH --- MainMenu : [isLoginSuccess] 6.<<redirect>> >
```

```
MainMenu --- User : 7.주화면 전송 >

note as N1
redirect에 의해 HTML 코드가 전송됨을 의미
end note

MainMenu -right- N1

LH --- LoginFailure : [not isLoginSuccess] 8.<<redirect>> >
LoginFailure --- User : 9.로그인 실패 화면 전송 >

@enduml
```


Q
A