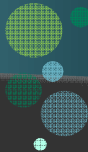


제11장

상속과 오버라이딩



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.

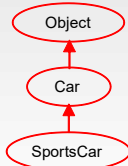
■ 모든 클래스는 Object가 가지고 있는 11개의 메서드를 전부 상속 받는다.

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

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

변파일 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 **this**와 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 기본적으로 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 추가해 준다.  
        super();  
    }  
  
    public String run() {  
        return "시속 : " + this.speed;  
    }  
}
```


13. super(int x) – 매개변수가 있는 조상 생성자

```
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
예외가 발생할까? 바로, 조상클래스의 생성자를 호출
하는 부분이 없다.

```
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 조상
클래스의 매개변수가 있는 생성자를 호출하는 것이다.

감사합니다.

