상속

- 상속이란?
  - 부모의 필드와 메소드 갖다 쓴다.
  - 부모 클래스 **vs** 자식 클래스
  - 슈퍼 클래스 **vs** 서브 클래스
- 상속의 장점
  - 코드의 재사용
  - 코드의 중복 방지
- 상속의 선언
  - extends

- 같은 이름의 메서드 사용
  - overloading(수평)
  - overriding(수직)
    - @Override

- protected : 상속 관계의 공개 설정

- super vs this
  - super : 부모의
  - this : 나의

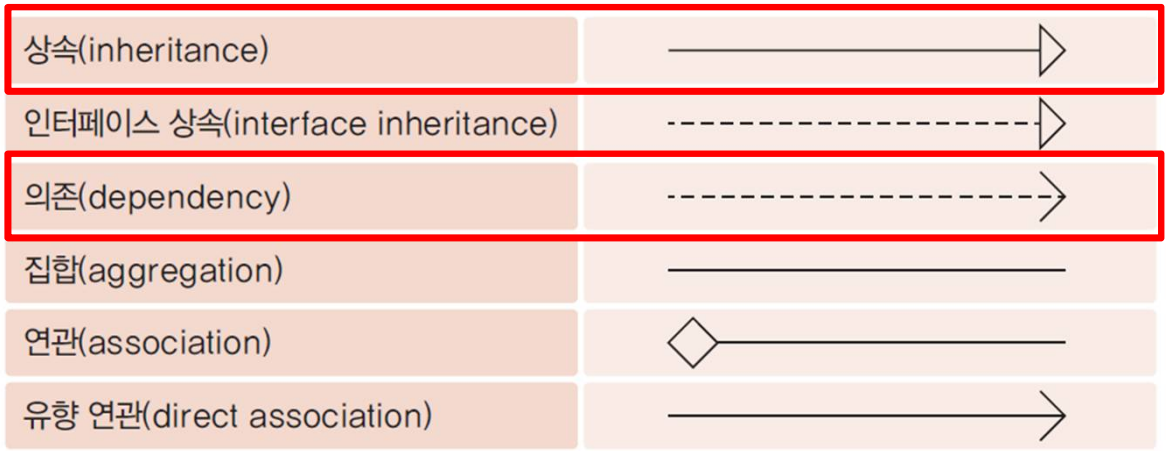
상속은
코드의 재사용,
코드의 중복 방지
하기 위한 중요한
기법입니다.

- 상속에서 생성자 호출 순서
  - 부모부터 자식순으로

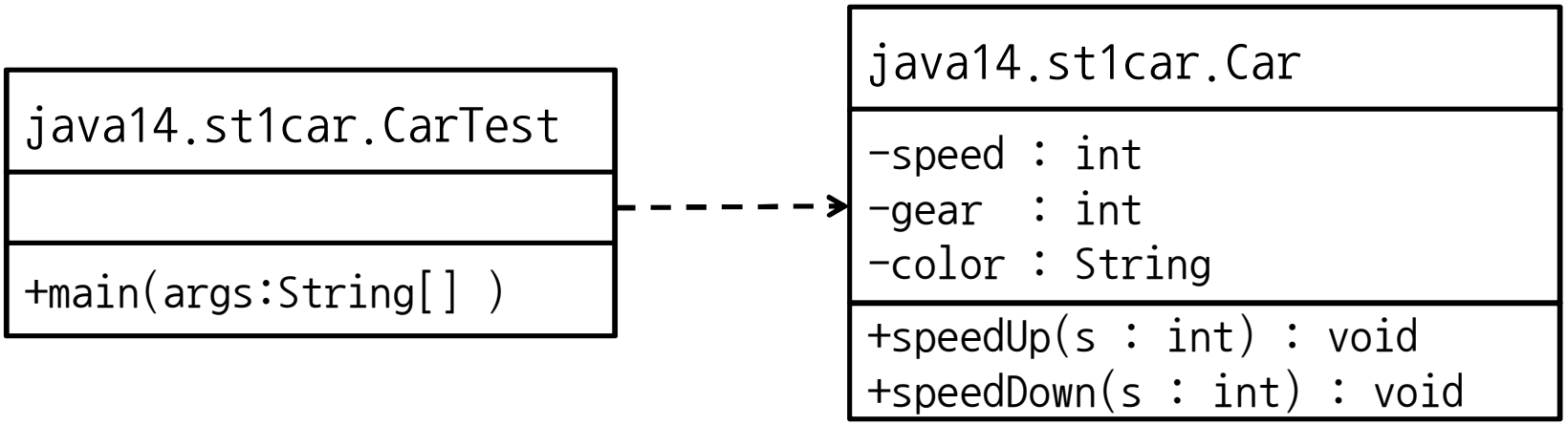
- 클래스 관계
  - 상속 : is-a
  - 의존 : has-a

- Object 클래스 중요 메서드
  - toString()
  - equals()

클래스와 클래스의 관계

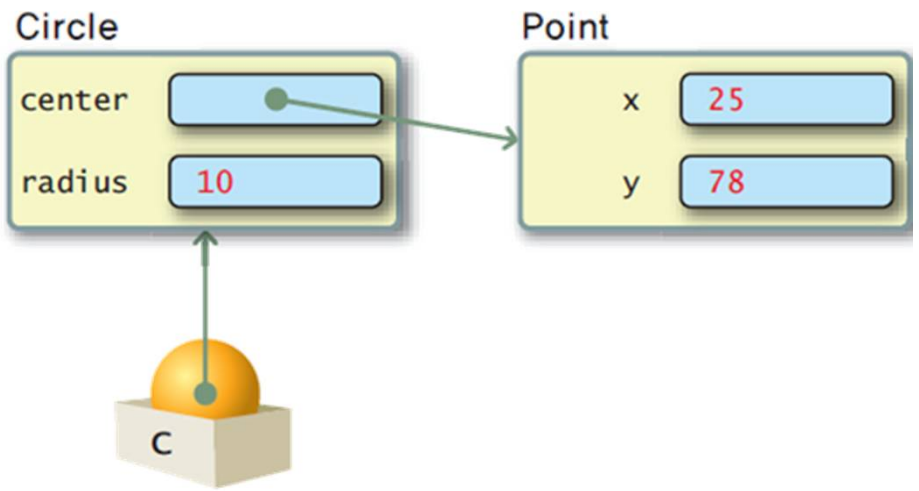


- 상속(is-a): 하나의 클래스가 다른 클래스를 상속한다.
- 의존(has-a): 하나의 클래스가 다른 클래스를 포함한다.



의존 관계 예제

패키지명: java14.st2circle
클래스명: Point, Circle, CircleTest



Point.java

```
public class Point {
   private int x, y;
   // 생성자
   public Point(int a, int b) {
      x = a;
      y = b;
   }
}
```

Circle.java

```
public class Circle {
   private int radius = 0;
   private Point center;      // Point 참조 변수가 필드로 선언되어 있다.

   // 생성자
   public Circle(Point p, int r) {
      center = p;
      radius = r;
   }
}
```

Circle 객체가 Point 객체를 포함하고 있다.

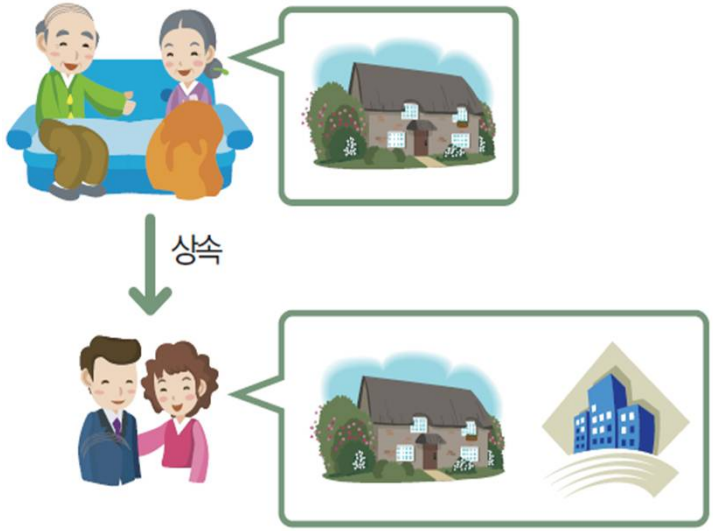
CircleTest.java

```
public class CircleTest {
   public static void main(String args[]) {
      // Circle 객체를 생성하고 초기화한다.
      Point p = new Point(25, 78);
      Circle c = new Circle(p, 10);
      }
}
```

Circle 객체를 생성할 때,
Point 객체 참조값을 넘긴다.

상속의 개념

- 상속이란 부모의 **필드**와 **메서드**를 자식이 물려받아서 사용하는 것
  - 부모의 유산을 물려받다
  - 자식이 부모의 것을 가진다



부모클래스==**수퍼클래스**

자식클래스==**서브클래스**

상속의 장점

- 상속의 장점
  - **상속을 통하여 부모 클래스의 필드와 메소드를 자식 클래스에서 사용 할 수 있다.**
    - **→ 코드의 재사용을 통해서**
    - **→ 코드의 중복을 피할 수 있다.**
  - 부모 클래스의 일부도 변경 가능 :
    - 메서드 재정의(오버라이딩)
  - 상속을 이용하게 되면 GUI 프로그램을 쉽게 작성할 수 있다.
    - swing , 안드로이드, JavaFX
  - 손쉽게 개발 및 유지 보수 할 수 있다.

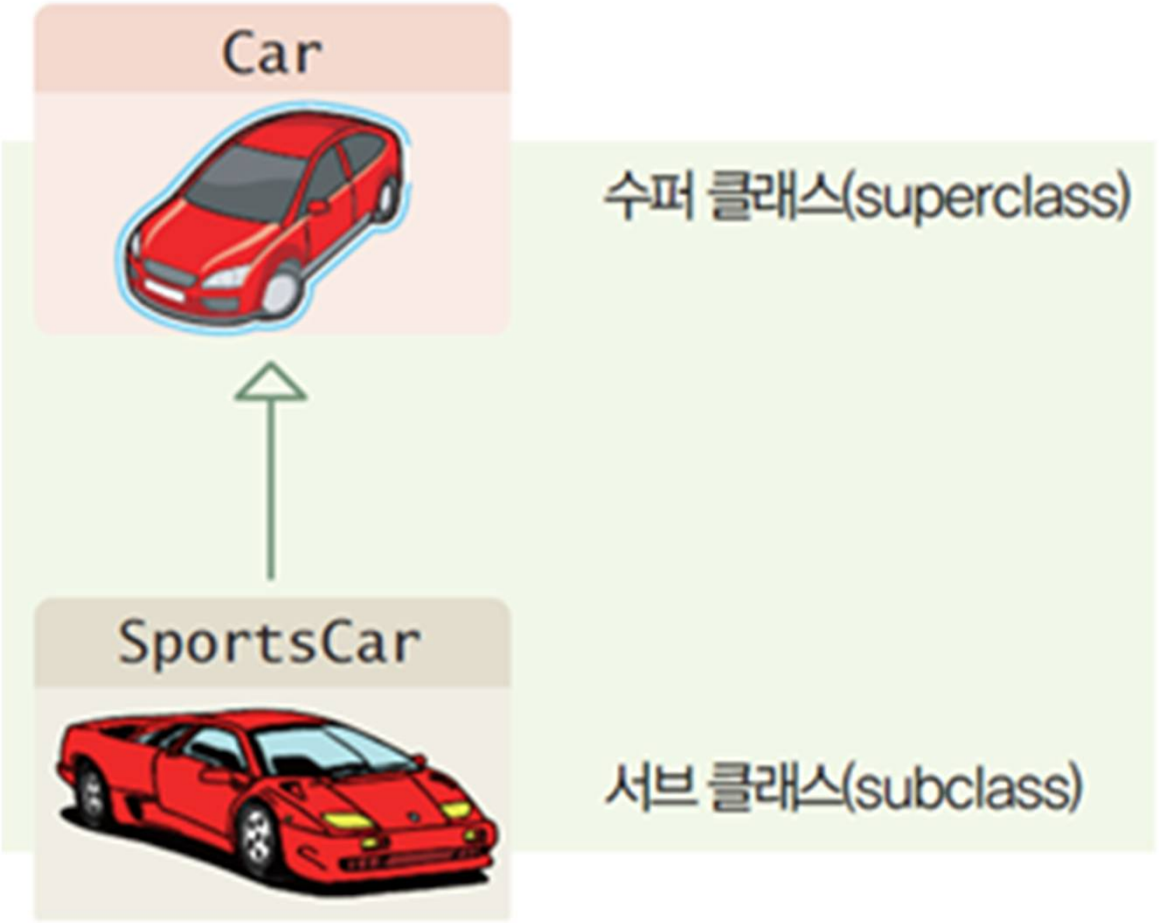
상속

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)

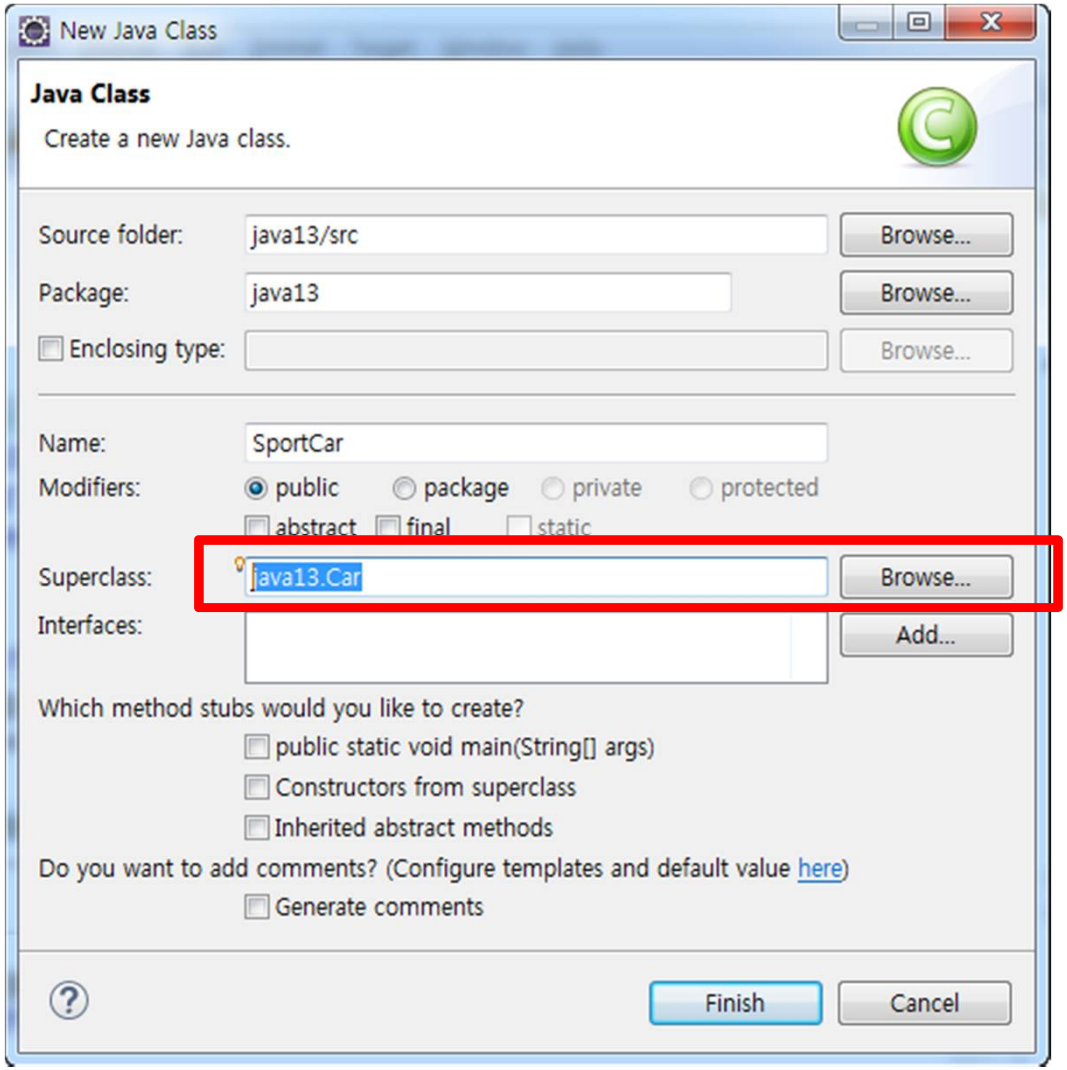
중간점검

1. 컴퓨터, 데스크탑, 노트북, 태블릿 사이의 상속 관계를 결정하여 보자.
2. 상속의 장점은 무엇인가?

상속의 계층 구조



상속의 선언


New Java Class
Java Class
Create a new Java class.
Source folder:
java13/src
Browse...
Package:
java13
Browse...
Enclosing type:
Browse...
Name:
SportCar
Modifiers:
public
package
private
protected
abstract
final
static
Superclass:
java13.Car
Browse...
Interfaces:
Add...
Which method stubs would you like to create?
public static void main(String[] args)
Constructors from superclass
Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
Generate comments
Finish
Cancel

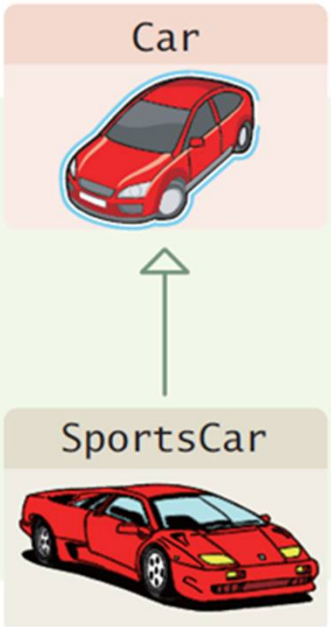
상속의 선언

```
class SubClass extends SuperClass
{
    ...// 추가된 메소드와 필드
}
```

상속을 의미한다. 수퍼 클래스를 확장하여 서브 클래스를 작성한다는 의미이다.

수퍼 클래스==부모 클래스

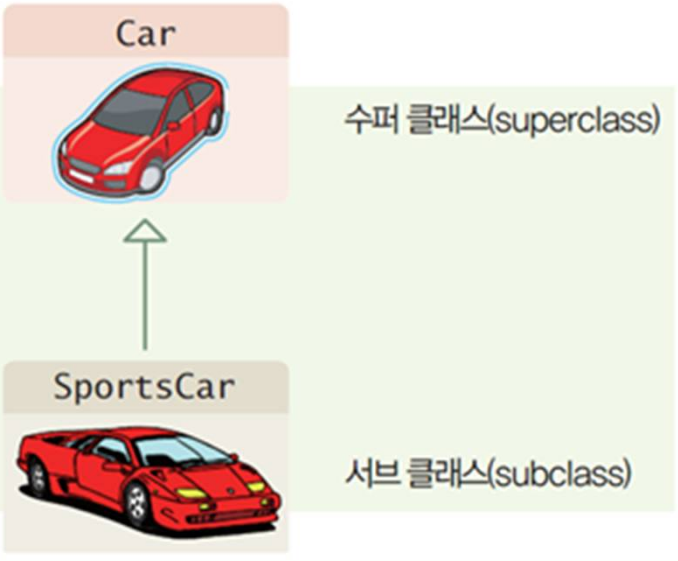
```
class Car
{
    ...
}
class SportsCar extends Car
{
    ...
}
```



수퍼 클래스(superclass)

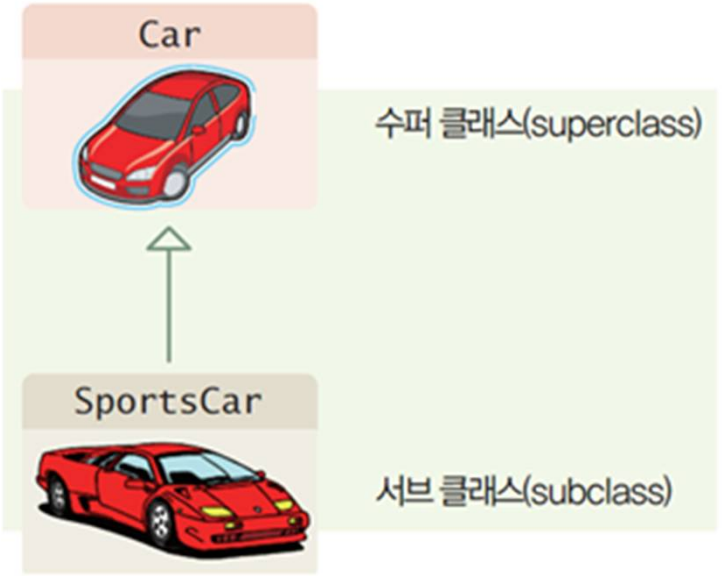
서브 클래스(subclass)

상속 사용




```
public class Car {
    // 3개의 필드 선언
    int speed;
    int gear;
    public String color;                    // 색상, 테스트를 위하여 공용 필드로 만들자.

    public void speedUp(int increment) {    // 속도 증가 메소드
        speed += increment;
    }
    public void speedDown(int decrement) {  // 속도 감소 메소드
        speed -= decrement;
    }
}
```

상속 사용

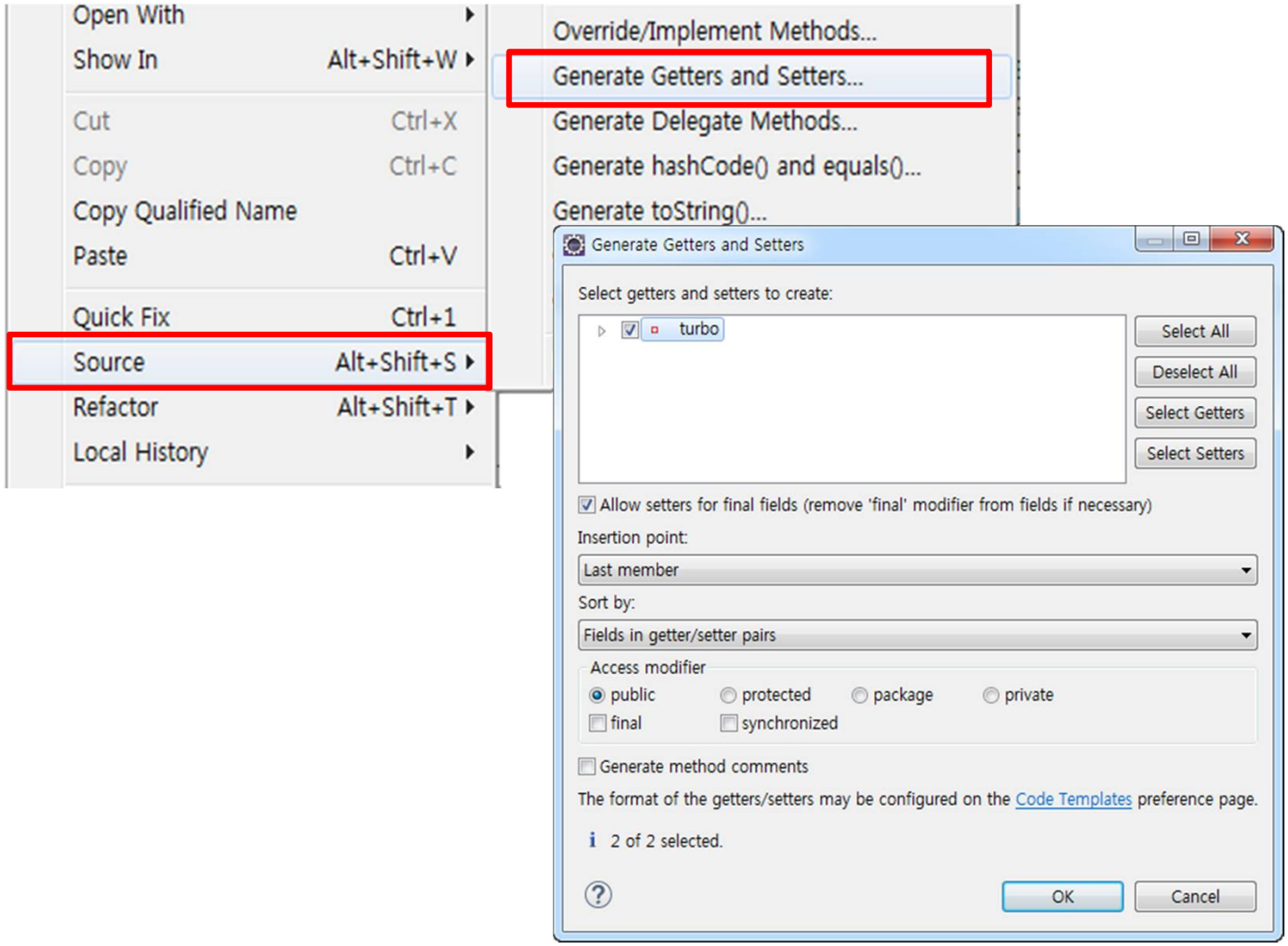


추가된 필드 ---------->

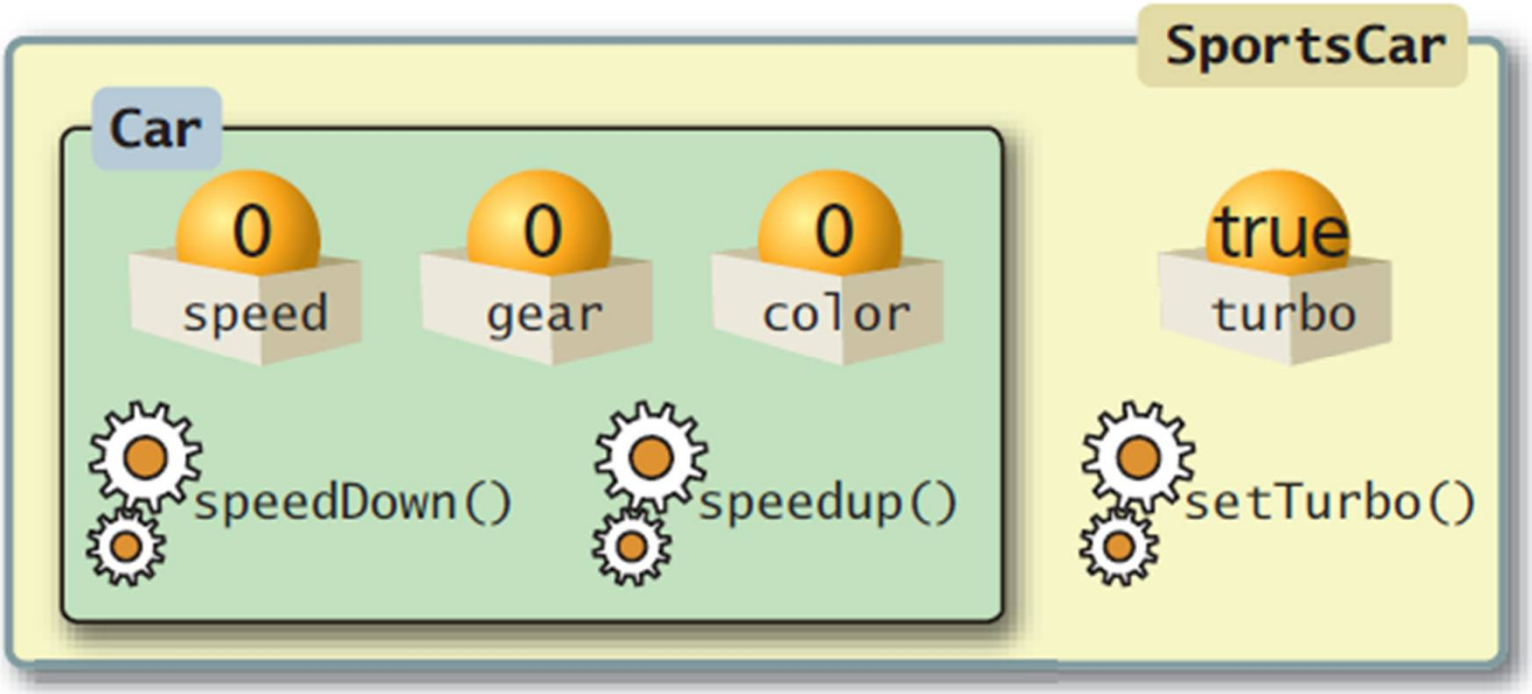
추가된 메소드 ---------->

```
class SportsCar extends Car {                    // Car를 상속받는다.
    boolean turbo;

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

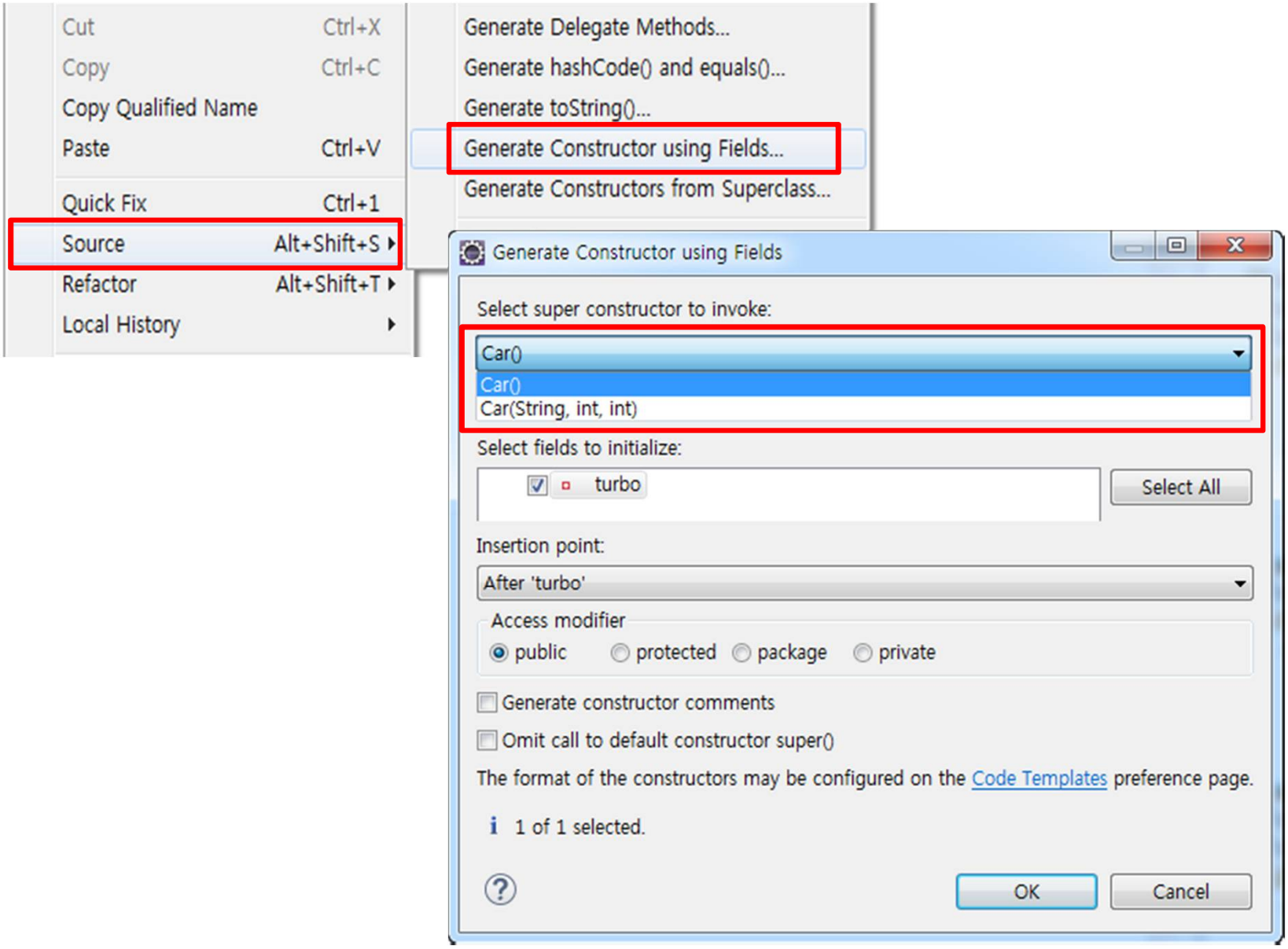
예제. setter와 getter 만들기



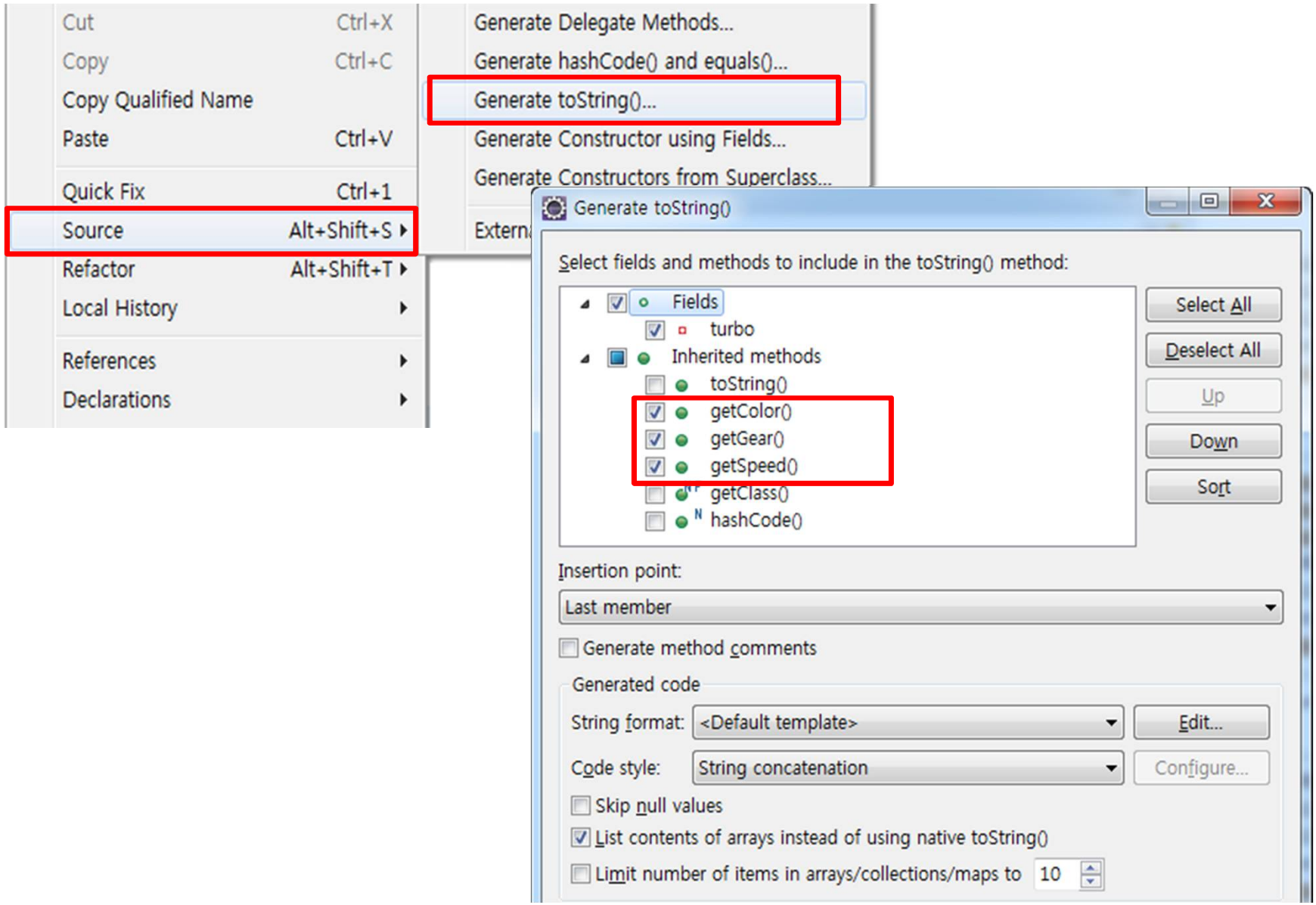
예제. 생성자 만들기


SportsCar
Car
0
speed
0
gear
0
color
true
turbo
speedDown()
speedup()
setTurbo()

예제. 생성자 만들기

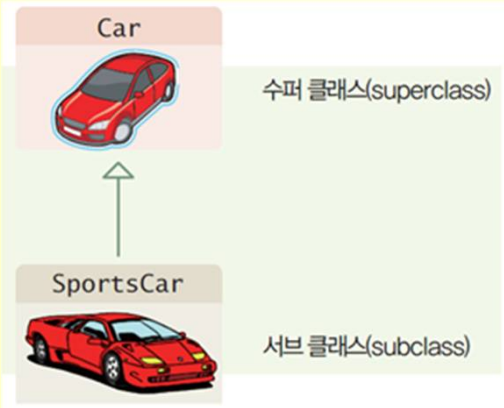

Cut Ctrl+X
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Quick Fix Ctrl+1
Source Alt+Shift+S
Refactor Alt+Shift+T
Local History
Generate Delegate Methods...
Generate hashCode() and equals()...
Generate toString()...
Generate Constructor using Fields...
Generate Constructors from Superclass...
Generate Constructor using Fields
Select super constructor to invoke:
Car()
Car()
Car(String, int, int)
Select fields to initialize:
turbo
Select All
Insertion point:
After 'turbo'
Access modifier
public protected package private
Generate constructor comments
Omit call to default constructor super()
The format of the constructors may be configured on the Code Templates preference page.
1 of 1 selected.
OK
Cancel

예제. toString() 만들기

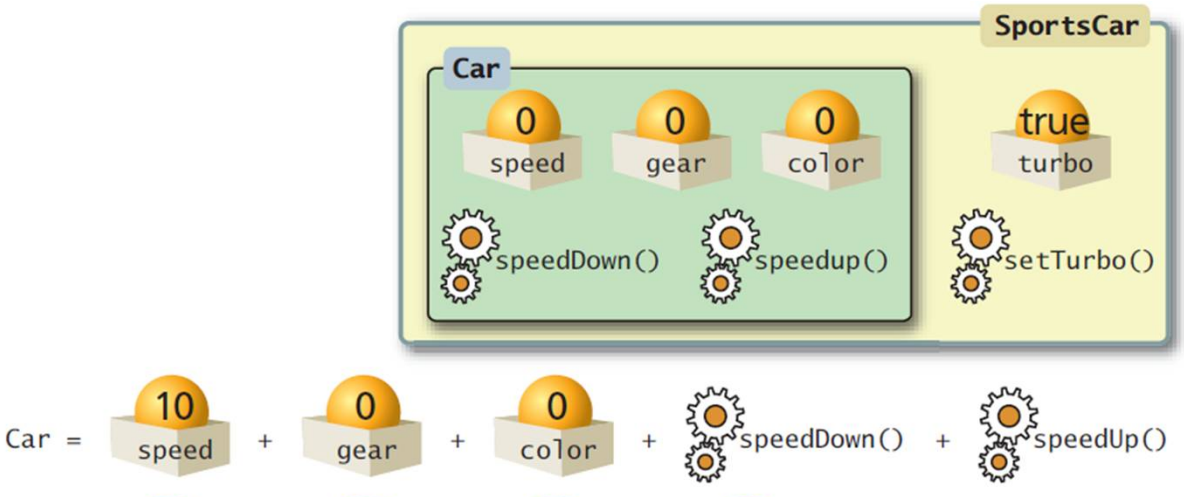


상속 사용

```
public class SportsCarTest {

    public static void main(String[] args) {

        // setter를 이용한 초기화
        SportsCar c1  = new SportsCar();
        c1.setColor("white");
        c1.setGear(6);
        c1.setSpeed(250);
        c1.setTurbo(true);

        SportsCar c4  = new SportsCar("green", 300, 5, true);

        // 생성자를 이용한 초기화: turbo 설정
        SportsCar c2  = new SportsCar(true);

        // 생성자를 이용한 초기화: turbo, color 설정
        SportsCar c3  = new SportsCar("black", true);

    }
}
```



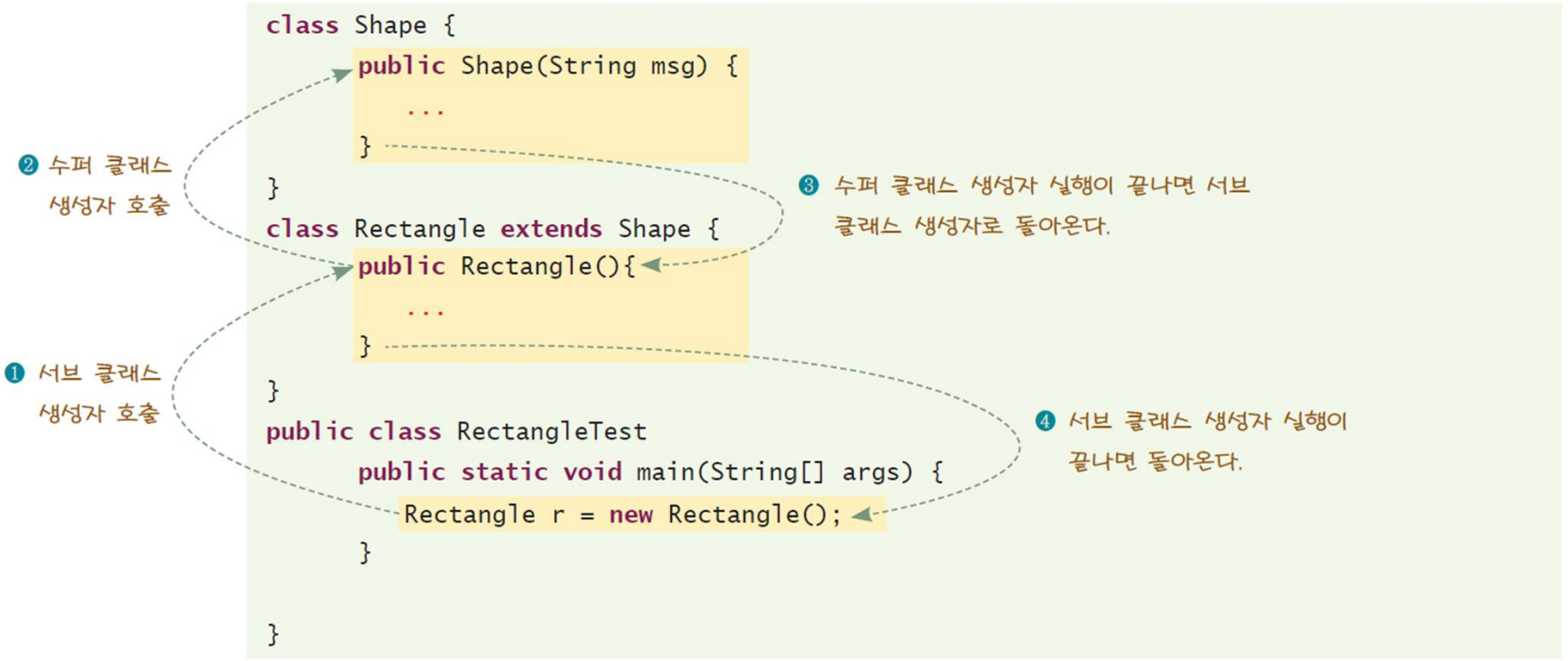
상속 사용



- 상속을 통하여 부모 클래스의 필드와 메소드를 사용 할 수 있다.

상속에서의 생성자 호출 순서

- 서브 클래스의 객체가 생성될 때, 서브 클래스의 생성자만 호출될까? 아니면 수퍼 클래스의 생성자도 호출되는가?




```
class Shape {
        public Shape(String msg) {
            ...
        }
}
class Rectangle extends Shape {
        public Rectangle(){
            ...
        }
}
public class RectangleTest
        public static void main(String[] args) {
            Rectangle r = new Rectangle();
        }

}
```

묵시적인 호출

- 묵시적인 호출

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

컴파일러가 Shape();을 자동적으로 넣어준다고 생각하라.

실행결과

```
Shape 생성자
Rectangle 생성자
```

명시적인 호출

- 명시적인 호출
  - super를 이용하여서 명시적으로 수퍼 클래스의 생성자 호출

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

public class Rectangle extends Shape {
    public Rectangle() {
        super("from Rectangle");          // 명시적인 호출
public class RectangleTest                    생성자()");
    public static void main(String[] args) {
        Rectangle r = new Rectangle();
    }

}
```

실행결과

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

실행결과

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

클래스간의 관계

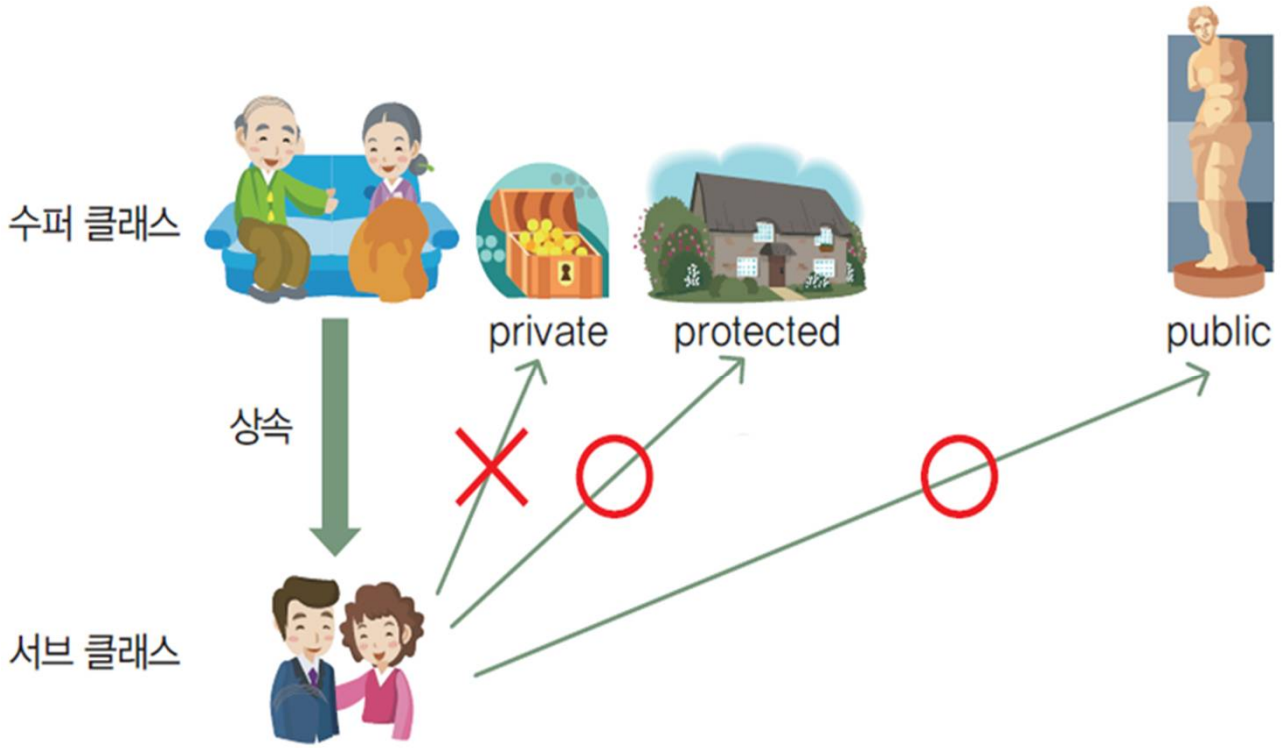
- 상속 : is-a
  - 자동차는 탈것이다. (Car **is a Vehicle).**
  - 사자는 동물이다.
  - 개는 동물이다.
  - 고양이는 동물이다.

- 의존 : has-a
  - 도서관은 책을 가지고 있다(Library **has a book).**
  - 거실은 소파를 가지고 있다.

```
class Point {
    int x;
    int y;
}
```

```
class Line {
    Point p1; // 객체 포함
    Point p2; // 객체 포함
}
```

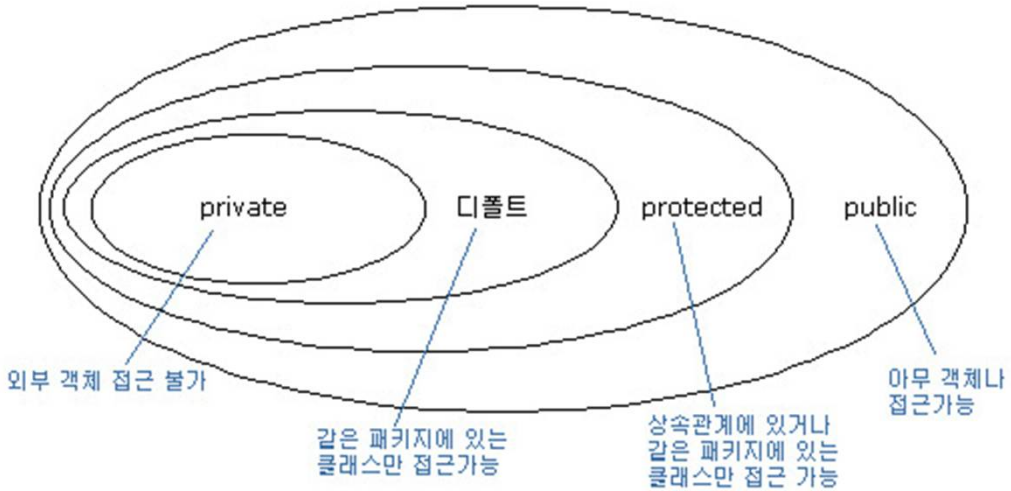
상속에서 접근 지정자



접근 지정자 protected는
상속 관계에서만 사용할 수 있다

access control

- public == 무조건 공개
- protected == 조건부 공개
- 디폴트 == 패키지 공개
- private == 비공개



분류	접근 지정자	클래스 내부	같은 패키지 내의 클래스	다른 모든 클래스
전용 멤버	private	O	X	X
패키지 멤버	없음	O	O	X
공용 멤버	public	O	O	O

중간점검

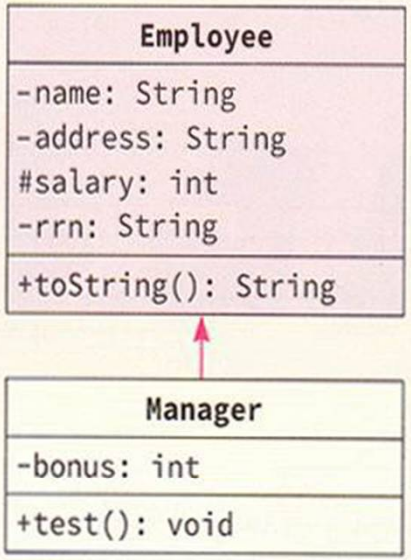
중간점검

1. Animal 클래스 (sleep(), eat()), Dog 클래스(sleep(), eat(), bark()), Cat 클래스 (sleep(), eat(), play())를 상속을 이용하여서 표현하면 어떻게 코드가 간결해지는가?
2. 일반적인 상자(box)를 클래스 Box로 표현하고, Box를 상속받는 서브 클래스인 ColorBox(컬러 박스) 클래스를 정의하여 보자. 적절한 필드(길이, 폭, 높이)와 메소드(부피 계산)를 정의한다.

상속 예제:직원과 매니저 클래스

직원(Employee)과 매니저(Manager)의 예를 가지고 프로그램을 작성하여 보자. 직원을 나타내는 클래스 Employee는 이름, 주소, 주민등록 번호, 월급 등의 정보를 가지고 있다. 매니저를 나타내는 Manager 클래스에는 보너스가 추가되어 있다. Employee 클래스에서 주민등록번호는 private로 선언되고 월급 정보는 protected로 선언된다.

```
+ : public
- : private
# : protected
```



위의 URL 클래스 다이어그램을 보고 Employee와 Manager 클래스를 구현한다. 각 클래스의 생성자를 추가한다. 자식 클래스인 Manager 클래스의 test() 메소드에서 부모 클래스의 필드 값을 출력하여 보자. 어떤 필드를 출력할 때, 문법적인 오류가 발생하는지 살펴보자.

상속 예제:직원과 매니저 클래스

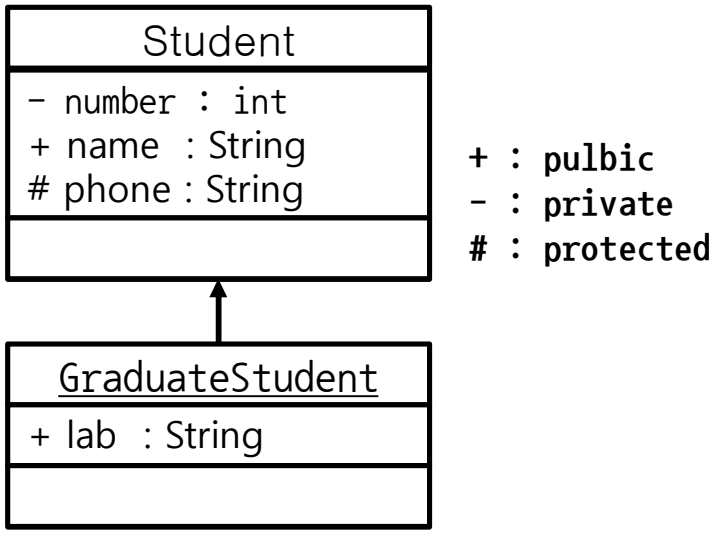
```
public class Manager extends Employee {
    private int bonus;

    // getter & setter

    // toString override

    // 생성자

    // 메서드 선언
    public void printSarary() {
        System.out.println(
            super.getName()
            + "(" + super.addr + ")"
            + ( super.salary + this.bonus)
        );
    }

    public void printJuminNum() {
        System.out.println( super.getJuminNum() );
    }
}

public class ManagerTest {

    public static void main(String[] args) {
        Manager m = new Manager();

        m.printJuminNum();

    }
}
```

학생과 대학원생 클래스 작성

- 클래스 정의가 다음과 같다고 할 때

```
class Student {
    private int number;
    public String name;
}
```

```
public class GraduateStudent {

    public String lab;

}
```



① 클래스 마다 각 필드의 Getter, Setter, toString() 작성하시오.

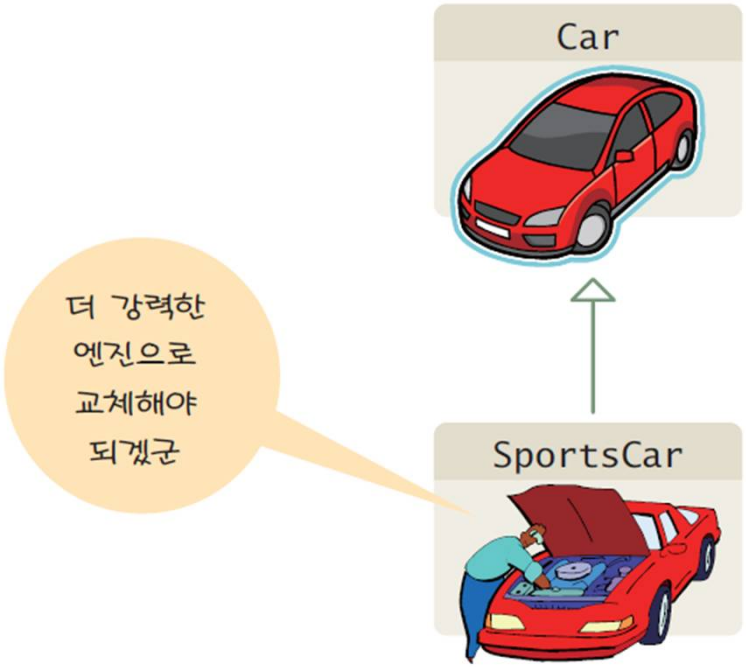
② toString()을 작성 하시오. GraduateStudent의 toString()은 Student의 필드 정보까지 나오도록 하시오.

③ Student, GraduateStudent 클래스에 생성자 메서드를 작성하시오.

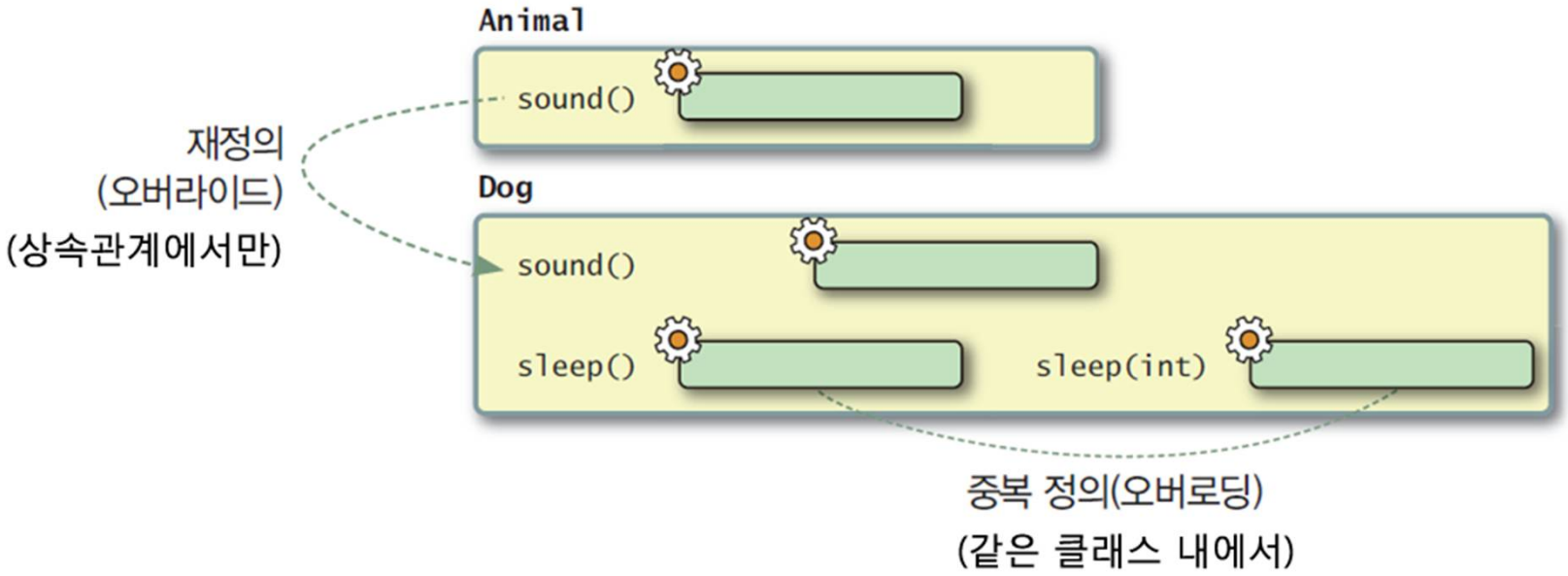
④ 테스트하기 위한 GraduateStudentTest 클래스를 작성하고 GraduateStudent의 toString()을 출력하시오.

메소드 재정의

- 메소드 재정의(method overriding):
  - 자식 클래스에서 부모 클래스의 메소드를 다시 정의하는 것
  - 상속 관계에서만 성립



중복 메서드 vs 메서드 재정의
overloading vs overriding



같은 이름으로 메서드를 정의하는 방법

메소드 재정의의 예

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

메소드 재정의

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

```
public class DogTest  {
        public static void main(String[] args) {
            Dog d = new Dog();
            d.sound();
        }
}
```

재정의된 메소드가 호출된다.

@Override

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

재정의를 의도하였으나 이름을 잘못 입력하였기 때문에 컴파일러는 새로운 메소드 정의로 간주한다.

```
class Dog extends Animal {
    @Override
    void saund() {              // 오류 발생!
        System.out.println("멍멍!");
    }
}
```

재정의를 의도하였다는 것을 확실하게 컴파일러에게 전달하여 오류를 막는다.

실행결과

```
The method saund() of type Dog must override or implement a supertype method
```

super

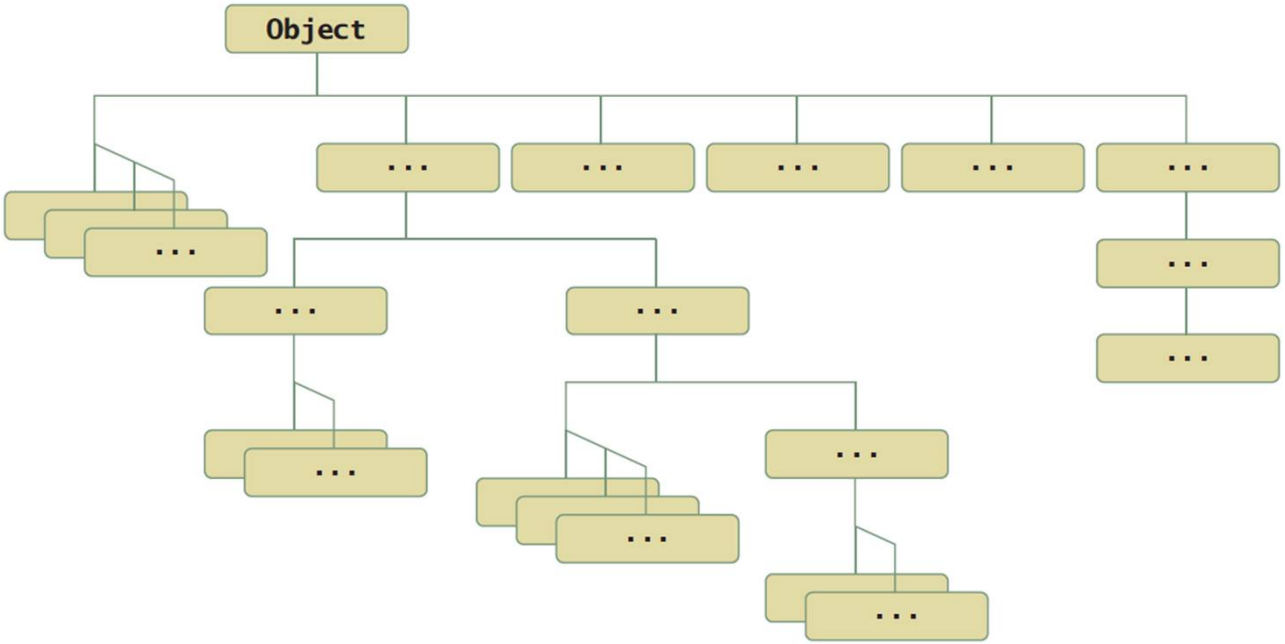
```
class ParentClass {
    int data=100;
    public void print() {
        System.out.println("수퍼 클래스의 print() 메소드");
    }
}
```

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

super vs this

java.lang.Object 클래스

- Object 클래스
  - 자바의 최상위 클래스
  - java.lang 패키지에 있음



Object의 중요 메소드

- **public String** toString()
  - 객체의 문자열 표현을 반환한다.
- **public boolean** equals(Object obj)
  - obj가 이 객체와 같은지를 나타낸다.
- **protected void** finalize()
  - 가비지 콜렉터가 객체를 삭제하기 전에 호출된다.

- **public final Class** getClass()
  - 객체를 생성한 클래스 정보를 반환한다.
- **protected Object** clone()
  - 객체 자신의 복사본을 생성하여 반환한다.
- **public int** hashCode()
  - 객체에 대한 해쉬 코드를 반환한다.

toString() 메소드

- Object 클래스의 toString() 메소드는 객체의 문자열 표현을 반환

```
System.out.println(firstCar.toString());
```

실행결과

```
모델 HMW 520...
```

equals() 메소드

```
class Car {
    private String model;
    public Car(String model) {      this.model= model;     }
    public boolean equals(Object obj) {
        if (obj instanceof Car)
            return model.equals(((Car) obj).model);
        else
            return false;
    }
}
public class CarTest {
    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.");
        }
    }
}
```

equals()를 재정의한다. String의 equals()를 호출하여서 문자열이 동일한지를 검사한다.

이 equals() 메소드를 사용하여 검사하는 다음과 같은 코드를 가정할 있다.

실행결과

동일한 종류의 자동차입니다

finalize()메소드

- 객체가 소멸되기 직전에 호출된다.
- Object 클래스는 finalize()라는 콜백(callback) 메소드를 정의한다.
- finalize()를 서브 클래스에서 재정의하여 자원을 반납하는 등의 정리 과정에 사용 할 수 있다

종단 클래스 : final

- final 클래스 : 상속이나 재정의할 수 없는 클래스.
  - 보안상의 이유로 상속을 막는 경우에 효과적인 방법
  - String 클래스

```
final class String {
    ...
}
```

종단 메소드 : final

- final 메서드 : 재정의(overriding) 할 수 없는 메서드

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

서브 클래스에서 재정의할 수 없도록 final로 지정한다.