

제11장

제어자와 다형성



1. 제어자(modifier)란?

■ 제어자의 개념

- 클래스, 멤버변수, 생성자, 메서드의 선언부에 사용되어 부가적 의미 부여(형용사)
- **제어자는 통상 접근 제어자와 기타 제어자로 나뉜다.**
- 하나의 대상에 제어자를 조합해서 사용할 수 있으나, 접근 제어자는 반드시 하나만 사용할 수 있다.

■ 제어자의 종류

- 접근 제어자 : public, protected, default, private
- 기타 제어자 : static, final, abstract, synchronized, transient 등
 - abstract : 추상클래스, 추상메서드 작성시 사용
 - synchronized : 동기화 메서드, 동기화 블록 작성시 사용
 - transient : 데이터 직렬화에 사용된다.

2. abstract – 추상적인, 미완성의

▣ abstract가 사용될 수 있는 곳

- 클래스, 메서드

클래스 앞에 붙을 때 : 클래스 내에 추상 메서드가 존재함을 의미한다.

메서드 앞에 붙을 때 : 선언부만 존재하고 구현부가 없는 추상 메서드임을 알린다.

```
abstract class Car {  
    // 추상 메서드  
    public abstract void method();  
}
```

* abstract class는 인스턴스를 생성할 수가 없다.

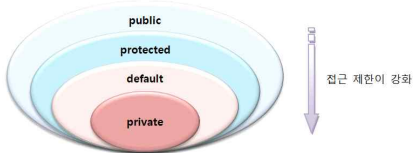
* abstract class는 상속에 의해서 완성이 되어야지만, 인스턴스를 생성하여 사용할 수가 있다.

3. 접근 제어자(access modifier)

■ 접근 제어자가 사용될 수 있는 곳

- 클래스, 멤버변수, 메서드, 생성자

■ 접근 제어자의 종류



접근 제한	적용 대상	접근할 수 없는 클래스
public	클래스, 필드, 생성자, 메소드	없음
protected	필드, 생성자, 메소드	자식 클래스가 아닌 다른 패키지에 소속된 클래스
default	클래스, 필드, 생성자, 메소드	다른 패키지에 소속된 클래스
private	필드, 생성자, 메소드	모든 외부 클래스

private : 같은 클래스 내에서만 접근 가능

default : 클래스를 선언할 때, **public**을 생략한 경우, 다른 패키지에서는 사용 불가

protected : 같은 패키지에서만 접근 가능, 상속 받지 않은 클래스도 접근 가능함. 단, 다른 패키지에서는 접근이 불가함.

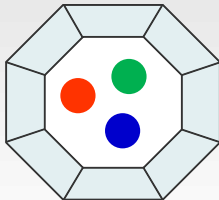
public : 어떤 클래스든 접근 가능 (다른 개발자가 사용할 수 있도록 라이브러리 클래스 만들 때 유용함)

4. 접근 제어자의 사용 이유

■ 사용 이유

- 외부로부터 데이터를 보호하기 위하여
- 외부에서 불필요한 접근, 내부적으로만 사용되는, 부분을 감추기 위해서

```
public class Phone {  
  
    private String model;  
    private String color;  
  
    public Phone(String model, String color) {  
        this.model = model;  
        this.color = color;  
    }  
  
    public String getModel() {  
        return model;  
    }  
    public String getColor() {  
        return color;  
    }  
}
```



상기 그림은 클래스의 캡슐화를 나타내고 있다.
하여, 자바는 보안에 매우 강한 언어이다.

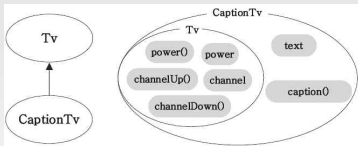
5. 다형성(polymorphism)

- 다형성(多形性)의 사전적 의미 : “여러 가지 형태를 취할 수 있는 능력
- P/G적 의미 : 하나의 참조변수로 여러 타입의 객체를 참조할 수 있는 것
즉, 조상의 참조변수로 자손타입의 객체를 다룰 수 있는 것이다.

```
class Tv {  
    boolean power; // 전원상태 (on/off)  
    int channel; // 채널  
  
    void power(){ power = !power;}  
    void channelUp(){ ++channel; }  
    void channelDown(){ --channel; }  
}  
  
class CaptionTv extends Tv {  
    String text; // 캡션내용  
    void caption() { /* 내용생략 */}  
}
```

```
Tv t = new Tv();  
CaptionTv c = new CaptionTv();
```

```
Tv t = new CaptionTv();
```



```
CaptionTv c = new CaptionTv();
```

```
Tv t = new CaptionTv();
```

위에 두개의 참조변수의 차이는 무엇일까?

6. 다형성의 원 타입의 개념

- 다형성은 조상타입의 참조변수로 자손타입의 객체를 다룰 수 있는 것.
- 하지만, 조상타입의 참조변수는 자손 타입을 다룰 수는 있지만, 절대 원래 타입은 벗어나지 못한다.(매우 중요함)

```
CaptionTv c = new CaptionTv();  
  
Tv t = new CaptionTv();  
  
class Tv {  
    boolean power; // 전원상태 (on/off)  
    int channel; // 채널  
  
    void power(){ power = !power;}  
    void channelUp(){ ++channel; }  
    void channelDown(){ --channel; }  
}  
  
class CaptionTv extends Tv {  
    String text; // 캡션내용  
    void caption() { /* 내용생략 */}  
}
```

* 두 개의 참조변수의 차이는 다룰 수 있는 멤버의 개수가 다른 것이다.(그 이유는 원 타입이 조상이기 때문)

7. 다형성의 참조

- ▣ 조상타입의 참조변수로 자손타입의 객체를 참조할 수 있지만, 반대는 허용이 안된다.
- ▣ 단, 메서드의 경우는 자손클래스에서 오버라이딩을 한 경우 자손클래스의 메서드가 호출된다.

```
public class Parent {  
    public void method() {  
        System.out.println("조상클래스 메서드");  
    }  
}
```

```
public class Child extends Parent {  
    //오버라이딩된 메서드 호출  
    @Override  
    public void method() {  
        System.out.println("자손클래스 메서드");  
    }  
}
```

```
Child child = new Child();  
//다형성 적용  
Parent parent = new Child();  
  
child.method();  
parent.method();
```

```
Console ☒  
<terminated> ChildExample [Java Appli  
자손클래스 메서드  
자손클래스 메서드
```


8. 다형성으로 인한 형변환

▣ 형변환의 전제 조건 – 상속, 구현관계에 있는 것만 객체타입 변환이 가능(매우중요)

▣ 자손타입에서 조상타입으로 형변환은 생략 가능하지만, 반대는 명시적 형변환을 반드시 해야 한다.

자손타입 → 조상타입 (Up-casting) : 형변환 생략가능 ← 조작 멤버갯수가 줄어들

자손타입 ← 조상타입 (Down-casting) : 형변환 생략불가 ← 조작 멤버갯수가 많아짐

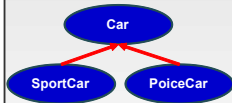
```
class Car {
    String color;
    int door;

    public void drive() {
        System.out.println("drive");
    }
    public void stop() {
        System.out.println("stop");
    }
}
class SportCar extends Car{
    public void speed() {
        System.out.println("speed");
    }
}
class policeCar extends Car{
    public void siren() {
        System.out.println("siren");
    }
}
```

```
public static void main(String[] args) {
    Car car = null;
    SportCar sc = new SportCar();
    SportCar sc2 = null;

    sc.speed();
    car = sc; //자손->조상(업캐스팅)
    car.speed(); //왜 예외가 발생할까?

    sc2 = (SportCar)car; //조상->자손(다운캐스팅)
    sc2.speed();
}
```



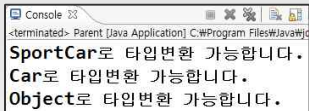
```
SportCar sportCar;
policeCar policeCar;
//서로 관계가 없는 상태는 형변환이
//성립되지 않는다.
sportCar = (SportCar)policeCar;
policeCar = (policeCar)sportCar;
```

9. instanceof연산자

- 참조변수가 참조하는 인스턴스의 실제 타입을 체크하는데 사용한다.
- 이항연산자, 연산결과는 boolean값이 된다.
- instanceof의 연산결과가 true이면, 해당 타입으로 형변환이 가능하다.

```
SportCar sportCar = new SportCar();

if(sportCar instanceof SportCar) {
    System.out.println("SportCar로 타입변환 가능합니다.");
}
if(sportCar instanceof Car) {
    System.out.println("Car로 타입변환 가능합니다.");
}
if(sportCar instanceof Object) {
    System.out.println("Object로 타입변환 가능합니다.");
}
if(sportCar instanceof PoiceCar) {
    System.out.println("PoiceCar로 타입변환 가능합니다.");
}
```



```
Console
<terminated> Parent [Java Application] C:\Program Files\Java\jdk-8.0.60\bin\java.exe
SportCar로 타입변환 가능합니다.
Car로 타입변환 가능합니다.
Object로 타입변환 가능합니다.
```

(중요) instanceof 연산자는 실제 형변환이 가능한지를 알아볼 때, 쓰는 것이 유용하다.

10. 참조변수와 인스턴스 변수의 연결

- 멤버변수가 중복 정의된 경우는 참조변수의 원래 타입에 따라 연결되는 멤버변수가 달라진다.(참조변수 원 타입에 영향을 받음)
- 메서드가 중복정의 된 경우는 참조변수의 타입에 관계없이 항상 실제 인스턴스의 타입에 정의된 메서드가 호출된다.(참조하고 있는 인스턴스에 영향을 받음)

```
class Parent {  
    int x = 100;  
  
    public void method() {  
        System.out.println("조상 메서드");  
    }  
}  
class Child extends Parent {  
    int x = 200;  
  
    @Override  
    public void method() {  
        System.out.println("자손 메서드");  
    }  
}
```

```
public static void main(String[] args) {  
    Parent p = new Child();  
    Child c = new Child();  
  
    System.out.println("p.x : " + p.x);  
    p.method();  
    System.out.println("c.x : " + c.x);  
    c.method();  
}
```

Console

<terminated> Main (5) [Java]

p.x : 100
자손 메서드
c.x : 200
자손 메서드

현업에서는 멤버변수에 직접 접근하는 경우가 거의 없다.
즉, 캡슐화가 되어있다는 것이다.
그래서 멤버 메서드를 이용해서 멤버변수의 값을 읽어온다.

11. 매개변수의 다형성

- ▣ 참조타입 매개변수는 메서드 호출 시, **자신과 같은 타입이거나 또는 자손타입의 주소를**
즉, 인스턴스를 넘겨주도록 한다.

```
class Product {  
    int price; // 제품가격  
    int bonusPoint; // 보너스점수  
}  
  
class Tv extends Product {}  
class Computer extends Product {}  
class Audio extends Product {}  
  
class Buyer { // 물건사는 사람  
    int money = 1000; // 소유금액  
    int bonusPoint = 0; // 보너스점수  
}
```

```
Buyer b = new Buyer();  
  
Tv tv = new Tv();  
Computer com = new Computer();  
  
b.buy(tv);  
b.buy(com);
```

```
Product p1 = new Tv();  
Product p2 = new Computer();  
Product p3 = new Audio();
```

```
void buy(Tv t) {  
    money -= t.price;  
    bonusPoint += t.bonusPoint;  
}
```

```
void buy(Product p) {  
    money -= p.price;  
    bonusPoint += p.bonusPoint;  
}
```

(중요) 여기서, 매개변수(Product p)의 개념을 잡는 것이 중요하다. 자신과 같은 타입이나 혹은 자손타입을 참조할 수 있다는 의미인 것을 분명히 이해하고 넘어가자.

12. 여러 종류의 객체를 배열로 다루기

■ 조상타입의 배열에는 조상 뿐만 아니라, 자손들의 객체도 담을 수 있다.(다형성)

```
Product p1 = new Tv();  
Product p2 = new Computer();  
Product p3 = new Audio();
```

```
Product p[] = new Product[3];  
p[0] = new Tv();  
p[1] = new Computer();  
p[2] = new Audio();
```

```
class Buyer { // 물건사는 사람  
    int money = 1000; // 소유금액  
    int bonusPoint = 0; // 보너스점수  
  
    Product[] cart = new Product[10]; // 구입한 물건을 담을 배열  
  
    int i=0;  
  
    void buy(Product p) {  
        if(money < p.price) {  
            System.out.println("잔액부족");  
            return;  
        }  
  
        money -= p.price;  
        bonusPoint += p.bonusPoint;  
        cart[i++] = p;  
    }  
}
```

감사합니다.

