제10장
상속과 오버라이딩

1. 상속(inheritance)이란?

▣ 상속의 개념

- **기존의 클래스를 재 사용하여 새로운 클래스를 작성하는 것.**
  - **관계없는 두 개 이상의 클래스를 조상(부모), 자손(자식)으로 직접적 관계를 만듦.**
  - **자손은 조상의 모든 멤버를 상속받음.(단, 생성자, 초기화 블록 제외)**
  - **자손의 멤버개수가 조상보다 적을 수가 없다.(같거나 많다.) – 다형성 개념 적용**

```
public class Bicycle {
    int id;
    String brand;
}
```

```
public class MoutainBike extends Bicycle {

}
```

많은 상속을 하다 보면, 멤버들이 늘어나는 것이 당연하다.
다른 클래스 생성의 기반이 된다. 그래서 베이스클래스, 기반 클래스 라고도 부른다.

```
public class MoutainBike {
    int id;
    String brand;
    String frame;
    int gear;
    String money;
}
```

→

```
public class MoutainBike extends Bicycle {
    //멤버변수가 5개이다.
    String frame;
    int gear;
    String money;
}
```

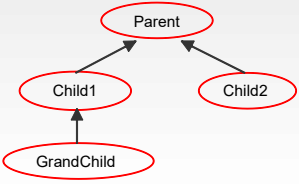
2. 클래스간의 관계(상속)

▣ 상속 관계 – 직접적 관계

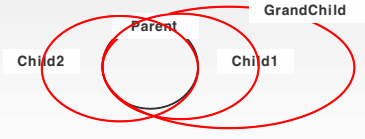
- 공통적으로 들어가는 부분은 조상에서 관리하고, 개별부분은 자손에서 따로 관리함.
- 조상의 변경은 자손의 변경을 일으킨다.(반대는 변경이 없다.)

```
class Parent{}
class Child1 extends Parent {}
class Child2 extends Parent {}
class GrandChild extends Child1 {}
```

* 상속 계층도(상속 관계도)



* 상속 다이어그램



3. 포함 관계(composite)

▣ 포함이란? → **다중 상속을 대체하는 방법**

- 클래스의 멤버변수로 다른 클래스를 선언하는 것.
- 규모가 적은 클래스를 먼저 만들고, 이것을 조합하여 규모가 큰 클래스로 만들어감.

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

```
class Circle {
    int x; // 원점의 x좌표
    int y; // 원점의 y좌표
    int r; // 반지름(radius)
}
```

→

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

원은 원점과 반지름으로 이루어진다. 하여, 따로 클래스를 만드는 것 보다, 위와 같이 먼저 Point클래스를 만들고, Circle클래스 안에 멤버변수로 사용하는 것이 코드의 재 사용성이 좋으며, 관리하기도 수월하다.

4. 상속이냐? 포함이냐?

▣ 이론적 방법 → 절대적인 것은 아니다. 단지 원론적인 얘기라고 생각하자.

- ‘is a’ 와 ‘has a’ 로 문장을 만들어보자.

  ex) Circle is a Point(미 성립)

  Circle has a Point(성립)

- 상속 관계 : ~은 ~이다.(is a)

  포함 관계 : ~은 ~을 가지고 있다.(has a)

▣ 현실적 방법

- 현업에서 개발 시, 만들어질 클래스에 영향을 가장 많이 주는 클래스는 상속하고 보조적인 것은 포함으로 돌려서 작성한다.

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

```
class Circle extends Point{
    int r; // 반지름(radius)
}
```

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

5. 상속과 포함 - 예제

▣ 원(Circle)은 도형(Shape)이다.(A Circle is a Shape.) : 상속관계

▣ 원(Circle)은 점(Point)를 가지고 있다.(A Circle has a Point.) : 포함관계

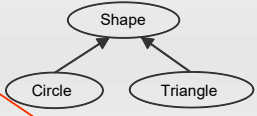
```
class Shape {
    String color = "blue";
    void draw() {
        // 도형을 그린다.
    }
}
```

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

    Point() {
        this(0,0);
    }

    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

```
class Circle extends Shape {
    Point center;
    int r;

    Circle() {
        this(new Point(0,0),100);
    }

    Circle(Point center, int r) {
        this.center = center;
        this.r = r;
    }
}
```

원점(0,0)과 반지름 100인 인스턴스 생성됨.

```
class Triangle extends Shape {
    Point[] p;

    Triangle(Point[] p) {
        this.p = p;
    }

    Triangle(Point p1, Point p2, Point p3) {
        p = new Point[]{p1,p2,p3};
    }
}
```




```
Circle c1 = new Circle();
Circle c2 = new Circle(new Point(150,150),50);

Point[] p = {new Point(100,100),
             new Point(140,50),
             new Point(200,100)
            };
Triangle t1 = new Triangle(p);
```

6. 단일 상속(single inheritance)

▣ 자바는 단일 상속만 허용한다.(C++은 다중 상속을 허용함)

▣ 다중 상속의 단점

- 조상클래스가 많아지며, 상속계층도 역시 복잡해진다.
- 클래스간에 관계를 관리하기 쉽지 않다.
- 조상클래스의 변수들로 인한 충돌도 배제할 수가 없다.

▣ 비중이 높은 클래스를 상속으로, 보조적인 클래스는 포함으로 사용한다.

7. Object 클래스 – 모든 클래스의 조상

▣ 사용자 정의 클래스(즉, 아무것도 상속을 받지 않는다면) 자동으로 Object를 상속받는다.

▣ 상속 계층도의 최상 위에 항상 있다.

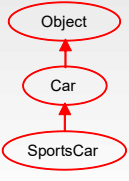
▣ 모든 클래스는 Object가 가지고 있는 11개의 메서드를 전부 상속 받는다.

ex) toString(), equals(), hashCode(), finalize() 등

```
public class Car {
    //...
}

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

컴파일 시

```
public class Car extends Object{
    //...
}

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



8. 오버라이딩(Overriding) - 재정의

▣ 조상클래스에서 상속받은 메서드를 자손한테 맞게끔 구현부를 수정하는 것

▣ 상속을 받을 때, 원하는 값만 상속을 받을 수 없다.하여, 필요하다면 오버라이딩을 한다.

* Override : '덮어쓰다' ,' ~에 우선하다' 등

```
public class Car {

    public void run() {
        System.out.println("차가 달립니다.");
    }
}

class SportsCar extends Car {

    @Override
    public void run() {
        System.out.println("스포츠 차가 달립니다.");
    }
}
```

9. 오버라이딩의 조건과 오버로딩과 비교

▣ 조건 : 반드시 메서드 선언부는 동일해야 한다.(리턴값, 메서드명, 매개변수)

구현부만 다르게 작성한다.

▣ 오버로딩과 오버라이딩의 비교

- 오버로딩 : 새로운 메서드를 만드는 것(new의 개념)
- 오버라이딩 : 선언부가 같고 구현부만 다른 것(modify, change의 개념)

```
public class Car {
    public void run() {
        System.out.println("차가 달립니다.");
    }
}
class SportsCar extends Car {
    //오버로딩
    public void run(int i) {
        System.out.println(i + "의 속도로 스포츠 차가 달립니다.");
    }
    @Override //오버라이딩
    public void run() {
        System.out.println("스포츠 차가 달립니다.");
    }
}
```

10. super – 조상클래스의 참조변수

▣ this – 인스턴스 자기 자신의 주소를 가지고 있는 참조변수와 같다.
또한, 지역변수와 인스턴스 멤버변수 구별한다.(변수의 모호성)

▣ super – 근본적으로 this와 같다. 조상의 멤버와 자신의 멤버를 구별 지을 때, 사용한다.

```
class Parent {
    int x = 20;
}

class Child extends Parent {
    int x = 100;

    public void method() {
        System.out.println("x = " + x);
        System.out.println("x = " + this.x);
        System.out.println("x = " + super.x);
    }
}
```

```
class Parent {
    int x = 20;
}

class Child extends Parent {

    public void method() {
        System.out.println("x = " + x);
        System.out.println("x = " + this.x);
        System.out.println("x = " + super.x);
    }
}
```

11. super - 예제

```
class Car {
    int speed = 80;

    public String run() {
        return "시속 : " + this.speed;

    }
}
class SportsCar extends Car {

    String mode = "auto";

    //오버라이딩 됨.
    @Override
    public String run() {
        //return "모드 : " + this.mode + "시속 : " + this.speed;
        return "모드 : " + this.mode + super.run();
    }
}
```

super는 근본적으로 this와 같다고 했다. 또한, 자손 클래스에서 명시적으로 조상 클래스의 메서드를 호출할 때도 사용할 수가 있다.

12. super() – 조상 클래스 생성자 호출

▣ 자손 클래스의 인스턴스를 생성하면, 자손과 조상의 멤버가 결합된 상태로 메모리에 할당된다.

▣ 조상의 멤버들도 반드시 초기화가 되어야 하기 때문에, 자손클래스의 생성자에서 첫 문장에 반드시 super()를 작성하여 조상 클래스의 생성자를 호출해야 한다.

▣ 조상 없는 자손이 있는가?(super()를 생략하면 컴파일러가 알아서 추가해 준다.)

```
class Car {
    int speed = 80;

    public Car() {
    }

    public String run() {
        return "시속 : " + this.speed;
    }
}
```

컴파일 시 →

```
class Car extends Object {
    int speed = 80;

    public Car() {
        //생략시 컴파일러가 알아서 추가해준다.
        super();
    }
    public String run() {
        return "시속 : " + this.speed;
    }
}
```

13. super(int x) – 매개변수가 있는 조상 생성자

```java
class Car {
    int speed;

    public Car(int speed) {
        this.speed = speed;
    }
}
class SportsCar extends Car {

    String mode;

    public SportsCar(String mode) {
        this.mode = mode;
    }
}
```

위의 코드에서 자손클래스 생성자에서는 왜 컴파일 에외가 발생할까? 바로, 조상클래스의 생성자를 호출하는 부분이 없다.

```java
class Car {
    int speed;

    //1번째 방법 : 기본생성자 추가
//  public Car() {
//  }

    public Car(int speed) {
        this.speed = speed;
    }
}
class SportsCar extends Car {

    String mode;

    public SportsCar(String mode) {
        //2번째 방법 : 매개변수 값을 지정함.
        super(80);
        this.mode = mode;
    }
}
```

위와 같이 2가지 해결 방법이 존재한다.

조상클래스에 기본 생성자 추가, 자손클래스에서 조상클래스의 매개변수가 있는 생성자를 호출하는 것이다.

감사합니다.