Object – Oriented Programming

LAB #7. Inheritance

Inheritance

- 상속
  - 다른 클래스에서 새로운 클래스를 만드는 과정
  - 원래의 클래스는 base 클래스, Parent 클래스 또는 Super 클래스라고 한다.
  - 새로운 클래스를 derived(파생) 클래스 또는 sub 클래스라고 한다.
- Subclass
  - Super 클래스에서 method와 instance variable을 상속 받음
  - method와 instance variable 추가로 선언 가능

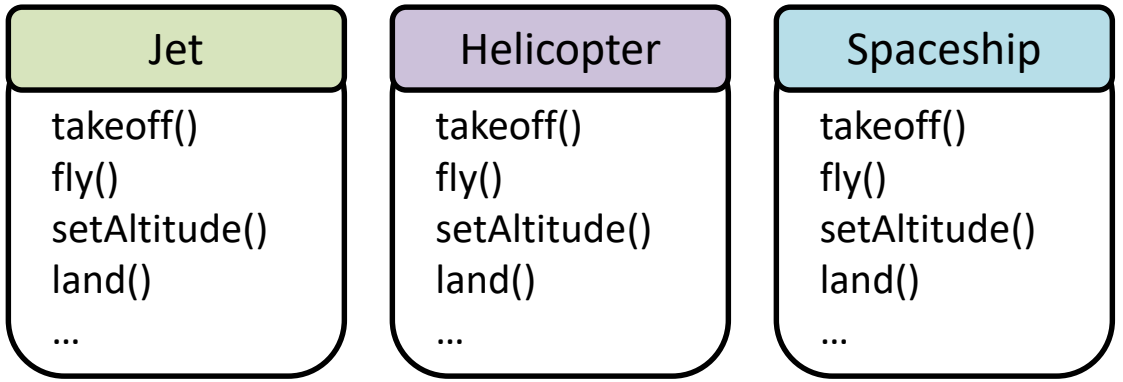
Inheritance

- 탈 것의 종류
  - 비행기
    ◎ 제트기, 헬리콥터, 우주선
  - 자동차
    ◎ 차, 트럭, 오토바이
  - 기차
    ◎ 디젤기차, 전동기차, 모노레일
  - 배
    ◎ 어업선, 유람선, 유조선

- 각각의 탈것 종류에 대해 클래스가 작성

Inheritance

- 비행기 타입의 샘플 코드
  - ◎ takeoff() 이륙
  - ◎ fly() 비행
  - ◎ setAltitude() 고도 설정
  - ◎ land() 착륙



- 이 코드는 대부분의 비행기에 사용될 것이다.
  - 공통점이 많다.
  - 비행기 타입마다 각각의 method를 작성하는 것은 낭비이다.
    * 하나를 변경하려면, 많은 method를 모두 변경해야 한다.

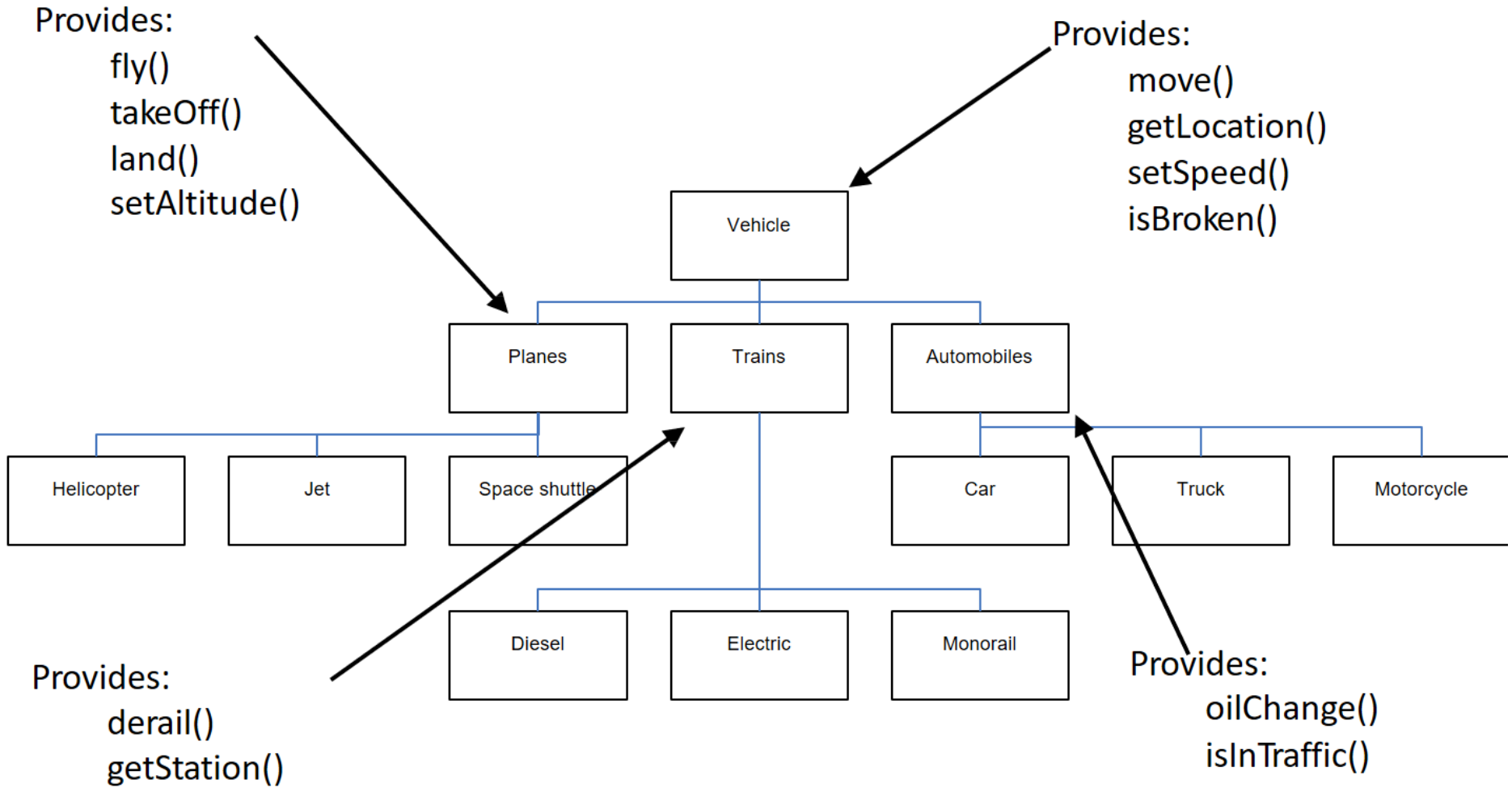
Inheritance

- 실제로, 모든 탈 것에는 유사한 방법이 있을 것이다.
  - ◎ move()
  - ◎ getLocation()
  - ◎ setSpeed()
  - ◎ isBroken()
- 이 코드는 모든 탈 것의 종류에 공통적으로 사용
  - 각 탈 것 종류에 대해 각각의 move() method를 작성하는 것은 낭비

- 하나의 move() method를 지정하고, 각 탈 것의 종류에서 그 코드를 상속 받도록 하면??

  -> 변경해야 할 일이 있을 때, 하나의 코드만 변경하면 된다.

Inheritance



Inheritance

- ‘Parent’ 클래스와 ‘Child’ 클래스를 생성

- ‘Child’ 클래스는 ‘Parent’ 클래스로부터 method를 상속 받는다.

- 일부 클래스는 Sub 클래스이자 Super 클래스이다.

Overriding

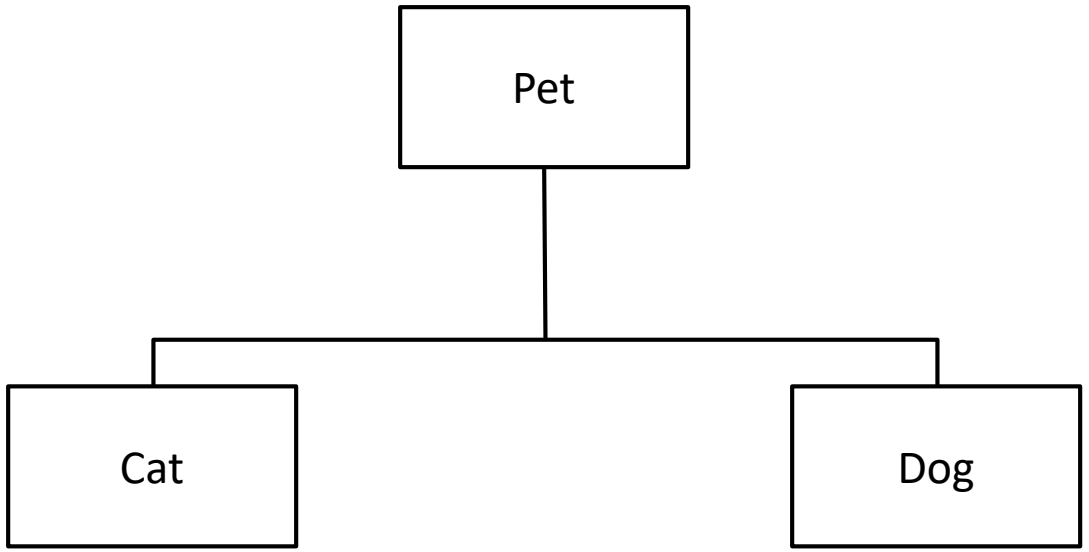
- 상속된 메소드의 정의를 변경하는 작업

- 클래스를 작성할 때마다 메소드를 재정의 할 수 있다.

    ◎ equals(Object obj)

    ◎ toString()

Overriding



Overriding

```
class Pet {
        private String name;
        public String getName() {
                return name;
        }
        public void setName(String petName) {
                name = petName;
        }
        public String speak( ) {
                return "I'm your cuddly little pet.";
        }
}
```

Overriding

```
class Cat extends Pet {
    public String speak( ) {
        return "Don't give me orders.\n" +
               "I speak only when I want to.";
    }
}
```

Cat subclass가 상속받은 speak 메소드를 **override**

```
class Dog extends Pet {
    public String fetch( ) {
        return "Yes, master. Fetch I will.";
    }
}
```

Dog subclass 에서 새로운 메소드 fetch 를 추가

Overriding

```
Dog myDog = new Dog();

System.out.println(myDog.speak());
System.out.println(myDog.fetch());
```

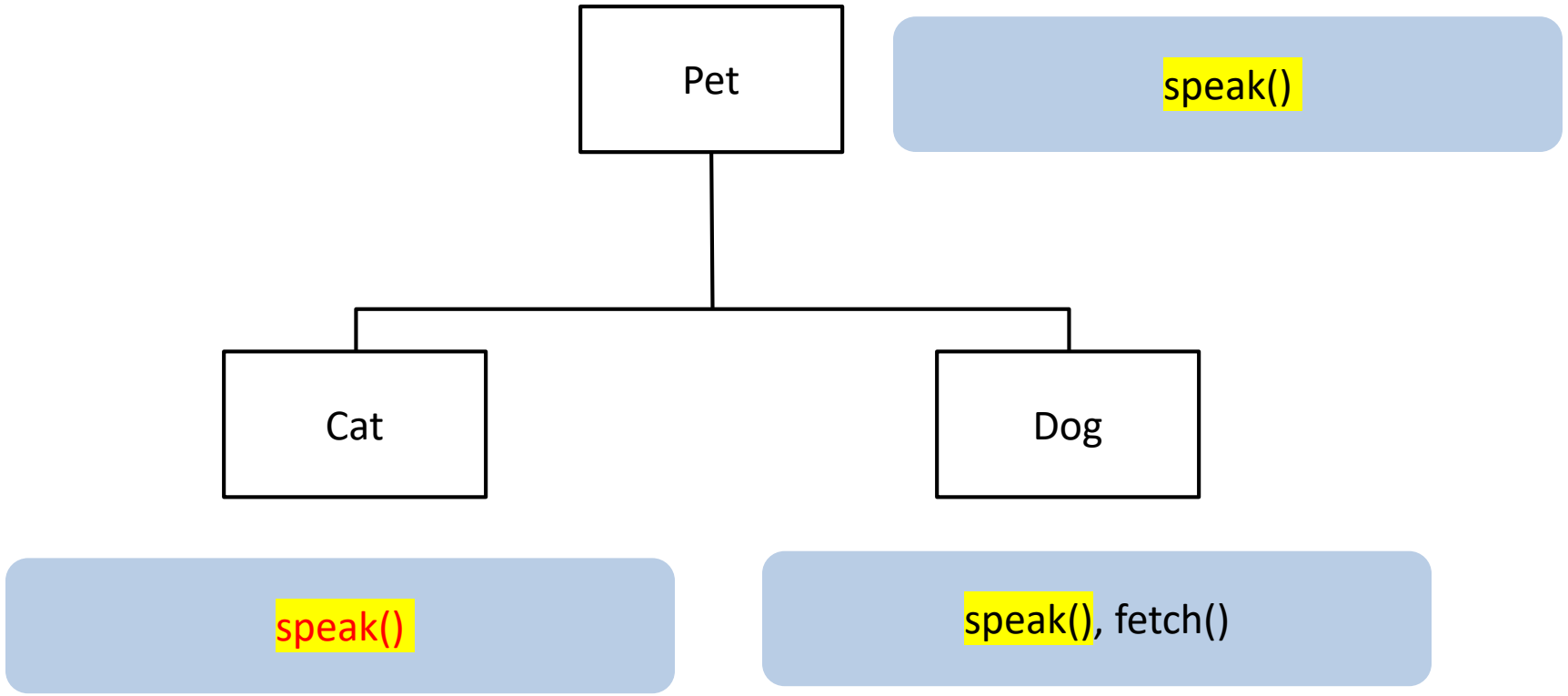
```
I'm your cuddly little pet.
Yes, master. Fetch I will.
```

```
Cat myCat = new Cat();

System.out.println(myCat.speak());
System.out.println(myCat.fetch());
```

```
Don't give me orders.
I speak only when I want to.
```

ERROR

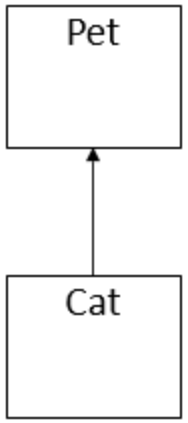
Overriding



Overriding

- 메소드를 override 하려면

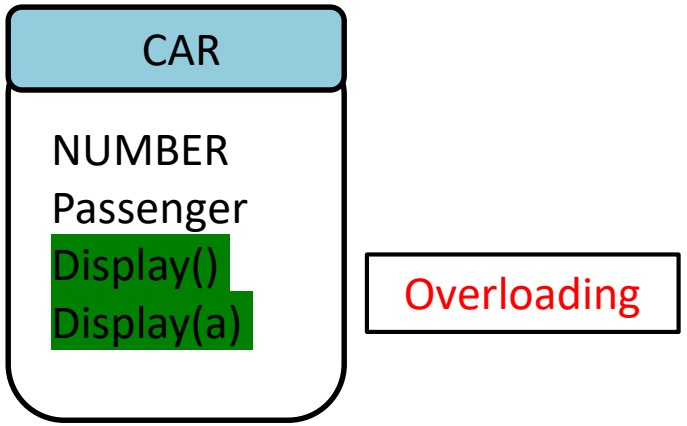
  -> Parent 클래스의 메소드와 **같은 헤더**를 사용하여 Sub 클래스에 메소드를 생성한다.



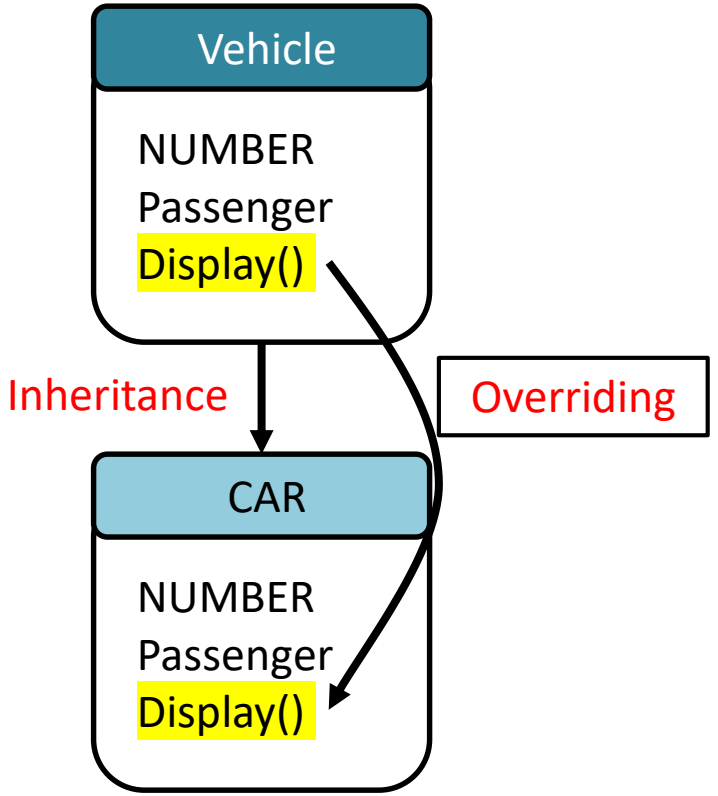

```
public String speak( ) {
    return "I'm your cuddly little pet.";
}

public String speak( ) {
    return "Don't give me orders.\n" +
           "I speak only when I want to.";
}
```

Overloading, Overriding

Overloading



Overriding



final Modifier

- 상속을 방지한다.
- final 클래스는 다른 클래스에서 상속 불가
- final 메소드는 subclass에서 재정의 불가

super

- 부모 클래스를 가리키는 키워드

- super() 은 부모 클래스의 생성자를 의미

- 부모의 생성자를 임의로 호출하지 않으면, 부모 클래스의 기본 생성자가 자동 호출됨

- 부모의 생성자는 생성자에서 가장 먼저 호출되어야 함

super

```
public class Person {
    public String name;
    public String gender;
    public String phoneNumber;

    public Person(String name, String gender, String phoneNumber) {
        this.name = name;
        this.gender = gender;
        this.phoneNumber = phoneNumber;
    }

    public Person() {
        name = "Lee";
        gender = "man";
        phoneNumber = "010-1234-5678";
    }
}
```

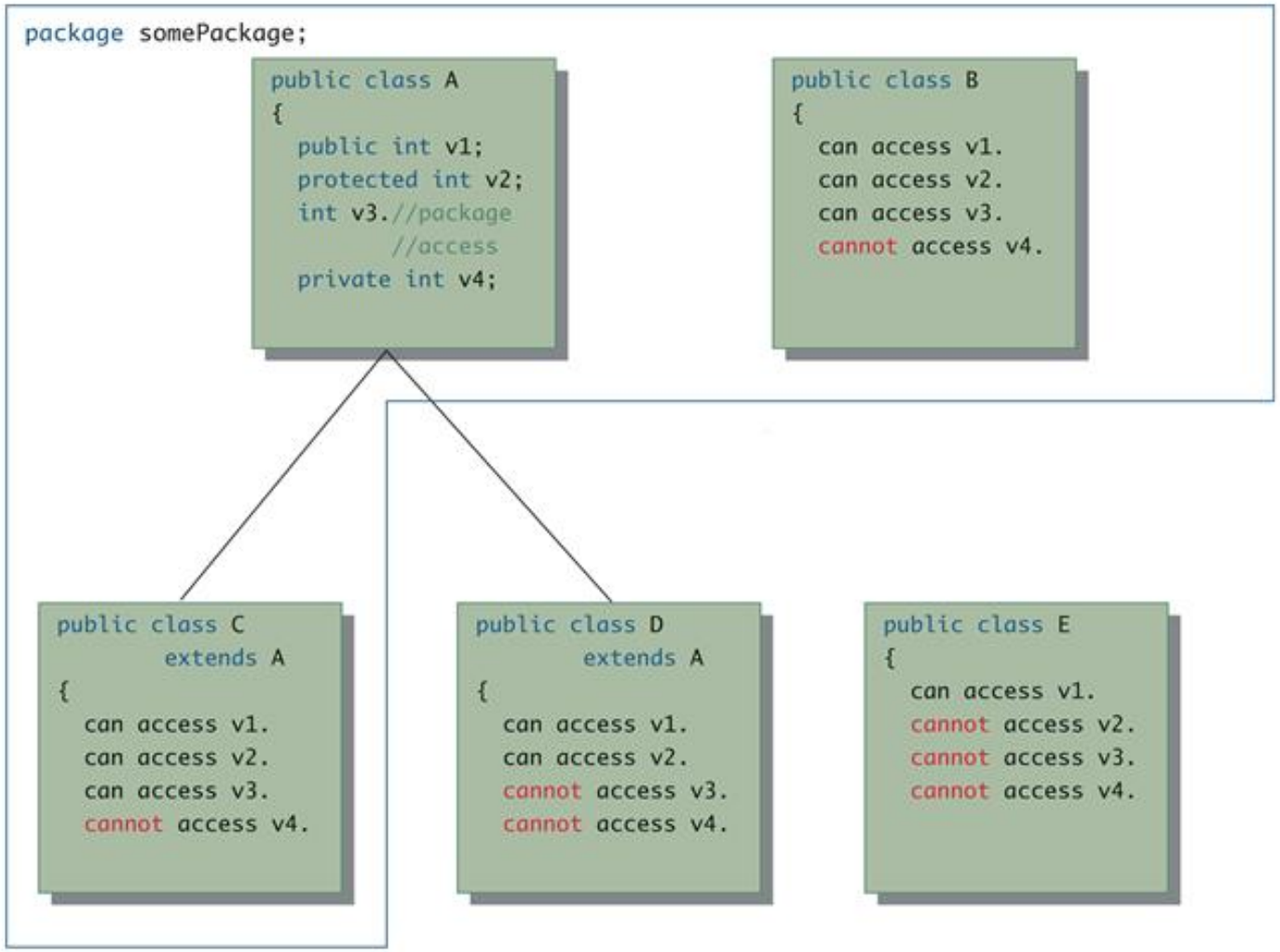
super

```
public class Student extends Person {
    public int studentId;
    public String university;

    public Student(String name, String gender,
                   String phoneNumber, int studentId, String university) {
        super(name, gender, phoneNumber);
        this.studentId = studentId;
        this.university = university;
    }

    public Student() {
        this("Lee", "man", "010-1234-5678", 2020012345, "Hanyang");
    }
}
```

Class access rights

Modifier	Class	Package	Subclass	World
public	O	O	O	O
protected	O	O	O	
no modifier	O	O		
private	O			

Class access rights



- 부모 클래스의 private 멤버를 직접 접근 불가
  - ➢ getter와 setter를 통해서 접근

Class access rights

```
public class A {
    public int v1;
    protected int v2;
    int v3;
    private int v4;

    public A() {
        v1 = 1;
        v2 = 2;
        v3 = 3;
        v4 = 4;
    }
}
```

Object Class

- Java는 Object라는 java.lang 패키지에 특별한 클래스를 갖고 있음

- Java의 모든 클래스는 Object 클래스에서 상속됨

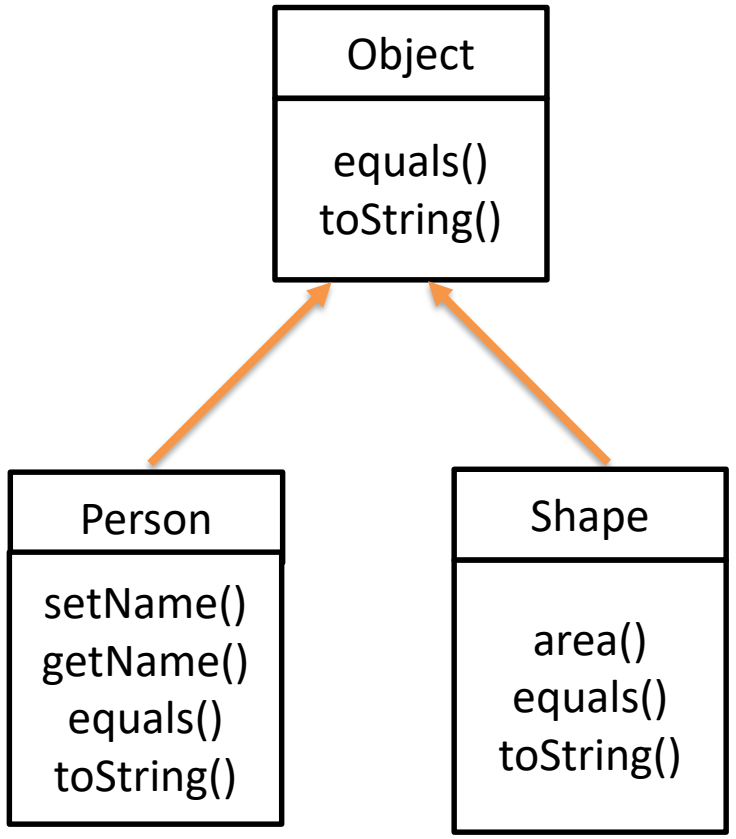
  -> 모든 클래스는 객체라는 것을 의미

- 클래스는 Object이기 때문에 Object type의 매개변수를 받기 위한

  특정 메소드가 작성 될 수 있음

Object Class

- Object 클래스에는 모든 Java 클래스에서 상속하는 메소드가 있다.
  - equals(Object obj)
  - toString()



Better equals Method

- equals 메소드는 Object 클래스에서 상속 받는다.

- equals 메소드를 override 하는 경우, Object를 매개변수로 받아야한다.

- obj는 캐스팅을 통해 비교됨 -> 비교하고자 하는 객체의 type으로 캐스팅

- equals(Object obj) 에서는

  ◎ obj가 null인지 확인
  ◎ class type이 일치하는지 확인

  두 과정을 거친 후 같은지 확인하려는 값을 비교한다.

Better equals Method

```
public boolean equals(Object obj) {
    if(obj == null)
        return false;
    else if(getClass( ) != obj.getClass( ))
        return false;
    else {
        Person person = (Person)obj;
        return  name.equals(person.name) && gender.equals(person.gender)
                && phoneNumber.equals(person.phoneNumber);
    }
}
```

getClass() Method

- 모든 객체는 Object 클래스에서 동일한 getClass() 메소드를 상속 받음

  * getClass() 메소드는 final로 정의되어 있어 재정의 할 수 없음

- 객체의 getClass() 호출은 해당 객체의 Class를 return

  -> 두 호출의 결과를 == 또는 != 로 비교하여 정확하게 동일한 클래스를 나타내는지 알 수 있음

```
(object1.getClass() == object2.getClass())
```

Instanceof vs getClass()

- instanceof는 자손 클래스의 객체에도 true 반환

```
Person p1 = new Person();
Person s1 = new Student();
Student s2 = new Student();

System.out.println(p1 instanceof Person);
System.out.println(p1 instanceof Student);

System.out.println(s1 instanceof Person);
System.out.println(s1 instanceof Student);

System.out.println(s2 instanceof Person);
System.out.println(s2 instanceof Student);

System.out.println(p1.getClass() == s1.getClass());
System.out.println(s1.getClass() == s2.getClass());
```

StringTokenizer

```
String str = "A B C D";
StringTokenizer st1 = new StringTokenizer(str);

while(st1.hasMoreTokens()) {
    System.out.println(st1.nextToken());
    System.out.println(st1.countTokens());
}
```

```
A
3
B
2
C
1
D
0
```

공백을 기준으로 하는 tokenizer 생성자

Token이 남아있는지 여부

다음 token return

남아있는 token의 수 return

과제

- Employee, Manager, Engineer Class를 생성한다.
  - Employee Class 에는 3개의 private instance variable이 있다.
    ◎ String name
    ◎ int employeeNum
    ◎ String department
  - name, employeeNum, department를 인자로 받는 생성자를 작성한다.
  - 각각 instance variable의 getter와 setter를 작성한다. (public)
  - name 과 employeeNum 이 같은 경우 true, 다른 경우 false 를 반환하는 equals(Object obj) 메소드를 작성한다.
  - Name과 employeeNum을 반환하는 toString() 메소드를 작성한다.
    - 형식 :
      Name : [name]
      Emp# : [employeeNum]

과제

- Manager Class는 Employee Class를 extends한다.
  - Manager Class에는 2개의 private instance variable이 있다.

    ◎ int officeNum
    ◎ String team

  - name, employeeNum, officeNum, team을 인자로 받는 Manager 객체의 생성자를 작성한다.

    -> Department를 "Management" 로 설정

  - Manager의 name과 employeeNum, location 및 하는 일을 반환하는 toString 메소드를 만든다.

    이때, location은 department 및 officeNum이다.

    - 형식 :
      Name : [name]
      Emp# : [employeeNum]
      location : [department], office : [officeNum]

  - name, employeeNum, officeNum이 동일할 경우 true, 다른 경우 false를 반환하는 equals 메소드를 작성

과제

- Engineer Class는 Employee Class를 extends한다.
  - Engineer Class는 2개의 private instance variable이 있다.

    ◎ String workZone
    ◎ String project

  - name, employeeNum, workZone, project 을 갖는 생성자를 작성한다.

    -> Department를 "Engineering" 설정

  - Engineer의 name, employeeNum, workZone이 동일하다면 true, 다르면 false를 반환하는 equals 메소드를 작성한다.

  - Engineer의 name, employeeNum, location, workZone을 반환하는 toString을 작성한다.
    (형식 : Name : [name]₩nEmp# : [employeeNum]₩nlocation : [department], zone : [workZone])

과제 제출

- Employee, Manager, Engineer Class를 제출

과제 문의: lhs9394@naver.com (조교 이효식)