둘중에 하나만 골라,

상속 or 컴포지션

갑자기…?

재활용은 쌉이득이다

객체지향에서 재사용성을 중요시하는 이유

> “
> Don’t reinvent the wheel
> **바퀴를 다시 발명하지 마라**
> ”

- 바퀴는 이미 개발되어있다
  이미 동작과 상태가 명확한 물체
- 이걸 그냥 사다가 다른 유용한 물체를 만들자
  예) 자동차
  바퀴를 내가 굳이 만들어서 시간 낭비할 필요 X

상속과 컴포지션, 둘 중 어떤 방법으로 재사용할 것인가?

상속 vs 컴포지션

- 상속과 컴포지션 둘 다 **재사용성이 목적**
- 가능 / 불가능의 측면에서만 보면 많은 경우에 둘 다 사용가능

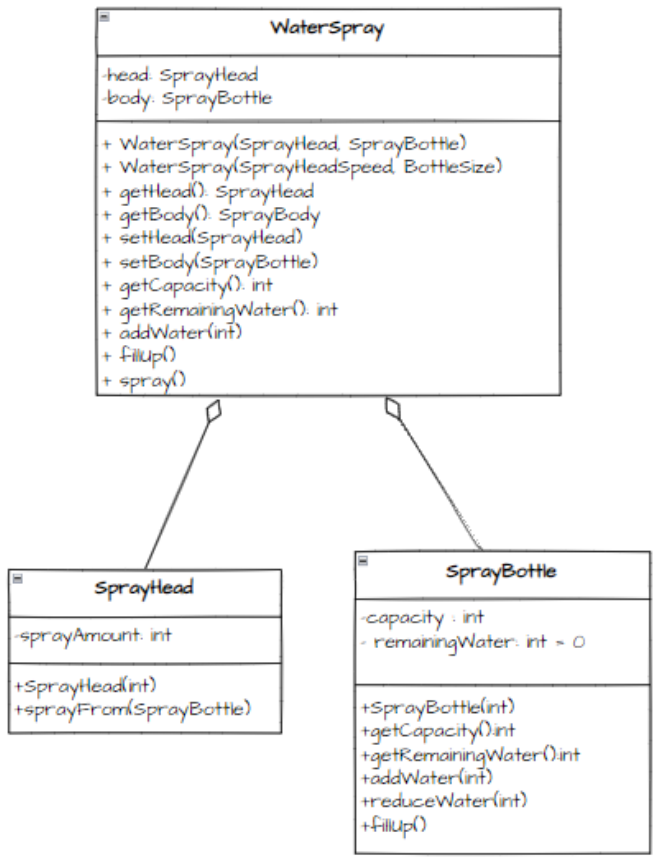
바퀴를 **상속** 받아 사용할 것인가?

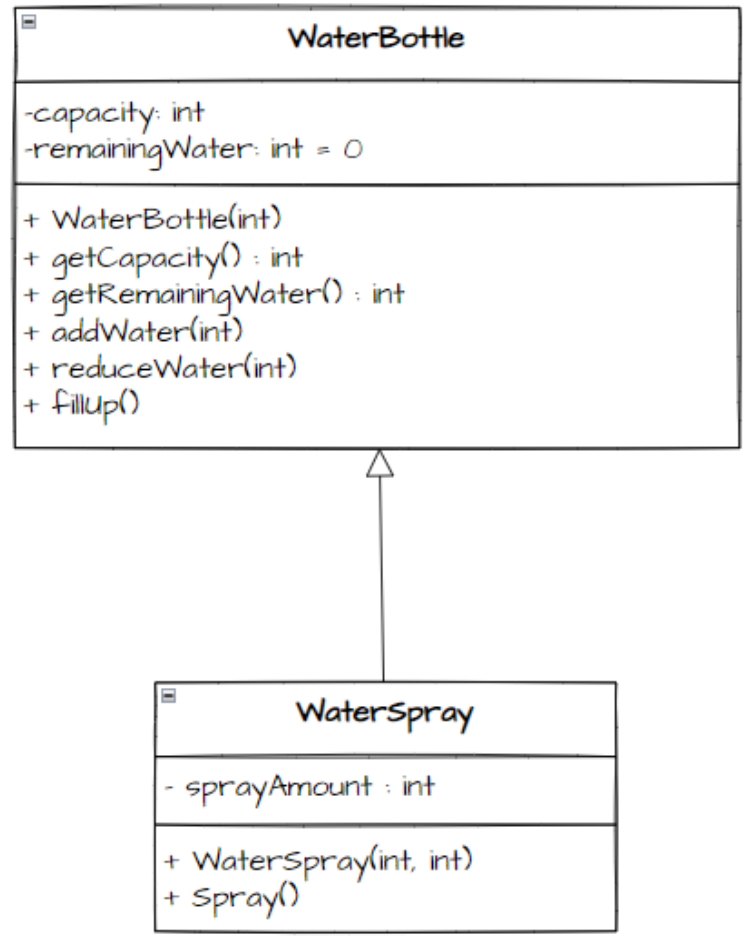
아니면

내부에 바퀴를 가지고 있는 **컴포지션**으로 만들 것인가?

컴포지션을 상속으로 바꾼 예

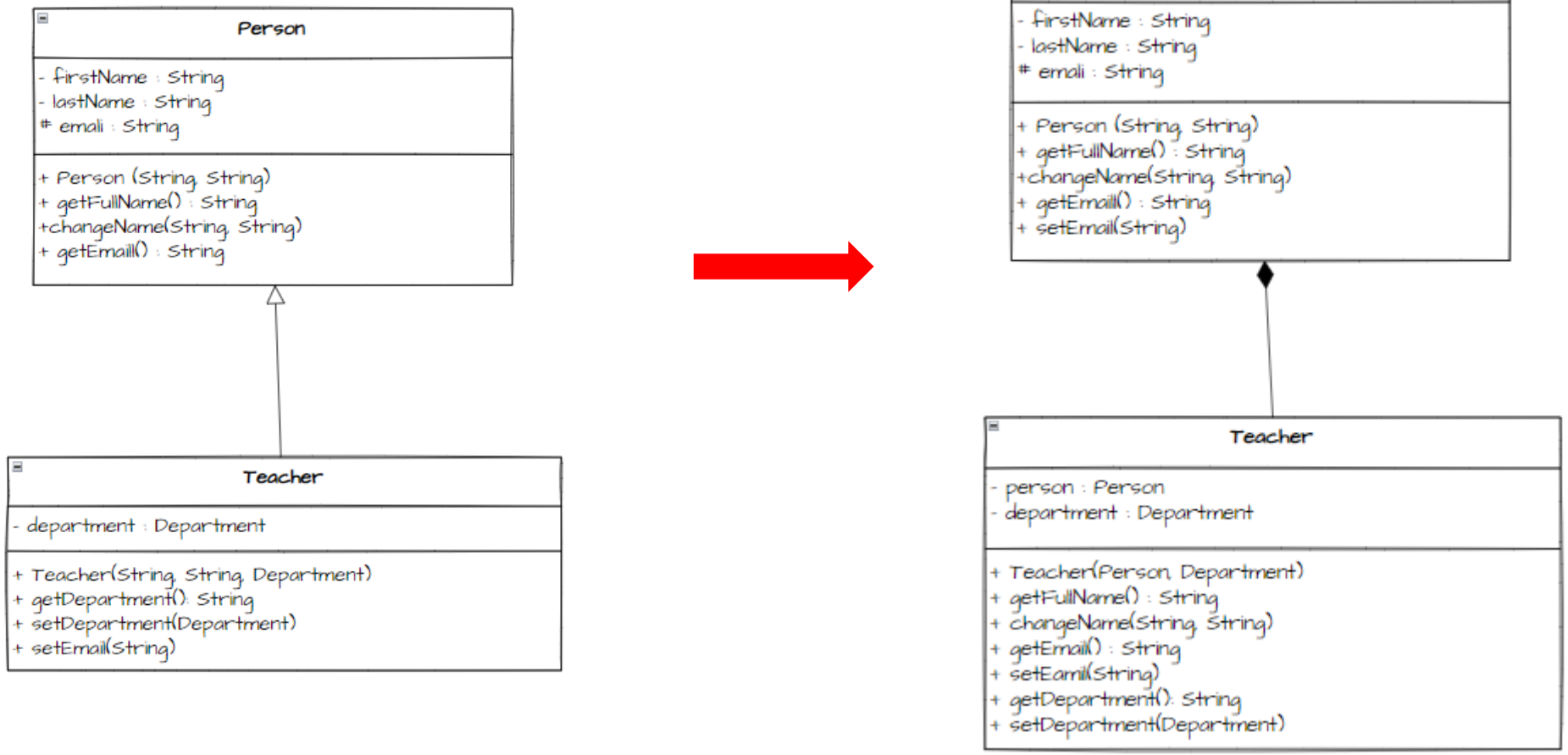
분무기





상속을 컴포지션으로 바꾼 예

사람과 선생



코드를 재활용하기 위해 상속과 컴포지션을 사용
이 둘은 서로 대체해서 모델링 하는게 가능은 하다

어떤 경우에서는 상속이 더 효율적이고,
또 어떤 다른 경우에는 컴포지션이 더 좋을때가 있다.

지금부터 “어떤 경우” 에 대해 알아보고
그리고 각 상황마다 어떤 것을 선택해야 하는지에 대해 알아보자!

상속 or 컴포지션 ?

언제 무엇을 사용할지는 아래의 경우로 나눠 생각해보자

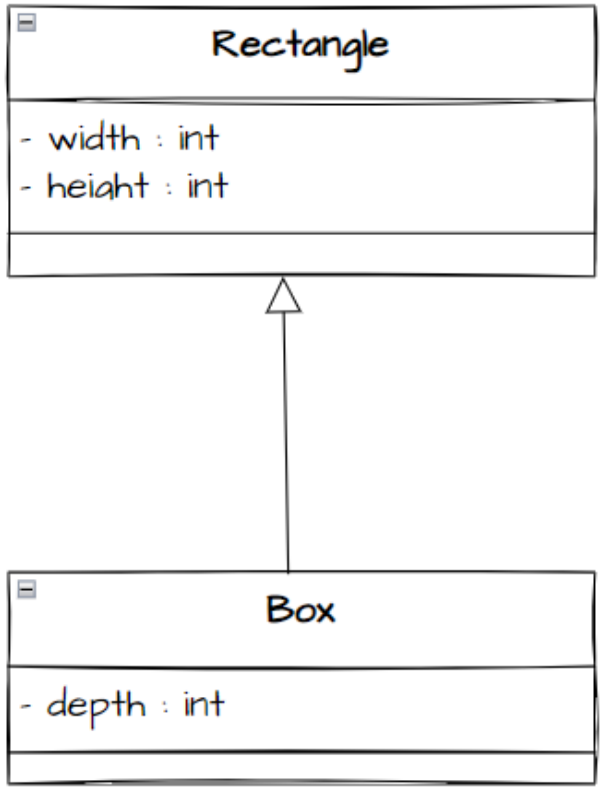
1. 기계상의 차이 때문에 하나를 골라야할 때
2. 용도 때문에 상속을 고를 수 밖에 없을 때
3. 관리의 효율성을 고려할 때
4. 그외 일반적인 상황

참고 : Udemy_개체지향 프로그래밍 및 설계

1. **기계상의 차이 때문에 하나를 골라야할 때**
2. 용도 때문에 상속을 고를 수 밖에 없을 때
3. 관리의 효율성을 고려할
4. 그외 일반적인 상황

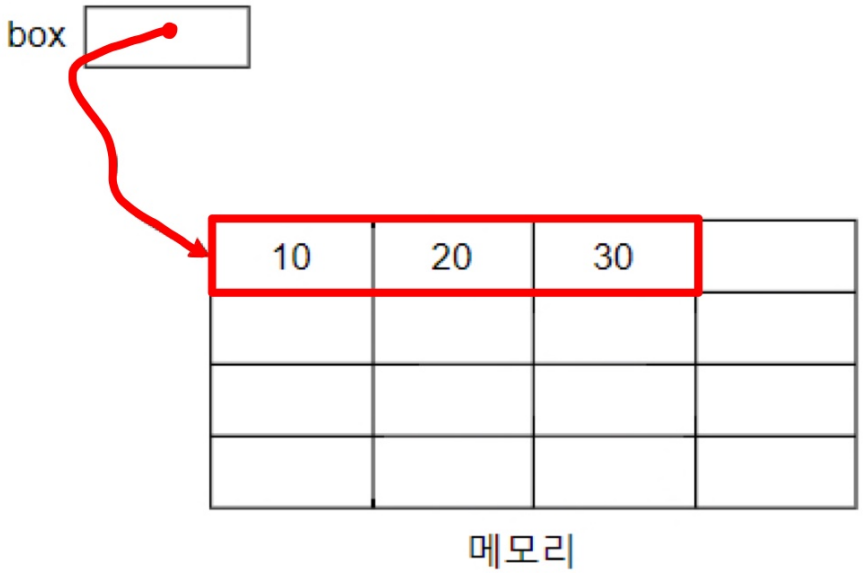
상속과 메모리

개체 생성 시, 메모리가 **하나의 덩어리**



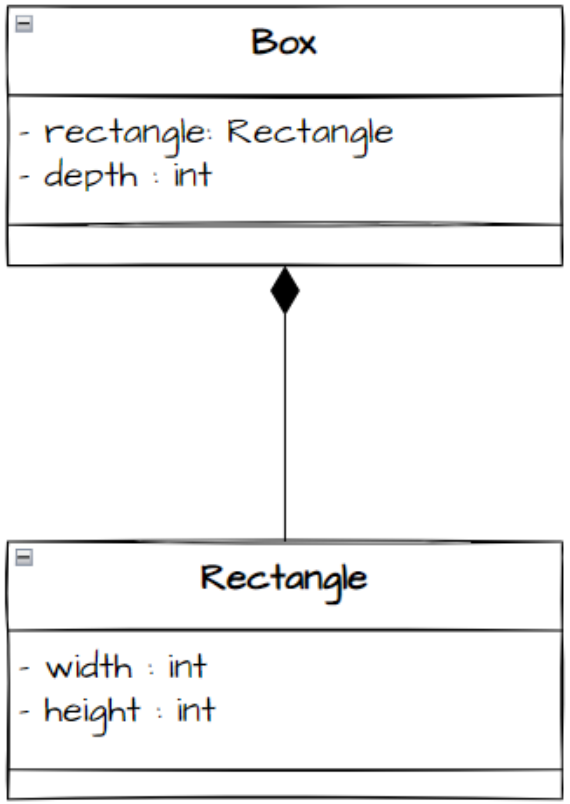
(멤버 변수만 보여줌)

```
Box box = new Box(10,20,30); // width, height, depth
```



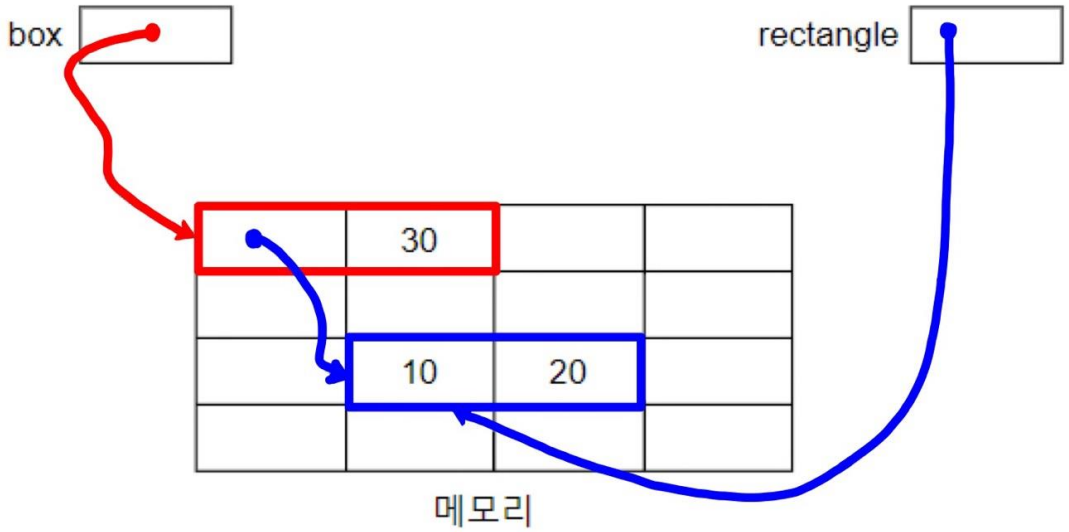
컴포지션과 메모리

개체 생성 시, 메모리가 **여러 덩어리**



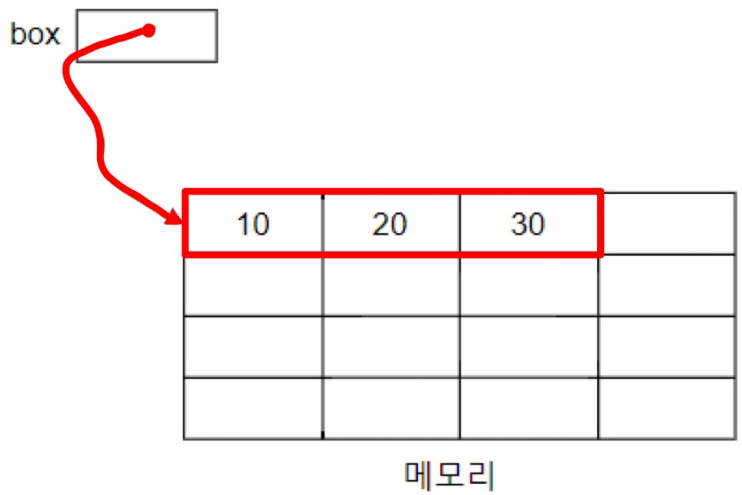
(멤버 변수만 보여줌)

```
Rectangle rectangle = new Rectangle(10,20);
Box box = new Box(rectangle, 30);
```



이러한 메모리 상의 차이는 실행 성능에 영향을 준다

상속 모델

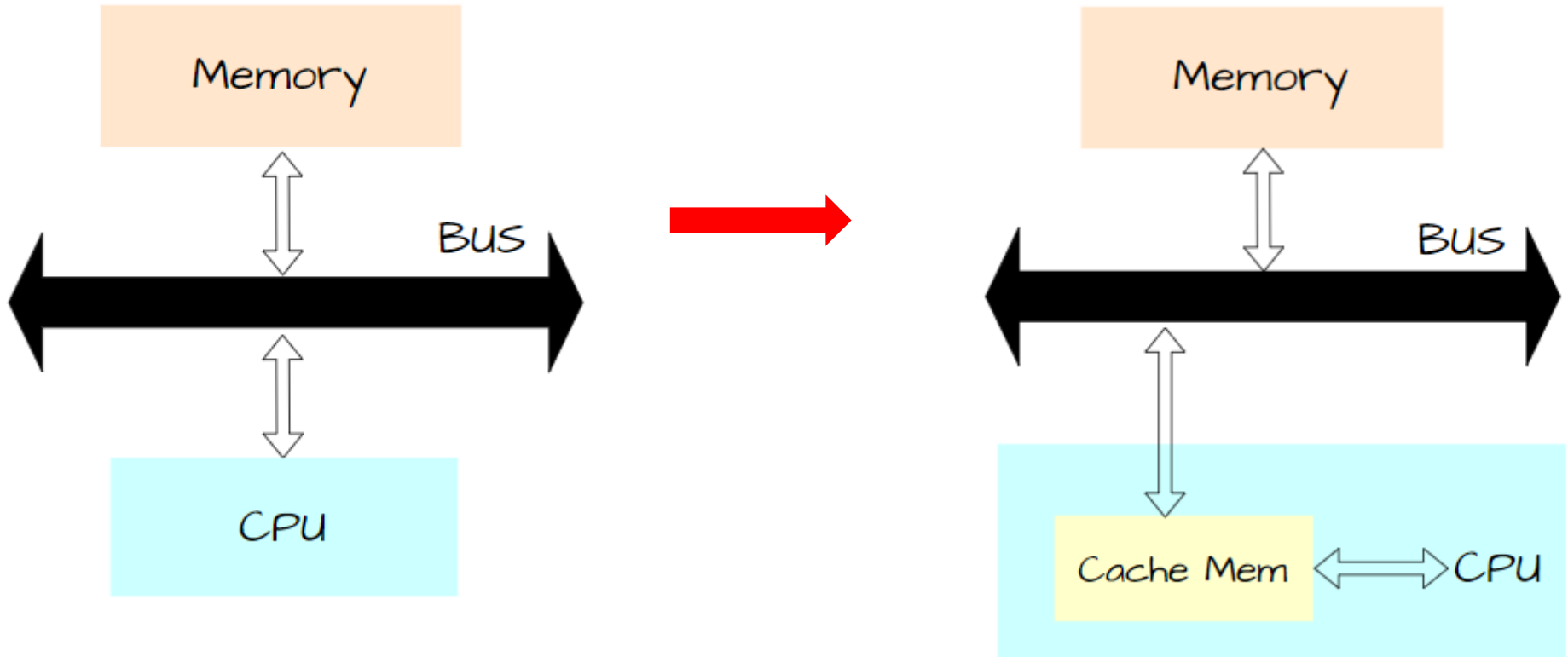


컴포지션 모델

프로그램 실행 중 첫번째 병목점

CPU 와 메모리 사이의 데이터 전송

CPU에 고속의 캐시 메모리를 탑재

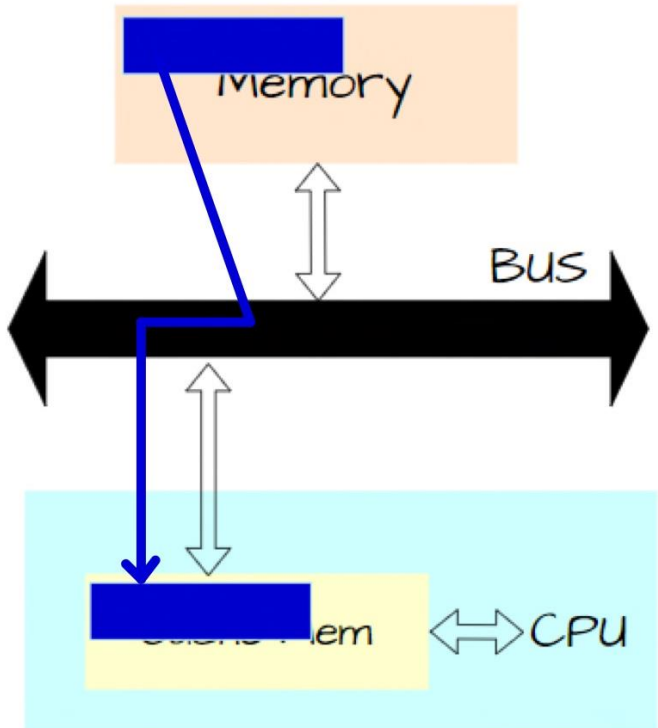


프로그램 실행 중 첫번째 병목점

상속 모델로 만든 객체

상속 모델로 만든 객체

객체가 한 번에 캐시 메모리에 들어갈 가능성이 높음

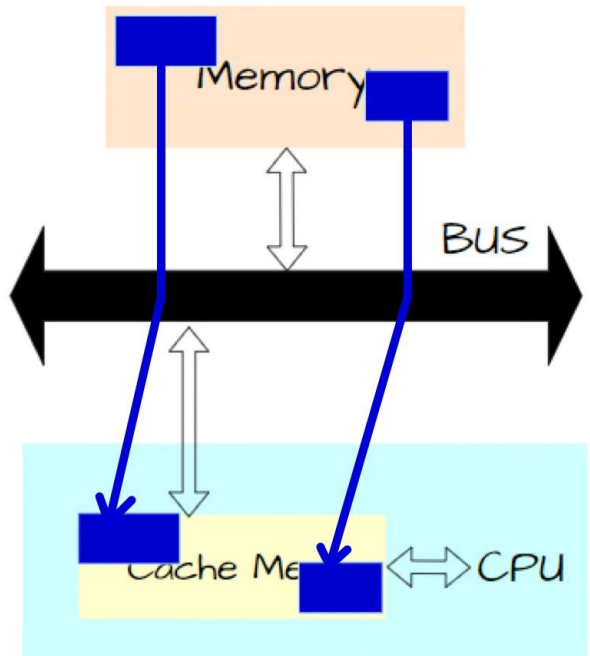


프로그램 실행 중 첫번째 병목점

컴포지션 모델로 만든 객체

컴포지션 모델로 만든 객체

개체 내 부품 수만큼 캐시 메모리로 로딩할 가능성이 높음



프로그램 실행 중 두번째 병목점

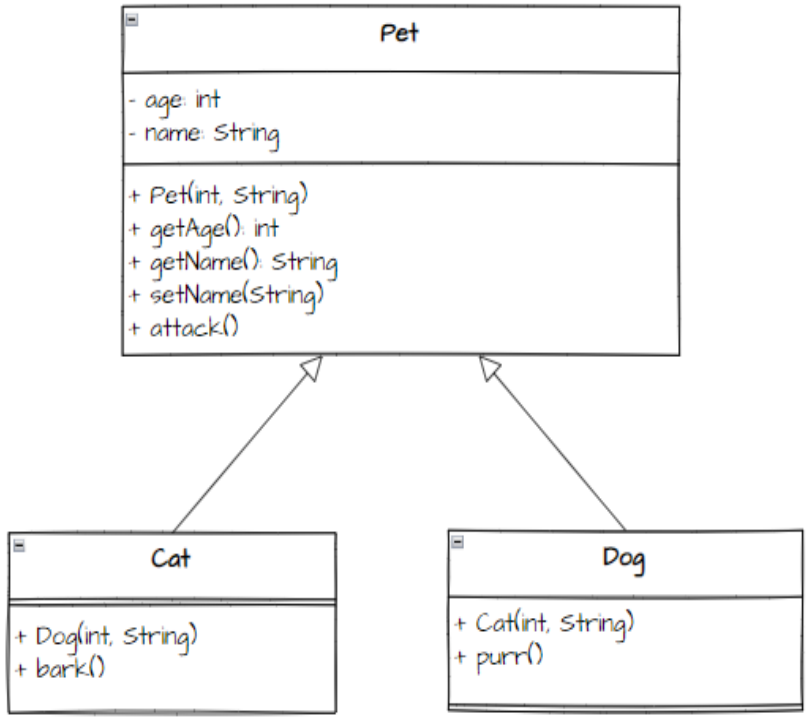
메모리 할당과 해제

- 새로운 메모리 할당(new)과 해제(delete)
- 상속은 메모리 딱 한 번씩
- 컴포지션은 한번 +부품 수 만큼씩

1. 기계상의 차이 때문에 하나를 골라야할 때
2. **용도 때문에 상속을 고를 수 밖에 없을 때**
3. 관리의 효율성을 고려할 때
4. 그외 일반적인 상황

다른 형의 객체들을 한꺼번에 처리하고 싶을 때

한꺼번에 attack!



- 다양한 객체를 리스트에 저장
- 거기서 같은 함수를 호출 (다형적으로)

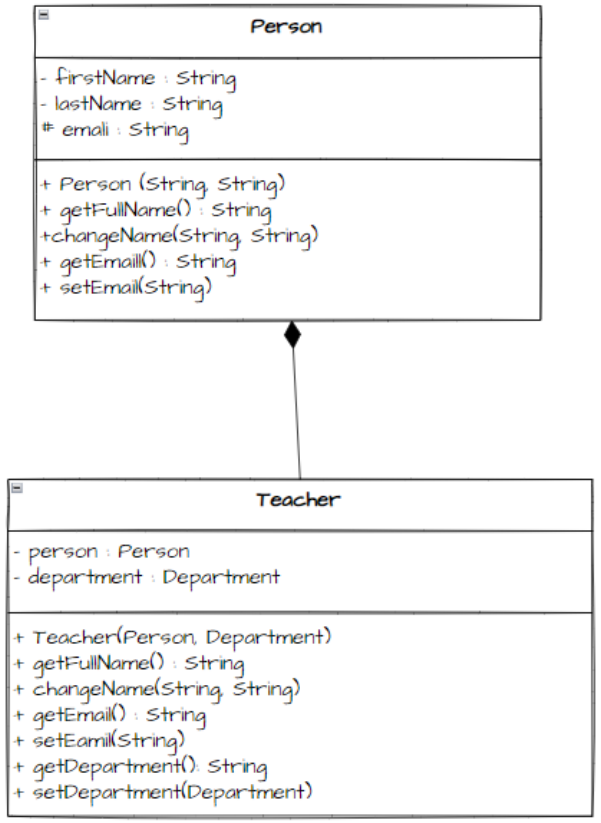
```
ArrayList<Pet> pets = new
ArrayList<Pet>();
pets.add(new Dog(3, "doldol"));
pets.add(new Cat(4, "nabi"));

for (Pet pet : pets){
    pet.attack();
}
```

1. 기계상의 차이 때문에 하나를 골라야할 때
2. 용도 때문에 상속을 고를 수 밖에 없을 때
3. **관리의 효율성을 고려할 때**
4. 그외 일반적인 상황

먼저, 컴포지션을 사용하면 관리하기 불편한 경우를 보자

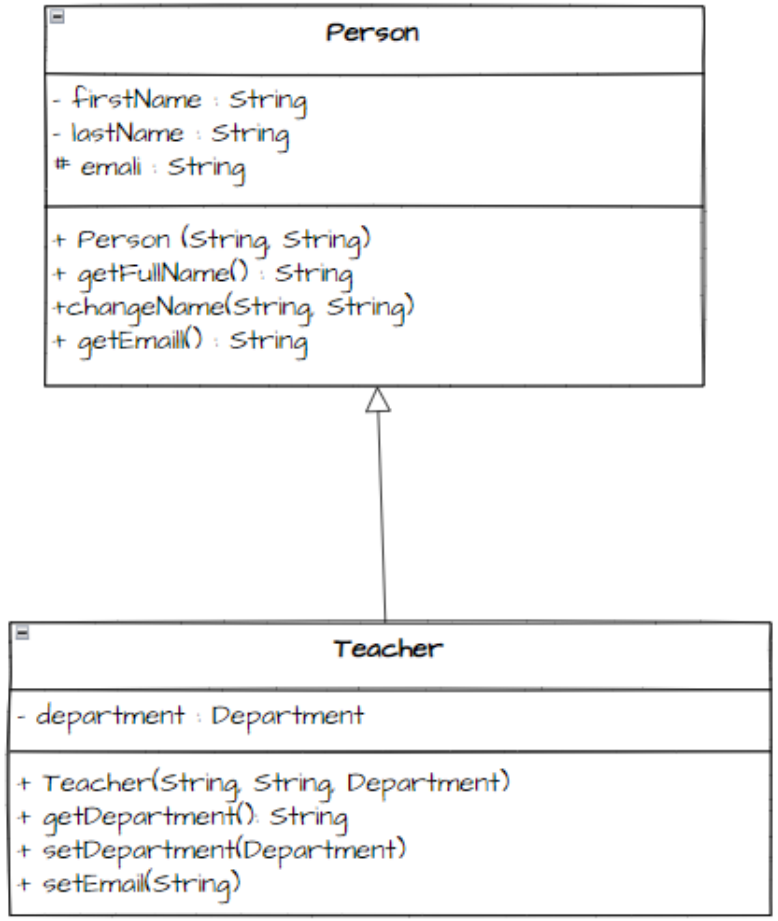
Person – Teacher 컴포지션 모델



- Person의 메서드를 호출하는 메서드 필요

```
class Teacher {
    ...
    public String getFullName(){
        return person.getFullName();
    }
    public void changeName(String firstName, String lastName){
        person.changeName(firstName, lastName);
    }
    ...
}
```

Person – Teacher 상속 모델




```
Teacher teacher = new Teacher(
"Seo", "Minjung", Department.ENGLISH);
System.out.println(teacher.getFullName());
System.out.println(teacher.getDepartment());
```

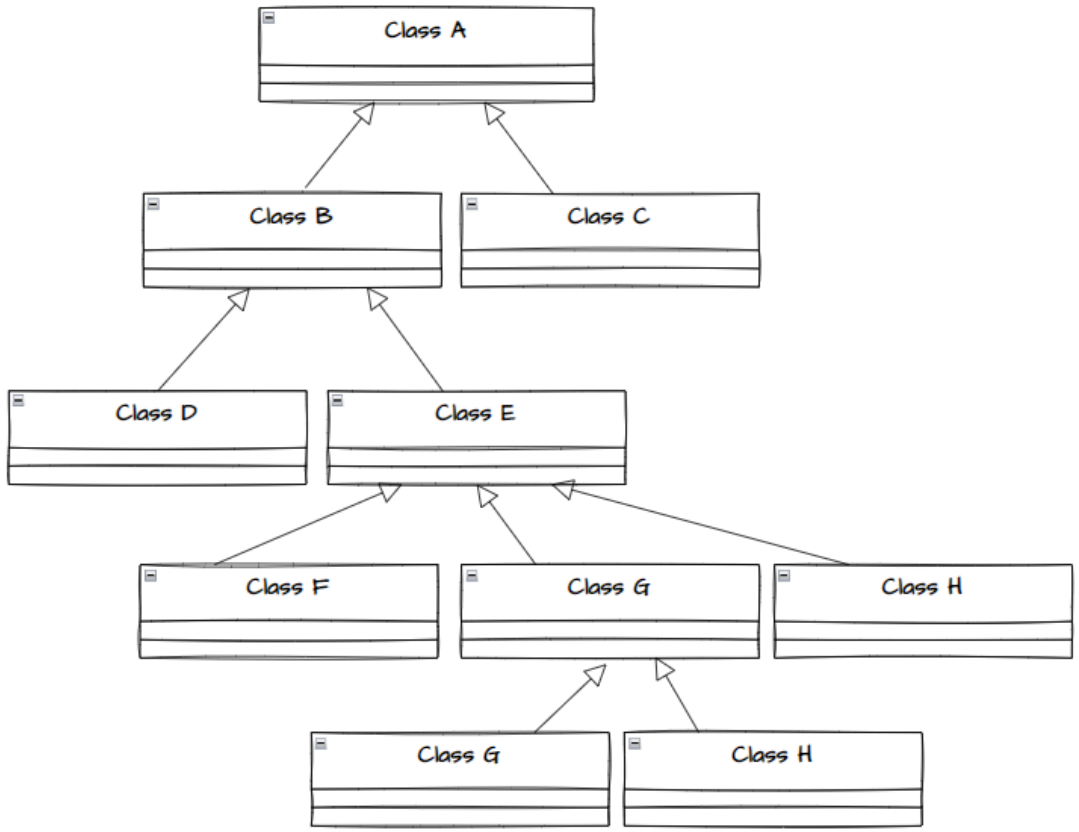
- 부모의 메서드를 Teacher 에서 또 만들 필요가 없음
- 즉, 자식 개체 상에서 부모의 메서드 호출 가능

부모 클래스의 메서드가 없다!

반대로, 상속을 사용하면 관리하기 불편한 경우를 보자

깊은 상속 관계

- Class B 를 바꾸면 그 아래 클래스도 모두 바뀜
- 자식 클래스에서 문제가 없는지 모두 확인해야 함
- 사실 컴포지션도 비슷한 문제가 있음
  - 상속보다는 덜함
  - 상속보다 조립성을 좀 더 강조했기 때문



1. 기계상의 차이 때문에 하나를 골라야할 때
2. 용도 때문에 상속을 고를 수 밖에 없을 때
3. 관리의 효율성을 고려할 때
4. **그외 일반적인 상황**

지금까지 봤던 세가지 경우는 어떤 걸 선택해야 하지 매우 명확했다.
하지만,
명확하지 않을 경우에는 어떻게 해야할까?

상식적으로 생각
그리고 공동체의 공통적인 규칙과 합의에 따라!

is-a 와 has-a 관계에 충실하자

…라고 이펙티브 자바에 적혀있다

> 계승은 하위 클래스가 상위 클래스의 하위 자료형(subtype)이 확실한 경우에만 바람직하다. 다시 말해서, 클래스 B는 클래스 A와 "IS-A" 관계가 성립할 때만 A를 계승해야 한다. 만일 A를 계승하고 싶다면, 스스로에게 물어보라. B는 확실히 A인가? 이 질문에 확실하게 "그렇다"고 답할 수 없다면, A를 계승하면 안 된다. 그 질문에 "아니다"라고 답했다면, B 안에 A 객체를 참조하는 private 필드를 두고, B에는 더 작고 간단한 API를 구현해야 한다. A는 B의 핵심적 부분이 아니며, B의 구현 세부사항에 불과하다.

이펙티브 자바 2판, 116페이지 부분발췌

- **is – a** 관계
  e.g.) **학생** 은 **사람** 이다
- **has – a** 관계
  e.g.) **대학** 은 **학생** 을 가지고 있다

컴포지션이 적절한 곳에 상속을 사용하면?

Hashtable 을 상속한 Properties

Class Properties

java.lang.Object
  java.util.Dictionary<K,V>
    java.util.Hashtable<Object,Object>
      java.util.Properties

All Implemented Interfaces:

Serializable, Cloneable, Map<Object,Object>

Direct Known Subclasses:

Provider

```
public class Properties
extends Hashtable<Object,Object>
```

특징

- Properties 클래스는 HashTable을 상속 받음
- HashTable을 상속받았기 때문에, Key 와 Value 를 갖는다
- HashMap 과 큰 차이가 없지만, Properties 클래스는 파일 입출력을 지원한다
- key=value 형식으로 작성된 파일을 key 와 value 로 나누어 저장할 때 유용

```
Properties p = new Properties();
p.getProperty("timeout");
p.get("timeout");
```

getProperty

```
public String getProperty(String key,
                          String defaultValue)
```

Searches for the property with the specified key in this property list. If the key is not found in this property list, the default property list, and its defaults, recursively, are then checked. The method returns the default value argument if the property is not found.

Parameters:

key - the hashtable key.

defaultValue - a default value.

Returns:

the value in this property list with the specified key value.

See Also:

setProperty(java.lang.String, java.lang.String), defaults

Properties 클래스의 함수

get

```
public V get(Object key)
```

Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.

More formally, if this map contains a mapping from a key k to a value v such that (key.equals(k)), then this method returns v; otherwise it returns null. (There can be at most one such mapping.)

Specified by:

get in interface Map<K,V>

Specified by:

get in class Dictionary<K,V>

Parameters:

key - the key whose associated value is to be returned

Returns:

the value to which the specified key is mapped, or null if this map contains no mapping for the key

Throws:

NullPointerException - if the specified key is null

See Also:

put(Object, Object)

Hashtable의 함수

참고 : 이펙티브 자바 2판

마지막으로 하고 싶은 말

- 시작했으면 끝을 보자
- ctrl + b 를 누르면 함수 구현부로 한번에 이동 가능하다
- 이북 사이트

둘 중에 하나만 골라, 상속 or 컴포지션

감사합니다