Power Java

# 제11장 상속



# 이번 장에서 학습할 내용

- 상속이란?
- 상속의 사용
- 메소드 재정의
- 접근 지정자
- 상속과 생성자
- Object 클래스
- 종단 클래스

# 상속이란?

- 상속의 개념은 현실 세계에도 존재한다.


그림 11.1 상속의 개념



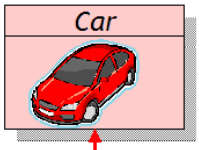
# 상속의 장점

- 상속의 장점
  - 상속을 통하여 기존 클래스의 필드와 메소드를 재사용
  - 기존 클래스의 일부 변경도 가능
  - 상속을 이용하게 되면 복잡한 GUI 프로그램을 순식간에 작성

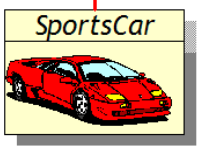
  - 상속은 이미 작성된 검증된 소프트웨어를 재사용
  - 신뢰성 있는 소프트웨어를 손쉽게 개발, 유지 보수
  - 코드의 중복을 줄일 수 있다.

# 상속

```
class Car
{
        int speed;
}
class SportsCar extends Car
{
        int turbo;
}
```




수퍼 클래스(superclass)

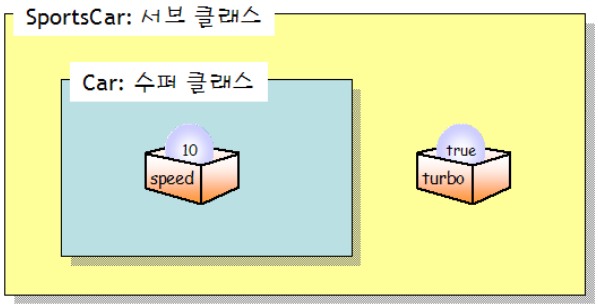



서브클래스(subclass)

상속한다는 의미



# 수퍼 클래스는 서브 클래스를 포함



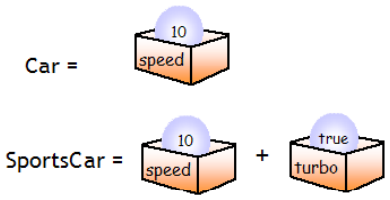




그림 11.3 수퍼 클래스는 서브 클래스를 포함한다.

# 상속의 예

| 수퍼 클래스 | 서브 클래스 |
|---|---|
| Animal(동물) | Lion(사자), Dog(개), Cat(고양이) |
| Bike(자전거) | MountainBike(산악자전거) |
| Vehicle(탈것) | Car(자동차), Bus(버스), Truck(트럭), Boat(보트), Motocycle(오토바이), Bicycle(자전거) |
| Student(학생) | GraduateStudent(대학원생), UnderGraduate(학부생) |
| Employee(직원) | Manager(관리자) |
| Shape(도형) | Rectangle(사각형), Triangle(삼각형), Circle(원) |

참고

- 수퍼 클래스 == 부모 클래스(parent class) == 베이스 클래스(base class)
- 서브 클래스 == 자식 클래스(child class) == 파생된 클래스(derived class)



# 상속의 계층 구조

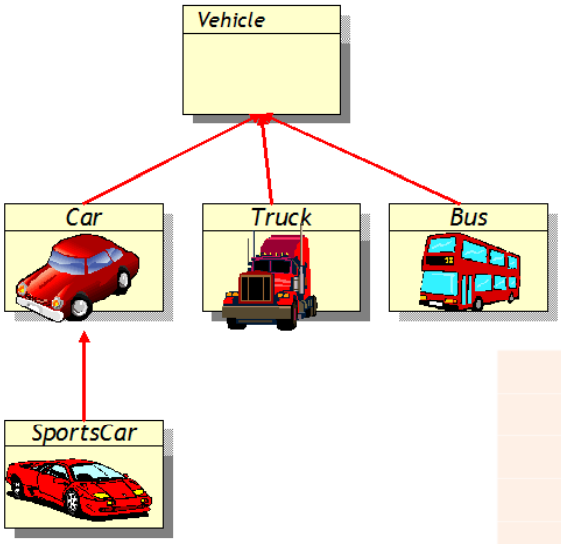



그림 11.4 상속 계층 구조도

```
class Vehicle { ... }
class Car extends Vehicle { ... }
class Truck extends Vehicle { ... }
class Bus extends Vehicle { ... }
class SportsCar extends Car { ... }
```

# 상속은 중복을 줄인다.

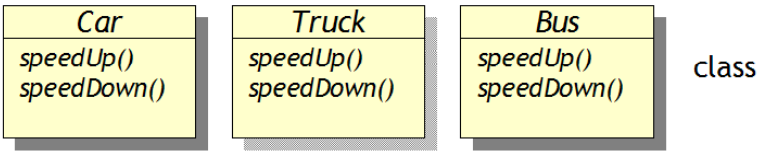
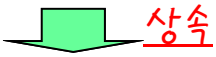



그림 11.6 각 클래스에 코드가 중복된다.



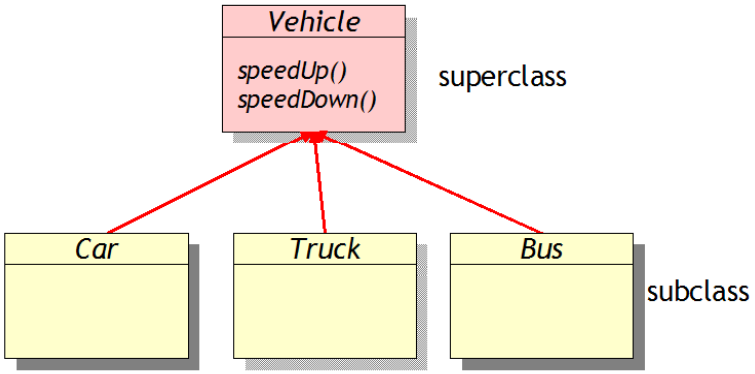




그림 11.7 중복되는 코드는 수퍼 클래스에 모은다.



# 중간 점검 문제

1. 사자, 호랑이, 개, 고양이, 여우, 악어, 도마뱀들을 상속 계층 구조를 이용하여 표현하여 보자.

# 상속의 구체적인 예

```
public class Car {
        // 3개의 필드 선언
        public int speed; // 속도
        public int gear; // 주행거리
        public String color; // 색상

        // 3개의 메소드 선언
        public void setGear(int newGear) { // 기어 설정 메소드
                gear = newGear;
        }
        public void speedUp(int increment) { // 속도 증가 메소드
                speed += increment;
        }
        public void speedDown(int decrement) { // 속도 감소 메소드
                speed -= decrement;
        }
}
```



# 상속의 예

```
class SportsCar extends Car {          // Car를 상속받는다.
        // 1개의 필드를 추가
        boolean turbo;

        // 1개의 메소드를 추가
        public void setTurbo(boolean newValue) { // 터보 모드 설정 메소드
                turbo = newValue;
        }
};
```

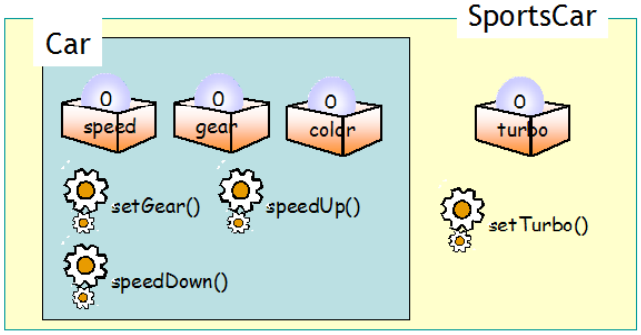



그림 11.8 Car와 SportsCar

# 상속의 사용

- 서브 클래스는 수퍼 클래스의 필드와 메소드를 마치 자기 것처럼 사용할 수 있다.

```
public class Test {
    public static void main(String[] args) {
        SportsCar  c = new SportsCar();
        c.color = "Red";                    // 수퍼 클래스 필드 접근
        c.setGear(3);           // 수퍼 클래스 메소드 호출
        c.speedUp(100);                     // 수퍼 클래스 메소드 호출
        c.speedDown(30);                    // 수퍼 클래스 메소드 호출
        c.setTurbo(true);                   // 자체 메소드 호출
    }
}
```



# 메소드 재정의

- 메소드 재정의(method overriding): 서브 클래스가 필요에 따라 상속된 메소드를 다시 정의하는 것

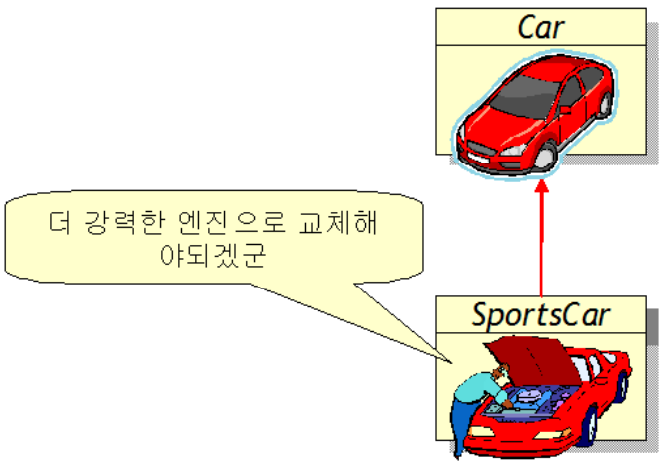
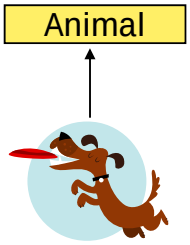



그림 11.9 메소드 재정의

# 메소드 재정의의 예

```
public class Animal {
    public void makeSound()
    {
            // 아직 특정한 동물이 지정되지 않았으므로 몸체는 비어 있다.
    }
};
```

```
public class Dog extends Animal {
    public void makeSound()
    {
            System.out.println("멍멍!“);
    }
};
```






# 메소드를 재정의하려면

- 메소드의 이름, 반환형, 매개 변수의 개수와 데이터 타입이 일치하여야 한다.

```
public class Animal {
            public void makeSound()
            {
            }
};
```

오버라이드가 아님

```
public class Dog extends Animal {
            public int makeSound()
            {
            }
};
```

# 중복 정의와 재정의

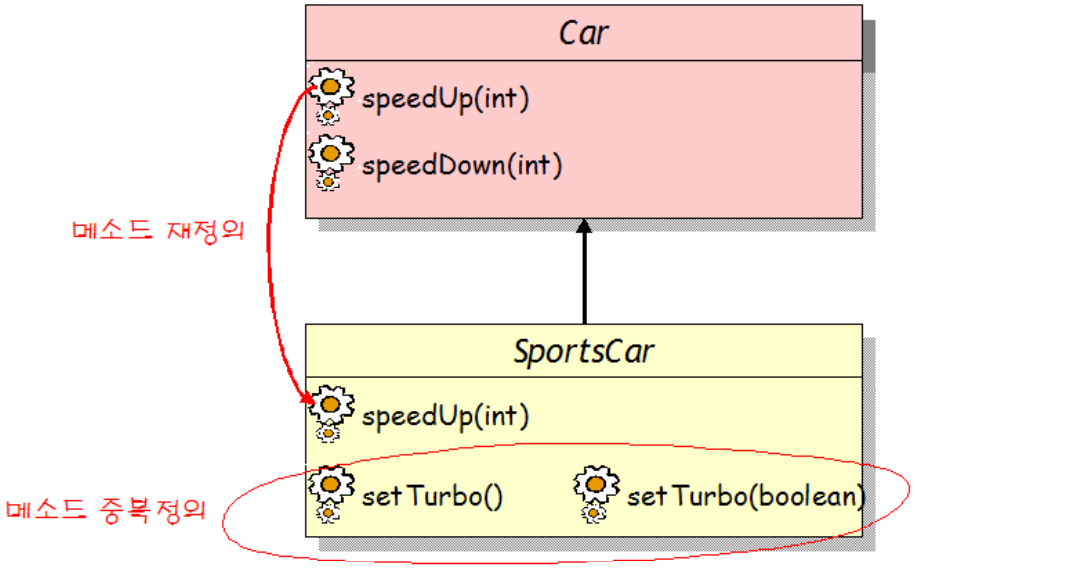



그림 11.10 메소드 재정의와 메소드 중복 정의



# super 키워드

```
class ParentClass {
    int data=100;
    public void print() {
        System.out.println("수퍼 클래스의 print() 메소드");
    }
}
public class ChildClass extends ParentClass {
    int data=200;
    public void print() { //메소드 재정의
        super.print();
        System.out.println("서브 클래스의 print() 메소드 ");
        System.out.println(data);
        System.out.println(this.data);
        System.out.println(super.data);
    }
    public static void main(String[] args) {
            ChildClass obj = new ChildClass();
            obj.print();
    }
}
```

```
수퍼 클래스의 print() 메소드
서브 클래스의 print() 메소드
200
200
100
```

수퍼클래스 객체를 가리킨다.

# 접근 지정자

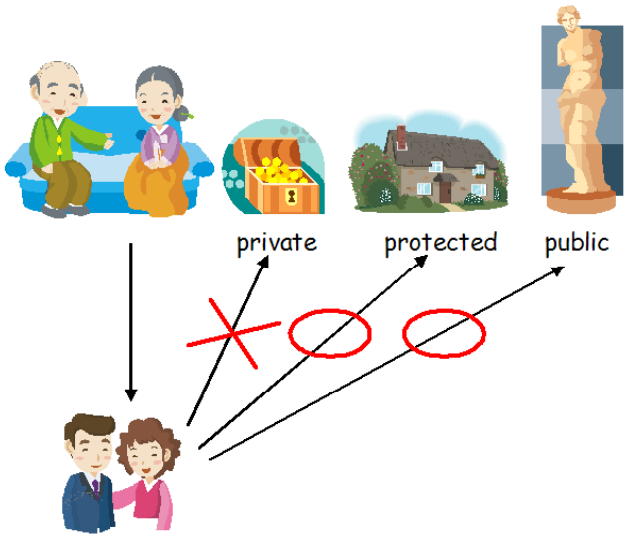



그림 11.12 상속에서의 접근 지정자



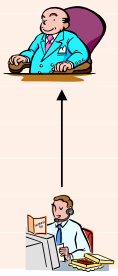
# 접근 지정자의 예

*Employee.java*

```
public class  Employee {
    public String name;       // 이름
    private int RRN;           // Regident Resgistration Number: 주민등록번호
    protected int salary;      // 월급

    protected int getSalary() {
        return salary;
    }
    protected void setSalary(int salary) {
        this.salary = salary;
    }

};
```



# 접근 지정자의 예

*Manager.java*

```
public class Manager extends Employee {
    private int bonus;

    public int getSalary() {          // 메소드의 가시성을 증가시키는 것은 가능하다.
        return salary + bonus;        // protected 멤버인 salary는 접근 가능
    }
    private void setSalary(int salary) { // 오류! 메소드의 가시성을 줄이면 안된다.
        super.salary = salary;
    }
    public void setRRN(int rrn) {
            RRN = rrn;                // 오류! private는 서브 클래스에서 접근 못함!
    }
};
```





# 상속과 생성자

```
class Shape {
        public Shape(String msg) {
            System.out.println("Shape 생성자() " + msg);
        }
};

public class Rectangle extends Shape {
        public Rectangle(){
            super("from Rectangle"); // 명시적인 호출
            System.out.println("Rectangle 생성자()");
        }

        public static void main(String[] args) {
            Rectangle r = new Rectangle();
        }

};
```

```
Shape 생성자 from Rectangle
Rectangle 생성자
```

# 묵시적인 호출

```
class Shape {
        public Shape() {
                System.out.println("Shape 생성자()");
        }
};

public class Rectangle extends Shape {
        public Rectangle() {

                System.out.println("Rectangle 생성자()");
        }

        public static void main(String[] args) {
                Rectangle r = new Rectangle();
        }
};
```

컴파일러가 Shape();을 자동적으로 넣어준다고 생각하라.

```
Shape 생성자
Rectangle 생성자
```



# Object 클래스

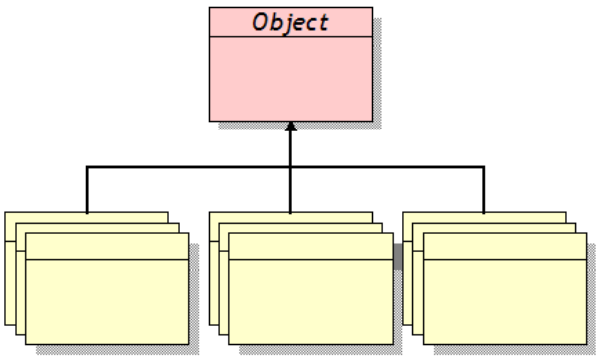



그림 11.13 Object 클래스는 상속 계층 구조의 맨 위에 있다.

# Object의 메소드

- **protected Object** clone() : 객체 자신의 복사본을 생성하여 반환한다.
- **public boolean** equals(Object obj) : obj가 이 객체와 같은지를 나타낸다.
- **protected void** finalize() : 가비지 콜렉터에 의하여 호출된다.
- **public final Class** getClass() : 객체의 실행 클래스를 반환한다.
- **public int** hashCode() : 객체에 대한 해쉬 코드를 반환한다.
- **public String** toString() : 객체의 문자열 표현을 반환한다.



# equals() 메소드

```
public class Car {
        private String model;

        public Car(String model) {
                this.model = model;
        }

        public String getModel() {
                return model;
        }

        public boolean equals(Object obj) {
                if (obj instanceof Car)
                        return model.equals(((Car) obj).getModel());
                else
                        return false;
        }
}
```

Object의 equals()를 재정의

# equals() 메소드

```
public static void main(String[] args) {
        Car firstCar = new Car("HMW520");
        Car secondCar = new Car("HMW520");
        if (firstCar.equals(secondCar)) {
                System.out.println("동일한 종류의 자동차입니다.");
        } else {
                System.out.println("동일한 종류의 자동차가 아닙니다.");
        }
}
```

재정의된 equals() 호출



# toString()

- Object 클래스의 toString() 메소드는 객체의 문자열 표현을 반환

```
public class Car {
        private String model;
        public Car(String model) {
                this.model = model;
        }
        public String toString() {
                return "모델: " + model;
        }
}
```

Object의 tostring()를 재정의

# 종단 클래스 와 종단 메소드

- 키워드 final을 붙이면 상속이나 재정의할 수 없다.

```
class Baduk {
    enum BadukPlayer { WHITE, BLACK }
    ...
    final BadukPlayer getFirstPlayer() {
        return BadukPlayer.BLACK;
    }
}
```

재정의할 수 없도록 한다.



# Q & A