AJOU UNIVERSITY 1973

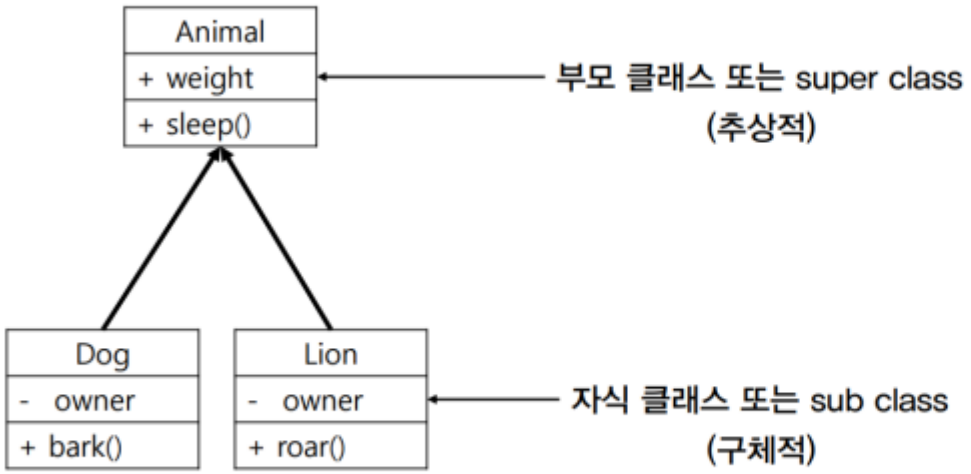
AJOU-MDC 국방디지털융합학과

객체지향 프로그래밍 및 실습

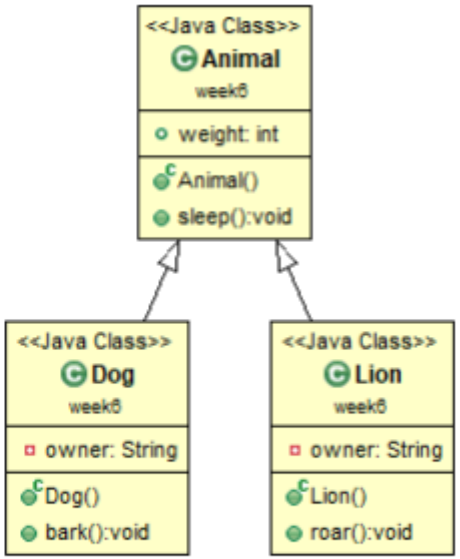
6주차. Inheritance

1. 상속

- Inheritance – 상속의 기본 개념
  - 객체 – (속성, 행동) 객체는 속성(attribute)과 행동(method)을 가지고 있다.
  - 상위 객체의 속성과 행동을 물려받을 수 있는 것



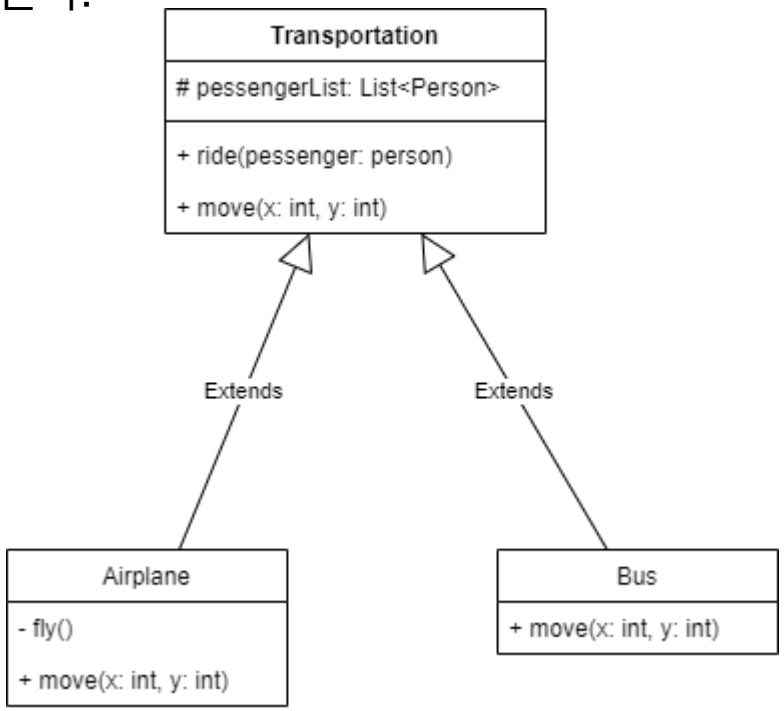
<상속의 개념도>



1. 상속

- Is-a relationship
  - Airplane is-a Transportation
  - Bus is-a Transportation
  - Airplane, Bus는 모두 Transportation이 할 수 있는 행동을 수행할 수 있다.
  - " 모두 Transportation이 가지는 속성을 가진다.

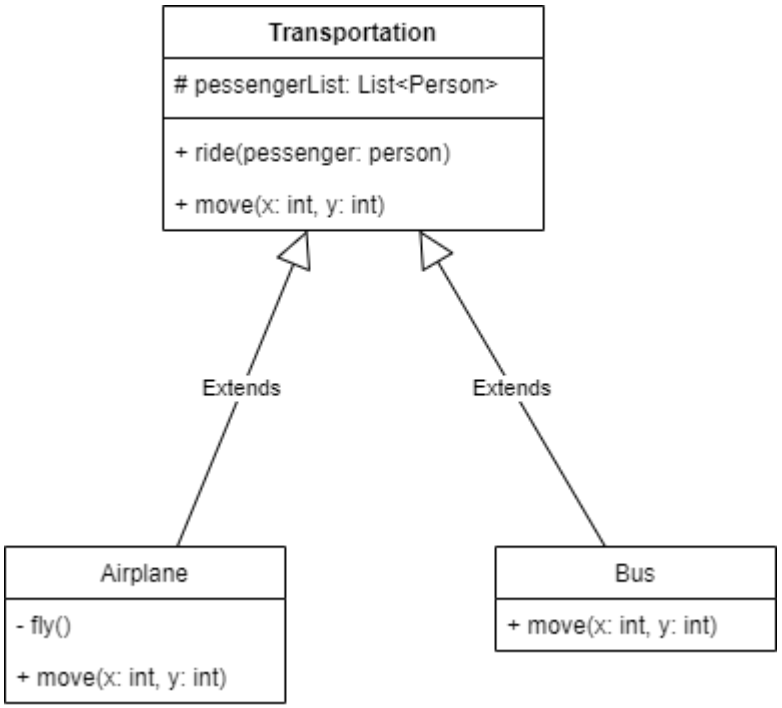
```
1 Person p1 = new Person("kim", 18);
2 Person p2 = new Person("jung", 20);
3 Person p3 = new Person("yun", 23);
4
5 Transportation airplane = new Airplane();
6 Transportation bus = new Bus();
7
8 airplane.ride(p3);
9 bus.ride(p2);
```



1. 상속

- Override
  - Transportation is moveable -> Airplane, Bus is moveable
  - 셋 다 move() 할 수 있지만, 그 방법은 구체적으로 다름
  - 비행기: 날아간다
  - 버스: 도로를 달린다

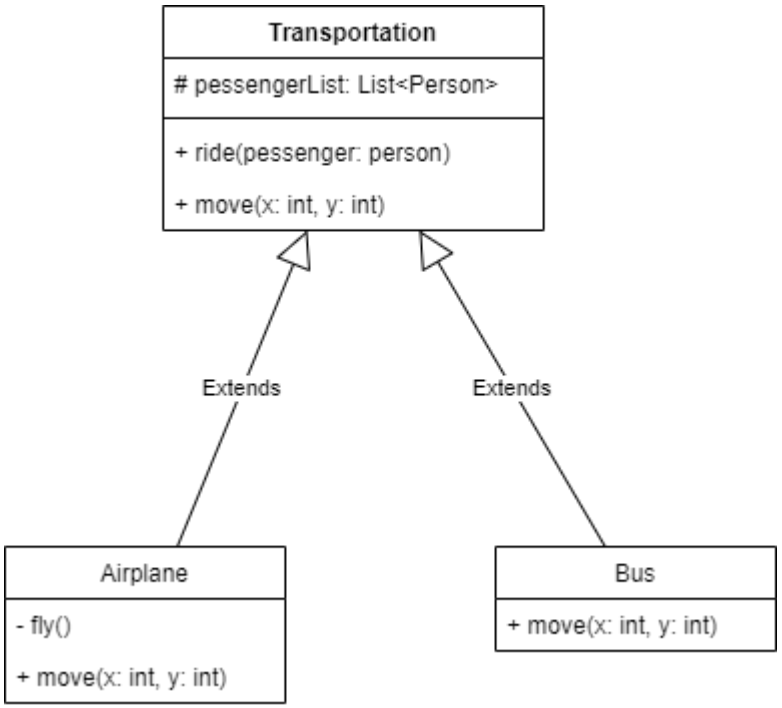
```java
package busairplane;

public class Bus extends Transportation {
    private void onTheRoad() {
        System.out.println("도로를 달린다");
    }

    @Override
    public void move(int x, int y) {
        onTheRoad();
        super.move(x, y);
    }
}
```



1. 상속

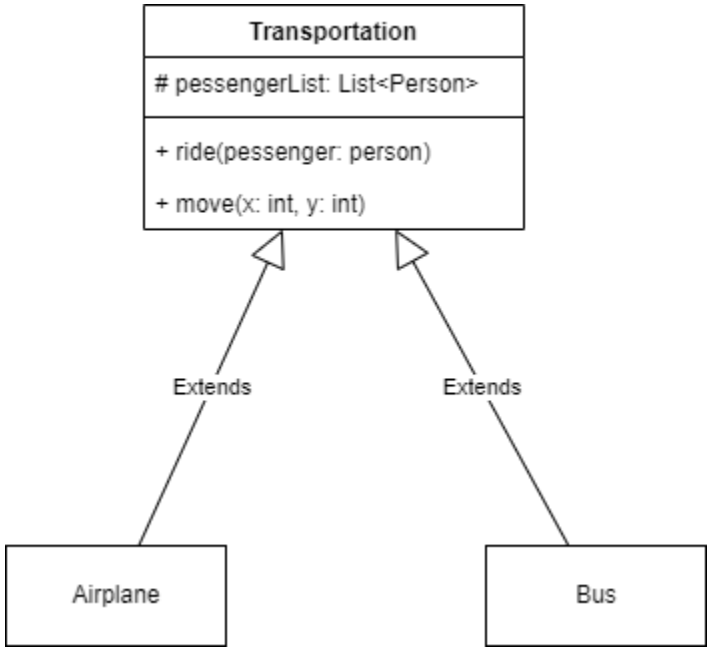
- Override
  - Transportation is moveable -> Airplane, Bus is moveable
  - 셋 다 move() 할 수 있지만, 그 방법은 구체적으로 다름
  - 비행기: 날아간다
  - 버스: 도로를 달린다

```java
package busairplane;

public class Airplane extends Transportation {
    private void fly() {
        System.out.println("비행 중");
    }

    @Override
    public void move(int x, int y) {
        fly();
        super.move(x, y);
    }
}
```



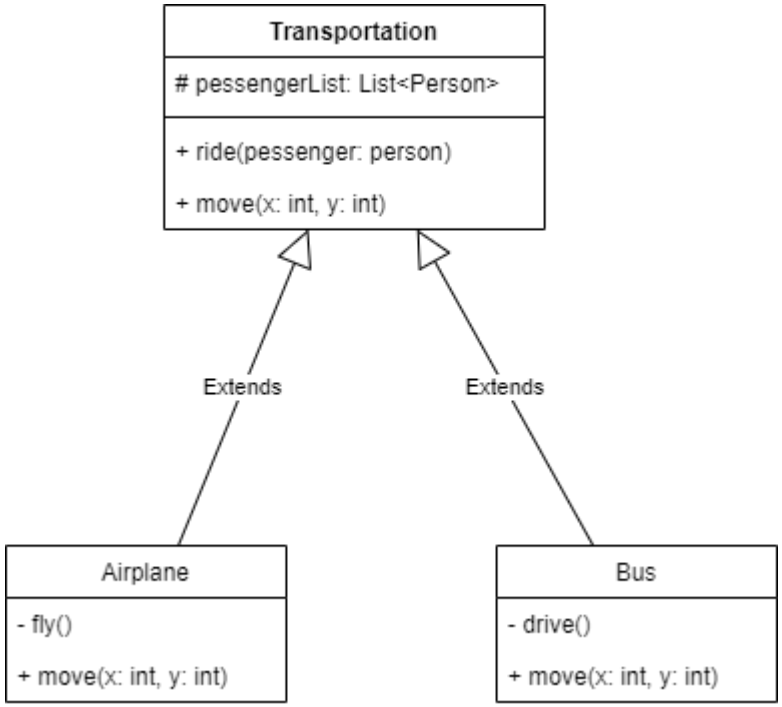
1. 상속

- **실습 문제 1**
  - 아래 Class Diagram을 참조하여 코드를 작성해보자



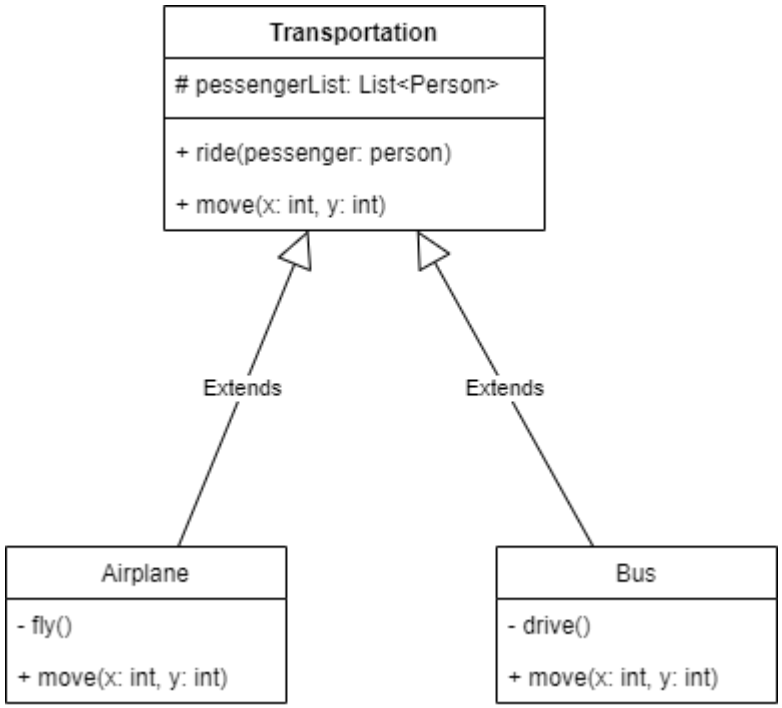
1. 상속

- **실습 문제 2**
  - 아래 Class Diagram을 참조하여 코드를 작성해보자
  - Override 적용



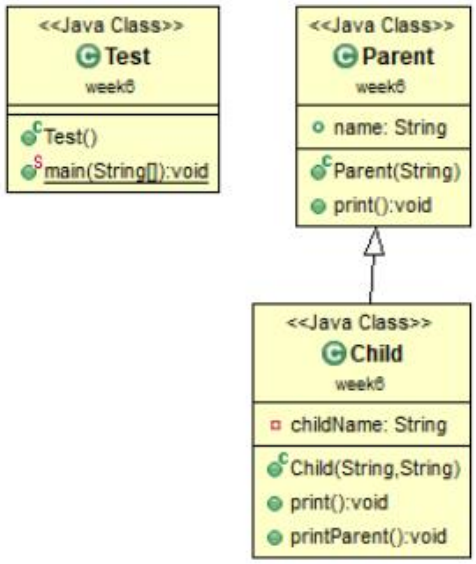
1. 상속

- **실습 문제 3**
  - TransportationTest 클래스를 생성하여 main메소드를 위치시킨다
  - Airplane클래스와 Bus 클래스의 메서드를 호출해본다
  - Eclipse code hint(ctrl+space) 기능을 통해 어떤 속성 또는 어떤 메서드에 접근 가능한지 테스트 해본다



1. 상속

- Super keyword
  - 하위 클래스에서 상위 클래스의 요소에 접근 할 수 있도록 해주는 키워드



Parent.java ▲

Child.java ▶

Test.java▼

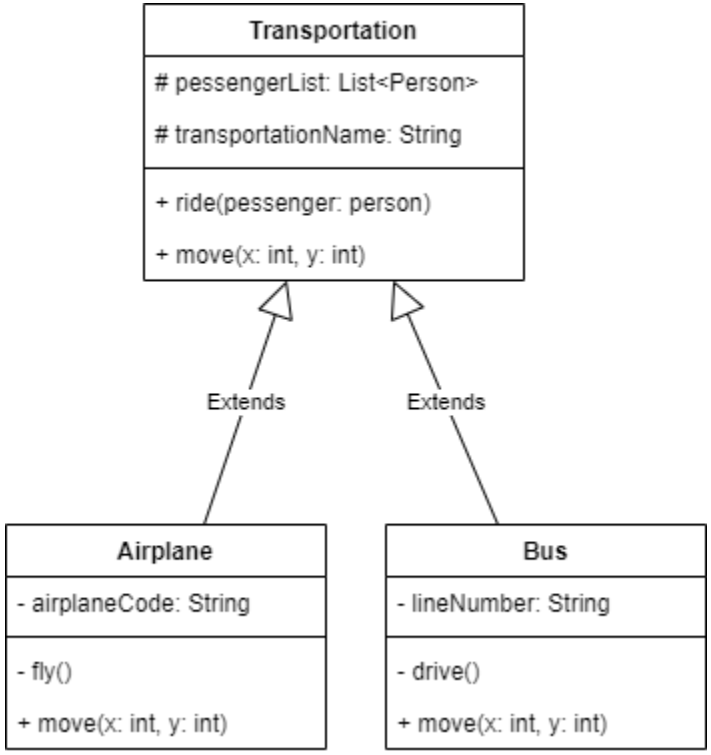
```java
public class Parent {
    public String name;

    public Parent(String name) {
        this.name = name;
    }

    public void print() {
        System.out.println("부모 클래스의 print() method");
        System.out.println("부모 이름: " + name);
    }
}
```

```java
public class Child extends Parent{
    private String childName;

    public Child(String name, String childName) {
        super(name);
        this.childName = childName;
    }

    @Override
    public void print() {
        System.out.println("자식 클래스의 print() method");
        System.out.println("자식 이름: " + childName);
    }

    public void printParent() {
        super.print();
    }
}
```

```java
public class Test {
    public static void main(String[] args) {
        // Mary의 자녀 Bob
        Child child = new Child("Mary", "Bob");
        child.print();
        child.printParent();
    }
}
```

1. 상속

- **실습 문제 4**
  - Transportation에서 필드를 초기화 할 수 있도록 생성자를 구성해본다.
  - Airplane, Bus에서 필드를 초기화 할 수 있도록 생성자를 구성해본다.



2. 상속 클래스 내에서의 접근제한자

- Protected member
  - Private member는 extends 하여도 접근 불가 – 상속받지 못함

```java
package protectedMember.a;

public class A {
    public String publicField;
    String defaultField;
    protected String protectedField;
    private String privateField;
}
```

```java
package protectedMember.a;

public class B extends A {
    public void accessTest() {
        super.publicField = "test";
        super.protectedField = "test";
        super.defaultField = "test";
        super.privateField = "test";
    }
}
```

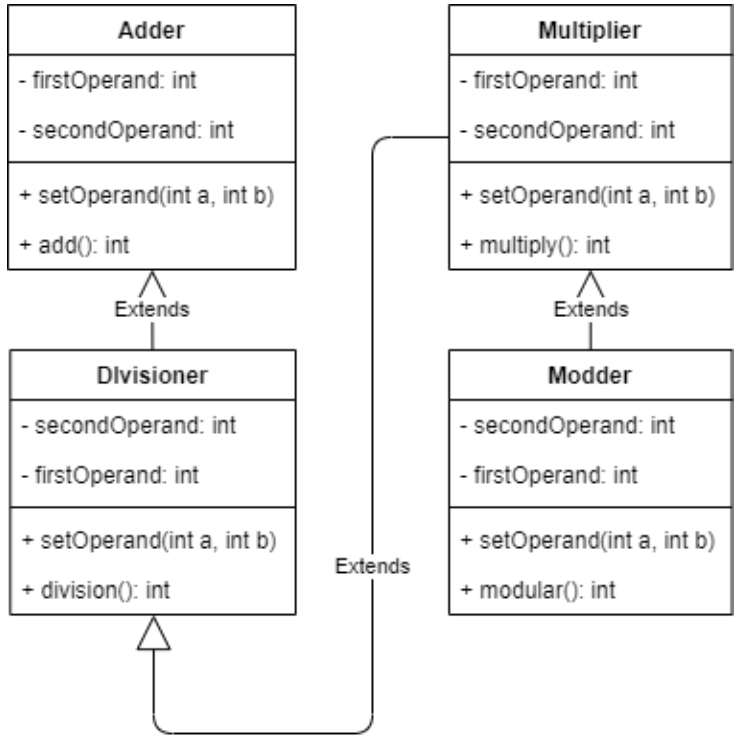
3. Composition vs Inheritance

- Sometimes composition is better than inheritance
  - Composition과 Inheritance는 결합도의 차이가 있다.
  - Composition is a far loosely coupling.
  - So it brings more flexibility and also allows us to change runtime behavior
  - Inheritance is known as the tightest form of coupling
  - 둘의 차이를 명확히 인식하고 Inheritance를 남발하지 않는 것이 좋다.

3. Composition vs Inheritance

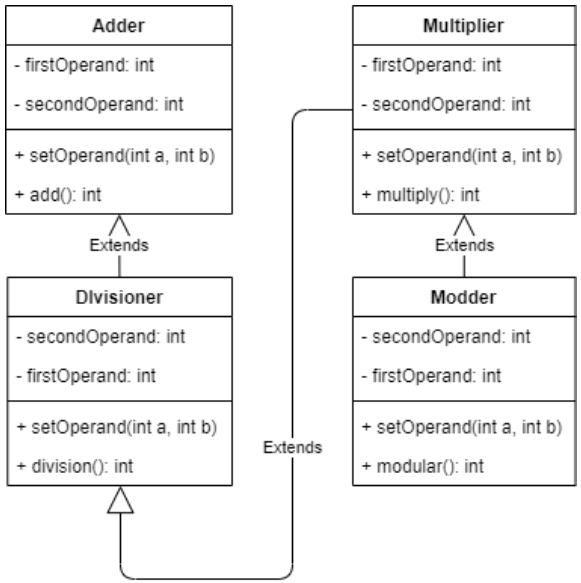
- Inheritance BAD USECASE: Composition이 더 좋은 경우
- **실습 문제 5**
  - 아래 클래스 다이어그램을 참조하여 계산기 프로그램을 제작해보자
  - add: a+b
  - division: a/b
  - multiply: a*b
  - modulor: a%b

  - CalcTest클래스에 main메소드를 생성하여
    덧셈, 뺄셈, 곱셈, 모듈러 연산을 테스트해본다.



3. Composition vs Inheritance

- Inheritance BAD USECASE: Composition이 더 좋은 경우
  - 오른쪽 구조에서는 외부 이용자 관점에서 어느 클래스를 인스턴스화 해야하는지 잘 알 수 없다.
    - Divisioner를 인스턴스화시키면 Multiplier와 Modder는 쓸 수 없다
    - Divisioner is-a Adder ???
  - 수정의 불편함
    - 상속관계가 조금이라도 바뀌면 기존의 코드가 무용지물 될 가능성이 있다.
    - 기능 추가가 불편하다



3. Composition vs Inheritance

- Inheritance BAD USECASE: Composition이 더 좋은 경우
  - Composition으로 바꾼다면…?
- **실습 문제 6**
  - 모두 Has-a 관계로 바꿔보자
  - Test클래스의 main메소드에서 직접적으로 Adder등의 클래스를 인스턴스화 시키지 않도록 한다.
  - CalcCompositionTest클래스에 main메소드를 생성하여 덧셈, 뺄셈, 곱셈, 모듈러 연산을 테스트해본다.