PLANTUML 실습

동의대학교 컴퓨터소프트웨어공학과 이종민교수

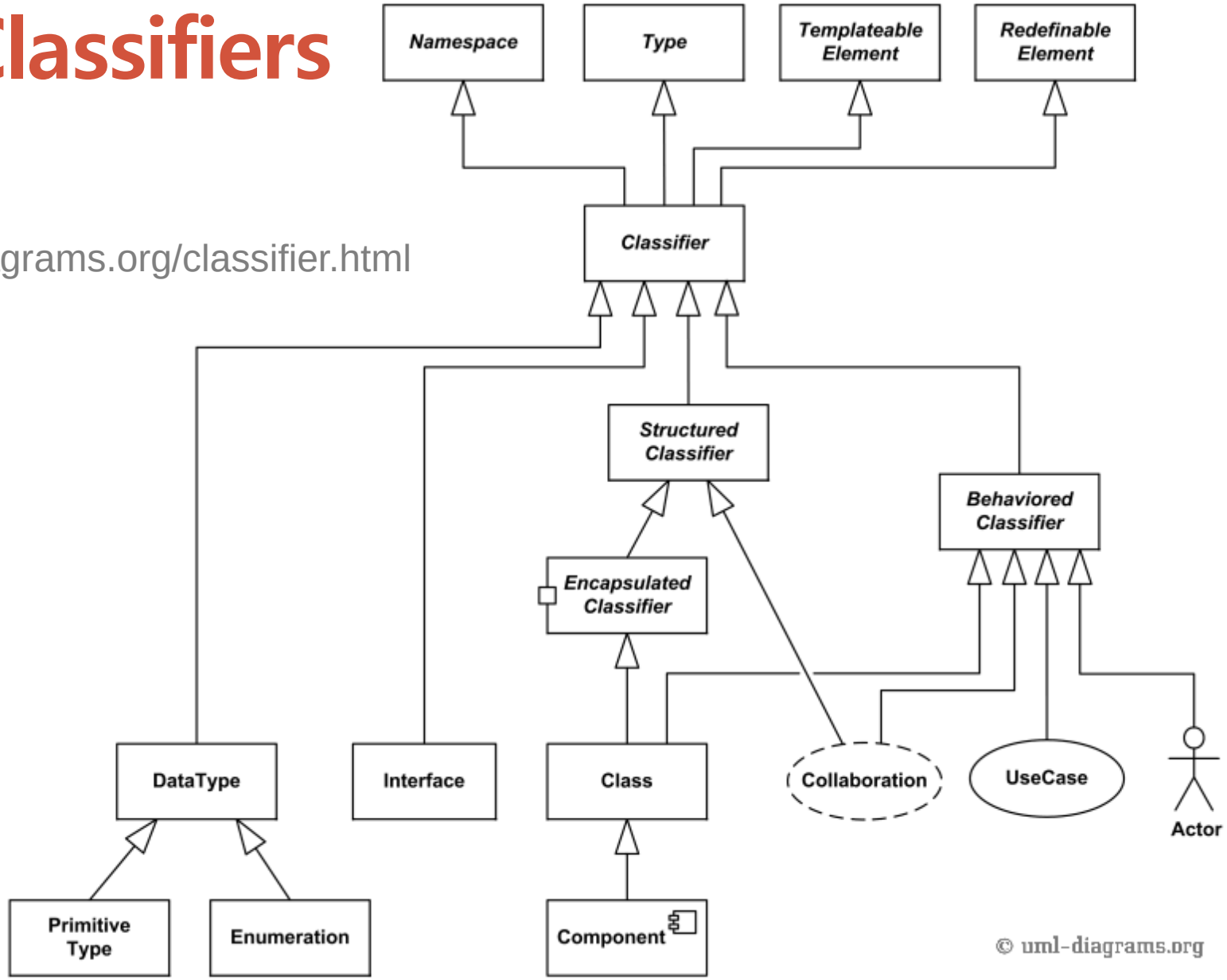
PlantUML 소개

- 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

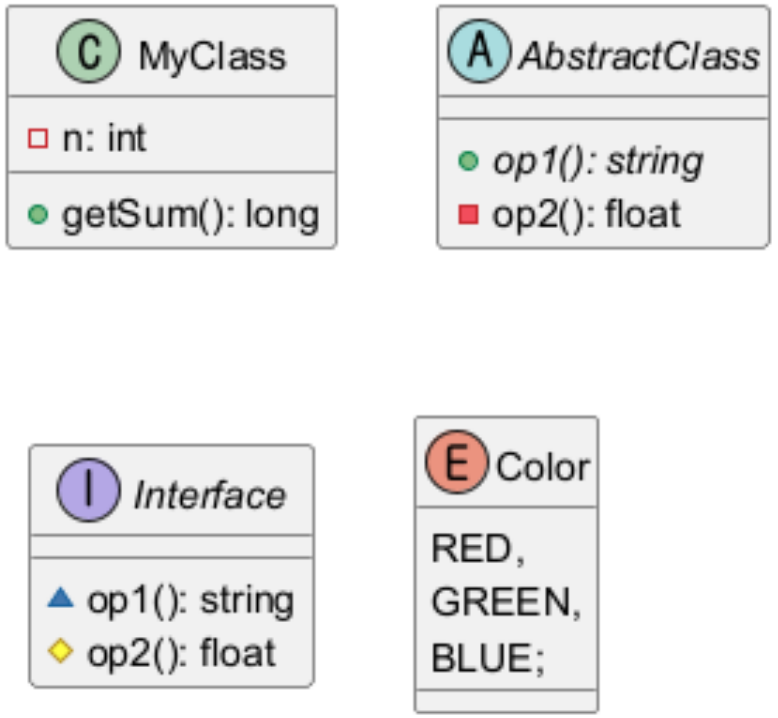
cf. https://www.uml-diagrams.org/classifier.html



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();

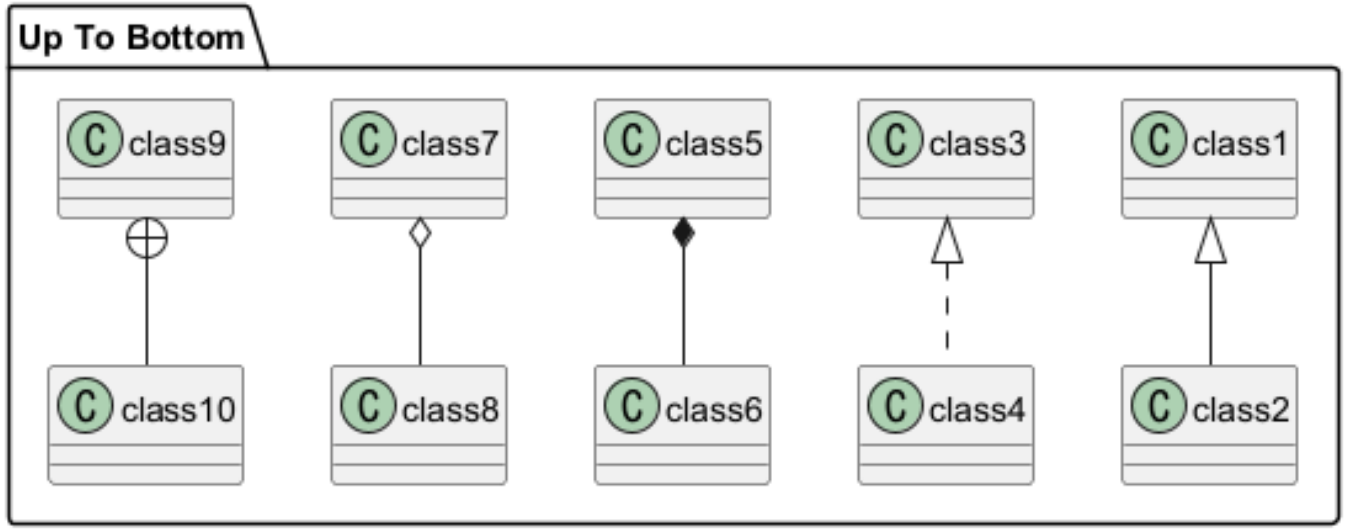
PlantUML 코드 샘플: UML 방식

```
@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
```

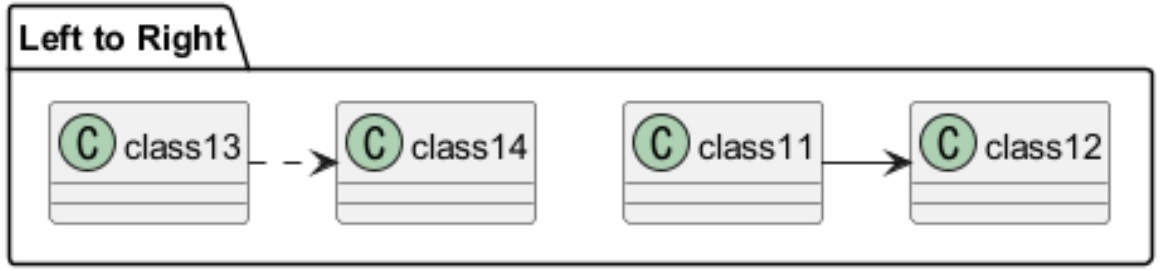
PlantUML 코드 샘플: Java 방식

```
@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.

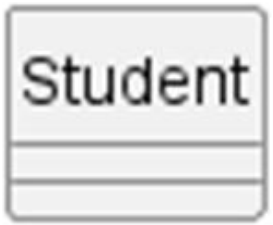
- 예제 → 대응하는 자바 코드
  - **+ name : String → public String name;**
  - **- n : int → private int n;**
  - **# animalType : Animal → protected Animal animalType;**

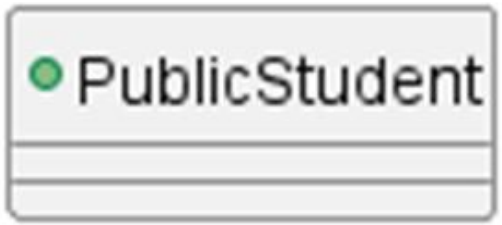
- 자바: 인스턴스 변수 또는 필드(field)
  - *cf.* https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html
- C++: 멤버 변수 (member variable)

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





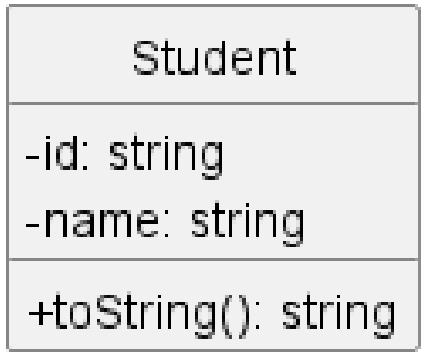

```
class Student {
}
```

```
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



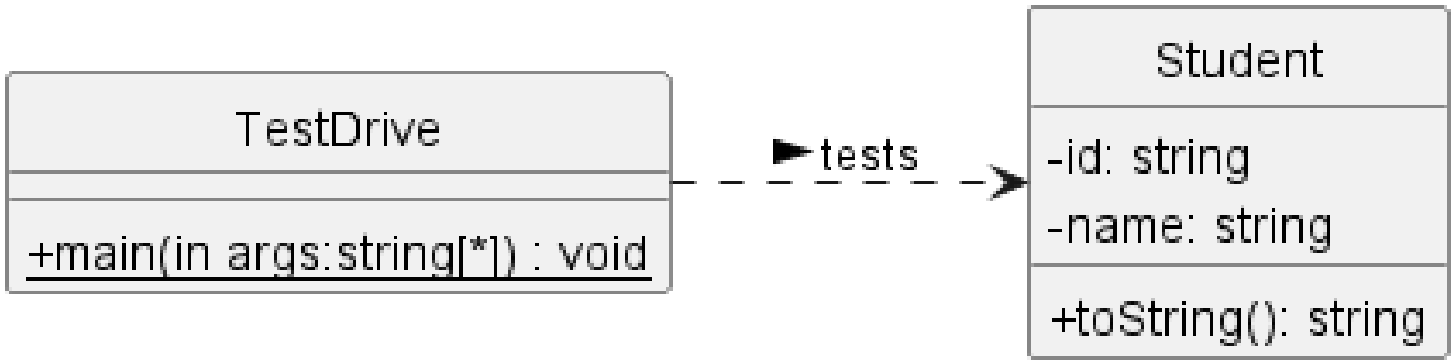

```
public class Student {
    private String id;
    private String name;

    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.

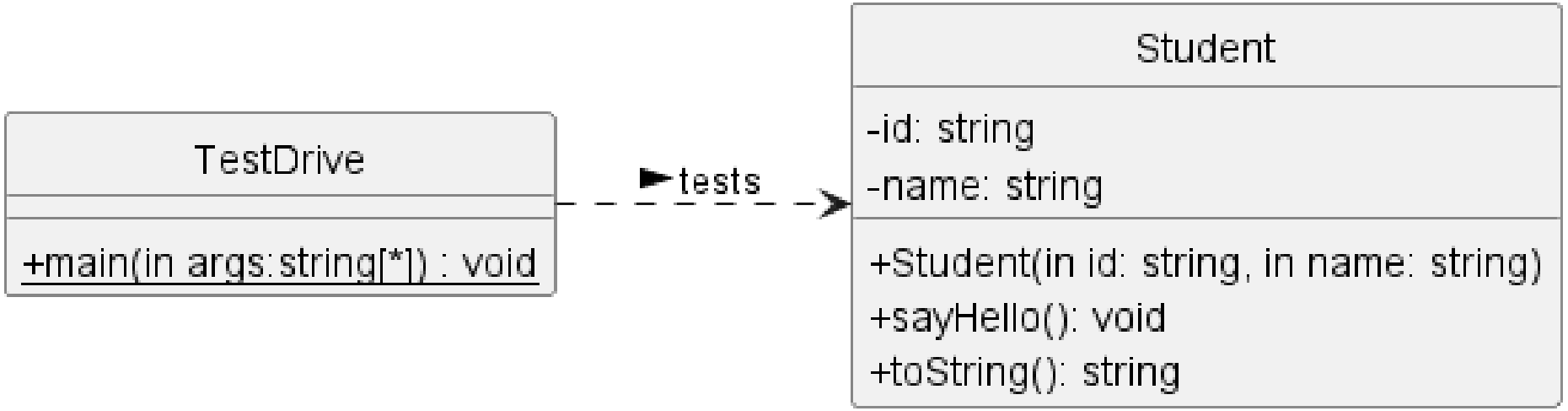


PlantUML 코드: cd-03.txt

```
@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

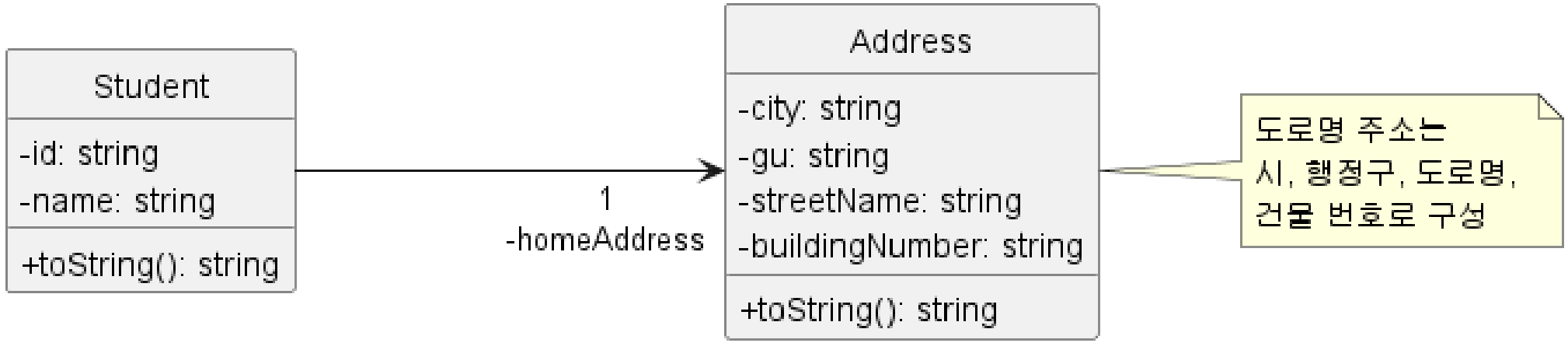


PlantUML 코드: cd-04.txt

```
@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

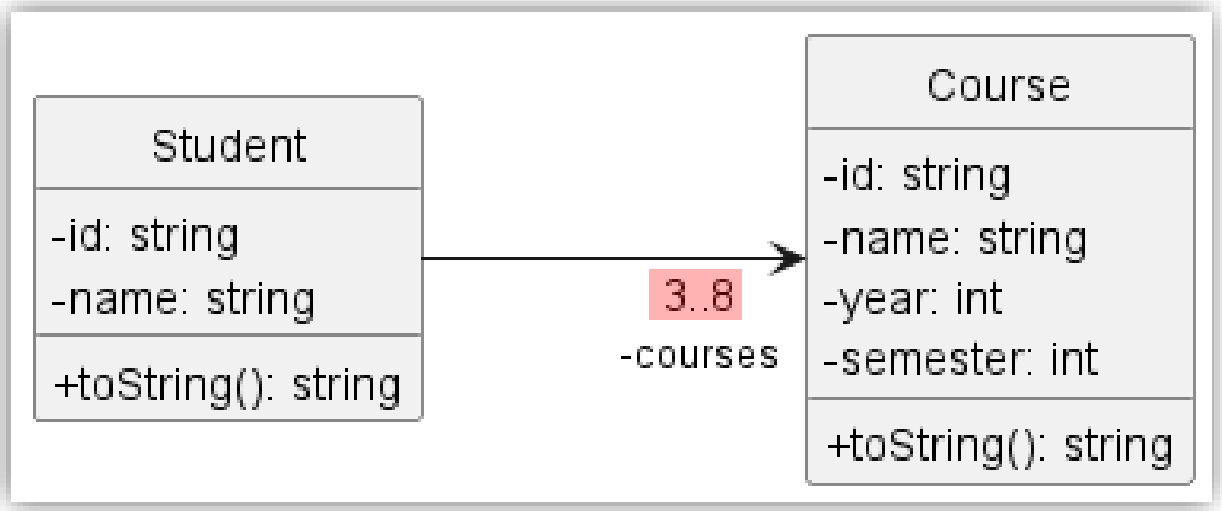
```
class Student {
  // 이전 생략
  private Address homeAddress;
  // 이하 생략
}
```

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



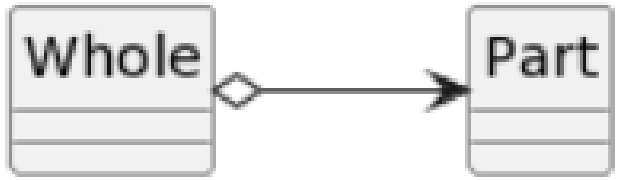
PlantUML 코드: cd-06.txt

```
@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'/
```

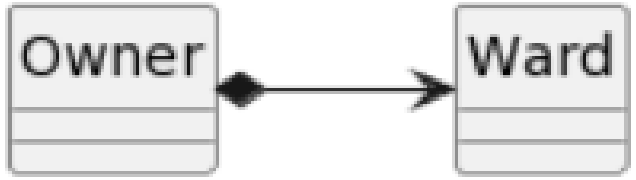
실습07: 집합과 합성

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

집합



합성

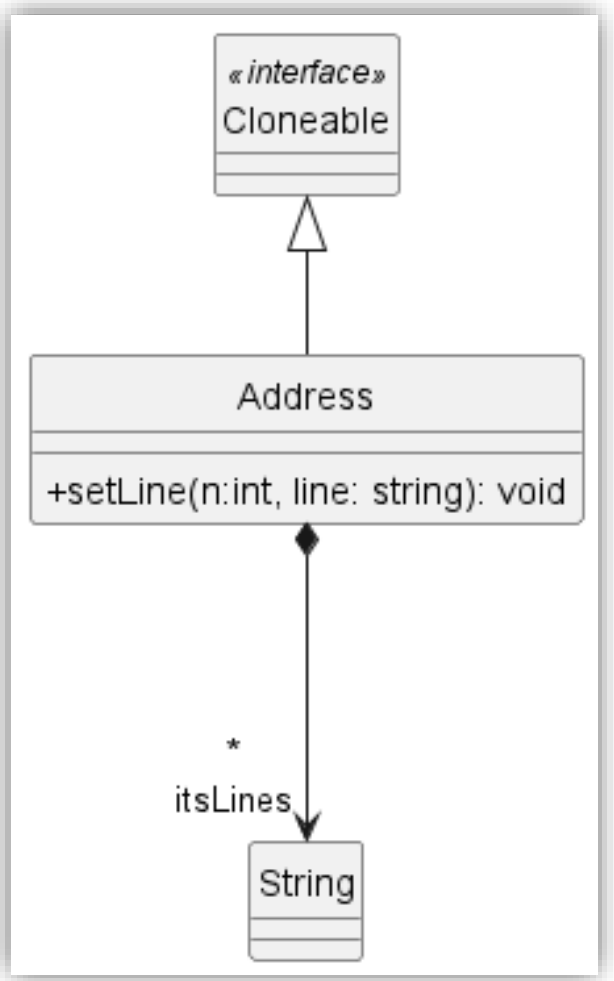


PlantUML 코드: cd-07.txt

```
@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

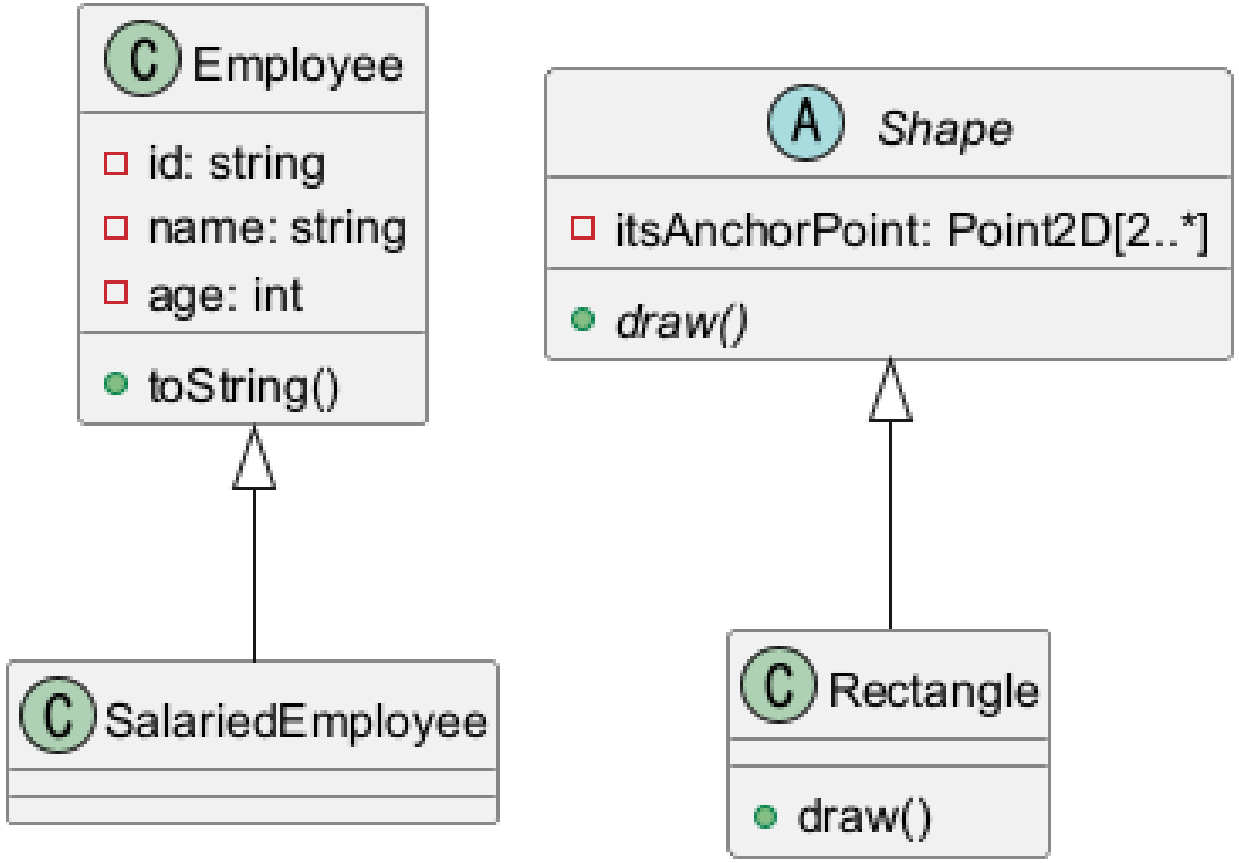
- Address 객체 복사할 때 Vector로 구현된 itsLines 인스턴스 변수도 같이 복사하여 생성해야 함.



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

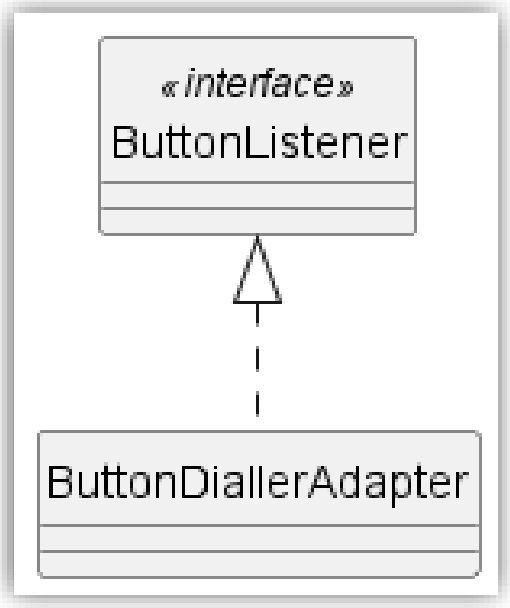


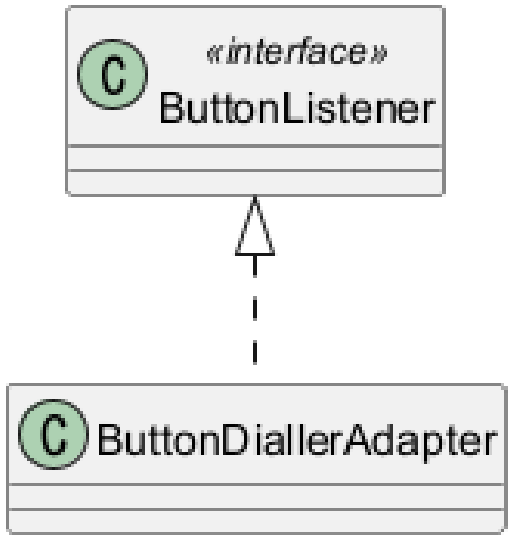
PlantUML 코드: cd-09.txt

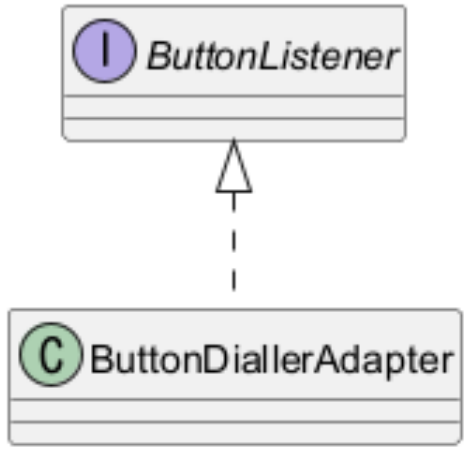
```
@startuml

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





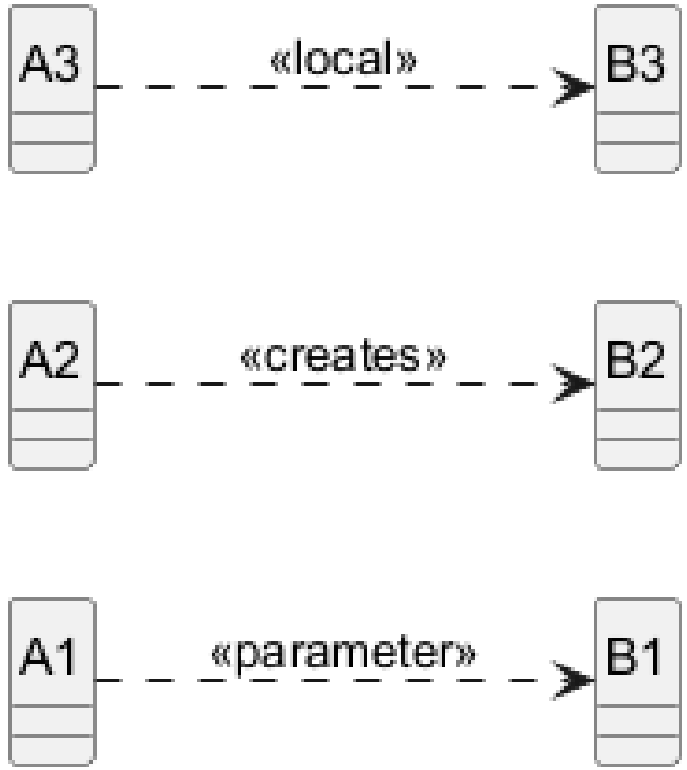


PlantUML 코드: cd-10.txt

```
@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 */ }
```

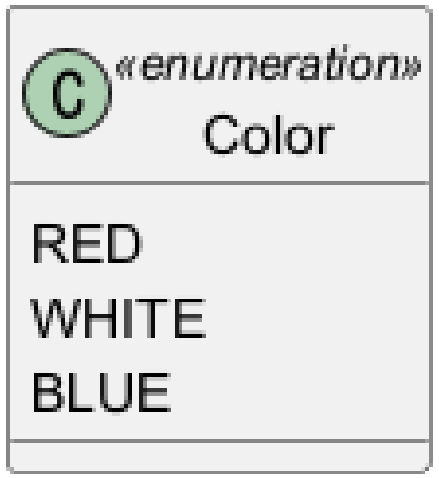
의존성 코드 예 (2/2)

[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)
```


Q
A