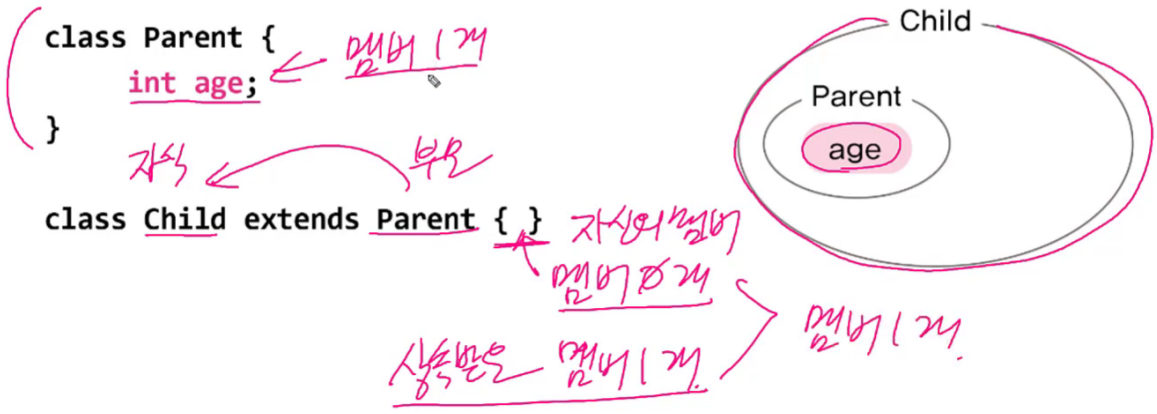
CH 7 객체 지향 2

상속

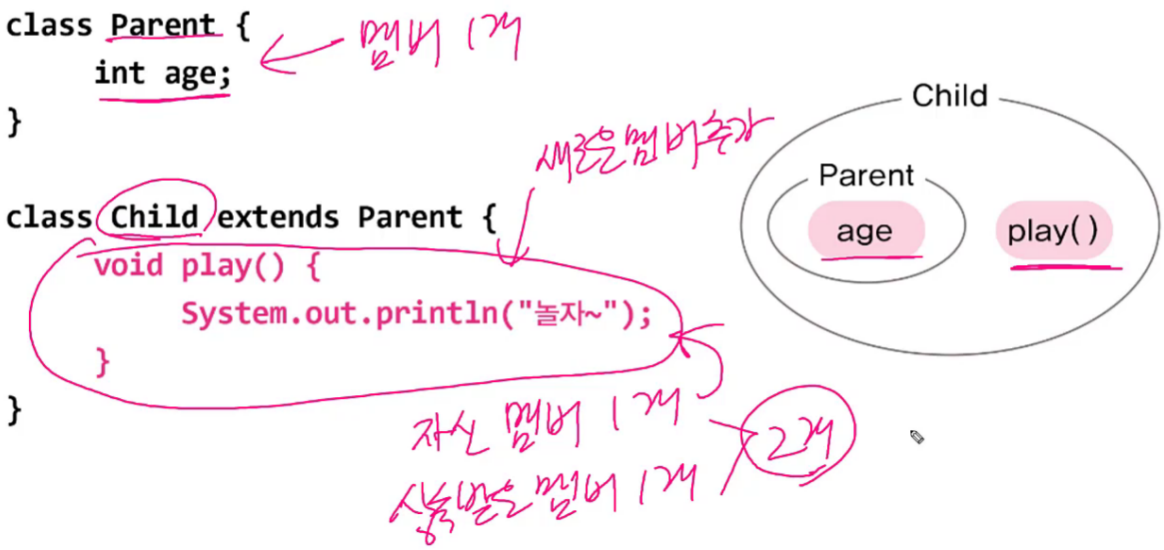
- 기존의 클래스로 새로운 클래스를 작성하는 것 ⇒ 코드의 재사용
- 두 클래스를 부모와 자식으로 관계를 맺어 주는 것

```
class 자식 클래스 extends 부모 클래스 {
}

// 예시
class Parent {
}
// 이 두 클래스는 상속 관계에 있다
class Child extends Parent {
}
```

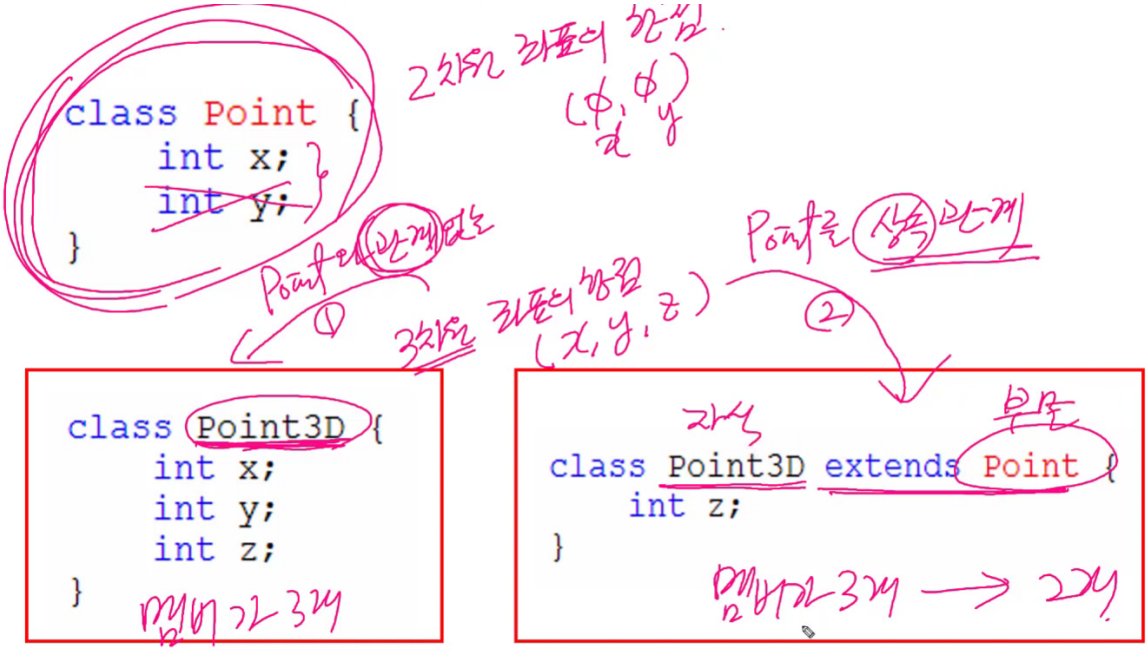
- 자손은 조상의 모든 멤버를 상속받는다
    - 생성자, 초기화 블럭은 제외한다
- 자손의 멤버 개수는 조상보다 적을 수 없다
    - 즉, 자손의 `멤버 개수`는 조상과 `같거나 보다 많다`



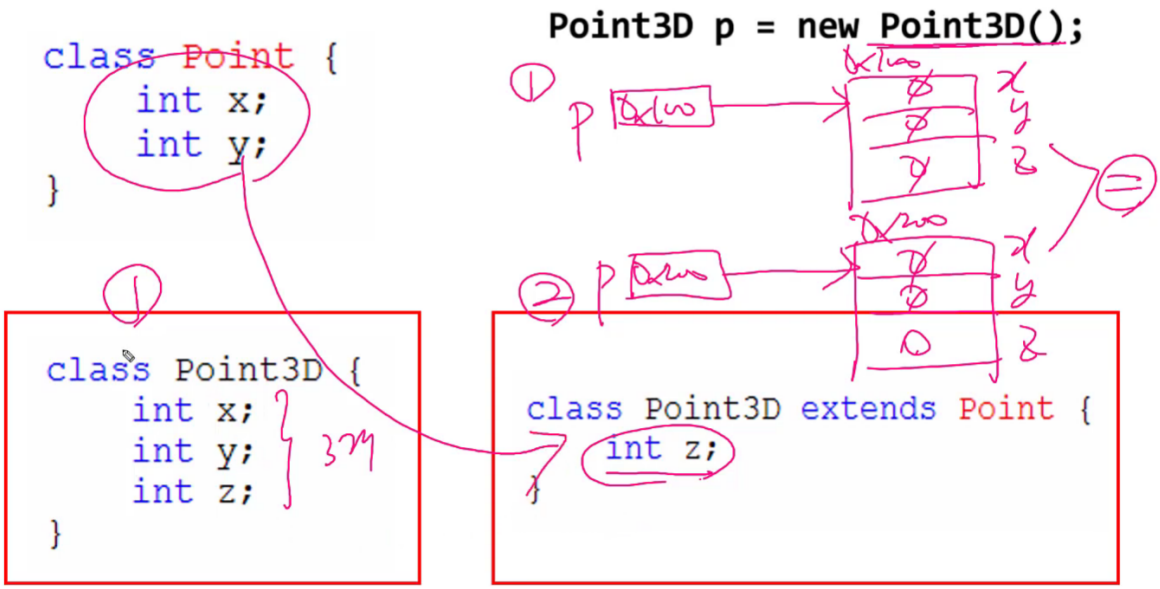
- 자손의 변경은 조상에 영향을 미치지 않는다
  - 자손은 점점 확장된다 (`extends` 라는 키워드를 쓰는 이유)



- 자식 클래스는 부모 클래스의 변경에 영향을 받는다
  - y를 제거하면 전자는 영향을 받지 않지만, 후자의 멤버에서도 y가 사라진다



- 상속을 받든 안 받든 객체가 생성된 그림은 같다




```
class Tv {
  // 부모 멤버 5개
  boolean power; // 전원 상태 (on/off)
  int channel;

  void power() {
    power = !power;
  }
```

```
  void channelUp() {
    ++channel;
  }

  void channelDown() {
    --channel;
  }
}

// 자식 멤버 2개
// 총 멤버 7개
class SmartTv extends Tv { // 스마트TV는 TV에 자막 기능 제공
  boolean caption; // 자막 상태 (on/off)
  // 자막이 on일 때만 text를 보여 준다

  void showCaption(String text) {
    if (caption) {
      System.out.println(text);
    }
  }

}

public class Ex7_1 {
  public static void main(String[] args) {
    SmartTv stv = new SmartTv();
    // 조상 클래스로부터 상속받은 멤버
    stv.channel = 10;
    stv.channelUp();
    System.out.println(stv.channel);

    stv.showCaption("Hello, world!");
    // 자막 켜기
    stv.caption = true;
    stv.showCaption("Hello, world!");

  }

}
```

포함 관계

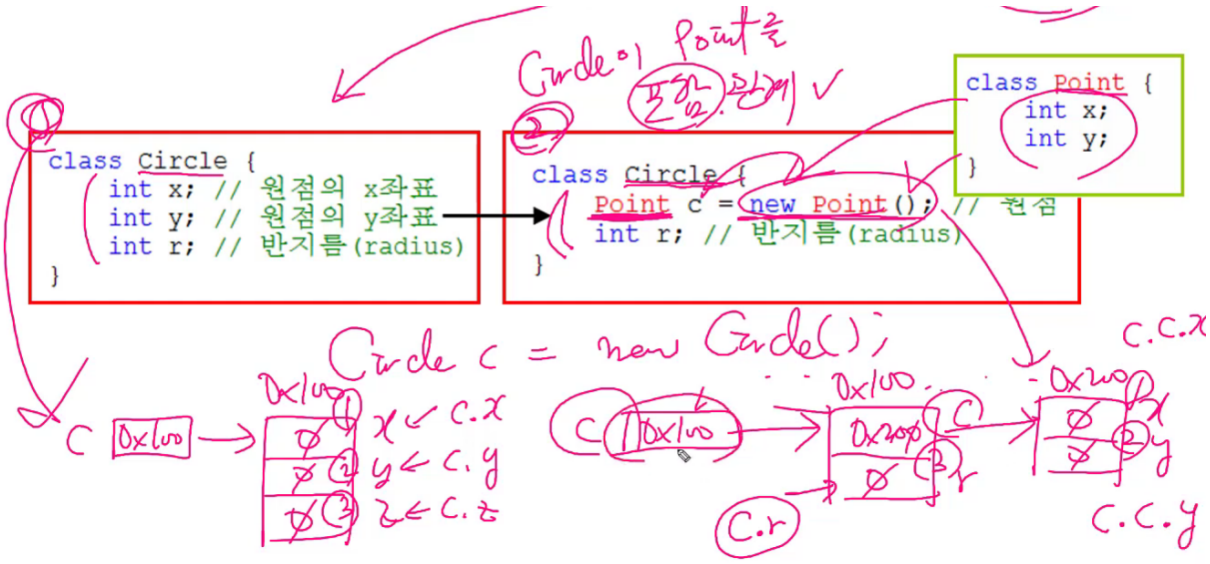
포함이란?

- 클래스의 멤버로 참조 변수를 선언하는 것

💡 클래스의 관계
1. 상속
2. 포함

- 포함 관계의 예시

```
// 포함 관계가 아닐 때
class Circle {
    int x; // 원점 x 좌표
    int y; // 원점 y 좌표
    int r; // 반지름
}

// 포함 관계일 때
class Point {
    int x;
    int y;
}

class Circle {
  Point c = new Point(); // 원점
  int r; // 반지름
}
```



- 저장 공간의 수는 같지만, 구조적 관계가 다르다
-

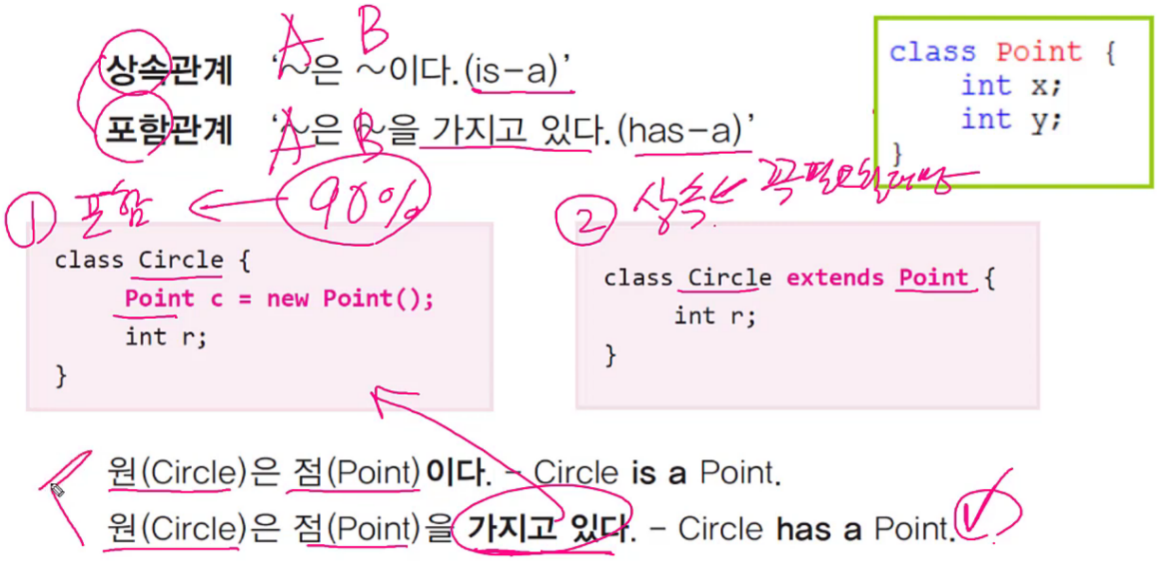
```
class Car {
    Engine e = new Engine();   // 엔진
    Door[] d = new Door[4];    // 문, 문의 개수를 넷으로 가정하고 배열로 처리했다.
    //...
}
```

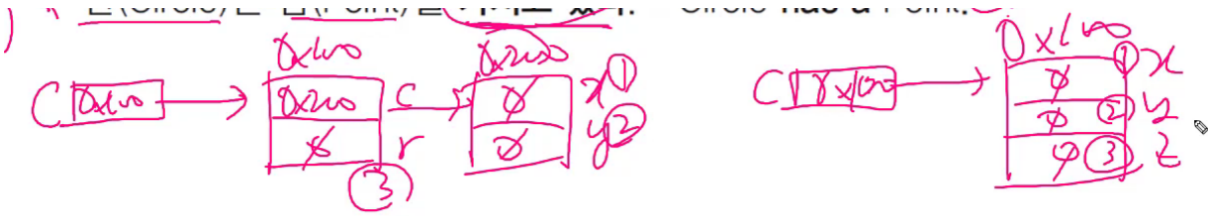
- 작은 단위의 클래스를 만들고, 이들을 조합해서 클래스를 만든다

클래스 간 관계 결정하기

> 💡 **상속 관계** '~은 ~이다. (is - a)'
> **포함 관계** '~은 ~을 가지고 있다. (has - a)'

- A와 B를 물결에 넣어 어느 표현이 더 자연스러운지 확인하기





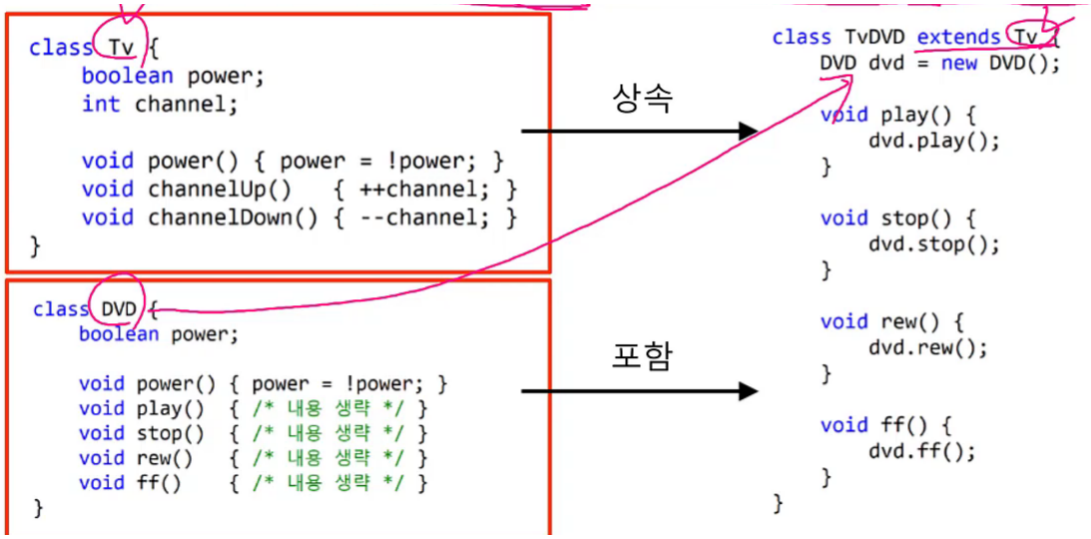
- 상속은 제약 사항이 많기 때문에 꼭 필요한 경우에만 쓰기
- 애매할 때는 포함 관계로 쓰기

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

class Circle extends Point {
  int r;

}

// 포함
class MyPoint {
  int x;
  int y;
}

class MyCircle {
  // 참조 변수의 초기화
  MyPoint p = new MyPoint();
  int r;

  // 참조 변수 초기화 방법 2
  // 생성자에서 초기화
  MyCircle() {
    p = new MyPoint();
  }
}

public class Ex7_2 {
  public static void main(String[] args) {
    Circle c = new Circle();
    c.x = 1;
    c.y = 2;
    c.r = 3;
    System.out.println("c.x=" + c.x);
    System.out.println("c.y=" + c.y);
    System.out.println("c.r=" + c.r);

    MyCircle mc = new MyCircle();
    mc.p.x = 4;
    mc.p.y = 5;
    mc.r = 6;
    System.out.println("mc.p.x=" + mc.p.x);
    System.out.println("mc.p.y=" + mc.p.y);
    System.out.println("mc.r=" + mc.r);

  }

}
```

단일 상속(Single Inheritance)

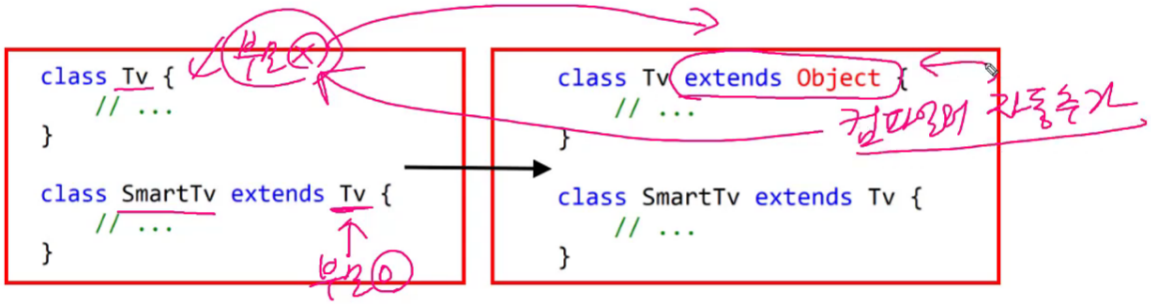
- 자바는 단일 상속만을 허용한다 ↔ C++은 다중 상속 허용

- 단일 상속이란? 하나의 부모(조상)만 상속받는 것을 의미한다
  - 조상이 여러 개면 충돌한다는 문제가 있기 때문이다
  - 인터페이스로 다중 상속 문제 해결 가능하다
- 비중이 높은 클래스 하나만 상속 관계로 하고, 나머지는 포함 관계로 한

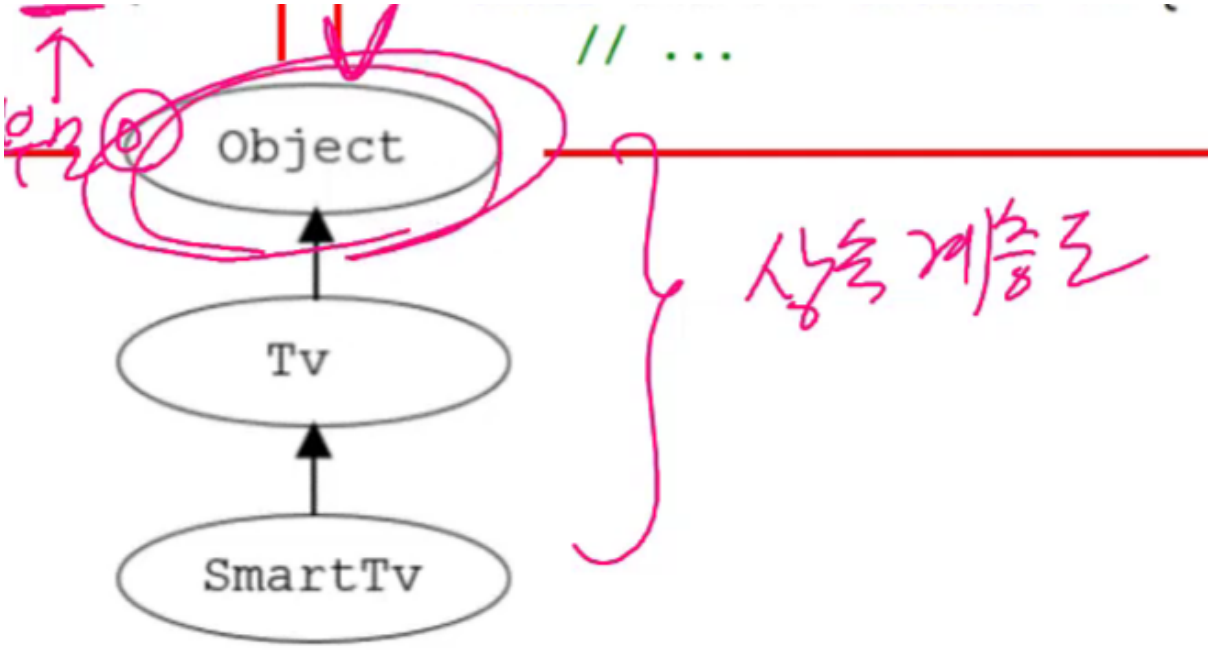


Object 클래스

- 모든 클래스의 조상 역할을 한다
- 부모가 없는 클래스는 자동적으로 Object 클래스를 상속받게 된다



- 모든 클래스는 Object 클래스에 정의된 11개의 메서드를 상속받는다
  - `toString(), equals(Object obj), hashCode()`




```
class Kitty extends Object {
  String color;
  int age;
  String name;
}

public class Ex7_3 {
  public static void main(String[] args) {
    Kitty k = new Kitty();
    System.out.println(k.toString()); //Kitty@2a139a55
    System.out.println(k); // toString을 안 붙여도 똑같은 결과가 나온다
    Kitty k2 = new Kitty();
    System.out.println(k2.toString()); // 객체를 새로 만들면 다른 주소 값이 나온다
  }
}
```