

09장. 객체지향 - 상속

1. 생성자

생성자는 인스턴스가 생성될 때 호출되는 '인스턴스 초기화'를 실행하며 클래스 멤버 변수의 초기화를 목적으로 한다.

생성자 역시 메소드처럼 클래스 내에 선언되며, 구조도 메소드와 유사하지만 다음과 같은 규칙을 가진다.

1. 생성자란 클래스이름과 동일하되 반환형은 없다.

public Test(){} ---> public void Test() {} X public int Test(){} X

void, dataType 등 선언하지 않는다.

2. 생성자는 객체를 생성할 때 자동으로 호출되며

목적은 멤버변수 초기화에 있다.

Test t1=**new Test()**;----->객체 생성시 명시 호출 된다. new 클래스명()이 new 생성자명();으로 호출되는 것이다.

3. 생성자도 접근 지정자를 가진다.

4. overload할 수 있으며 this()라는 키워드로 내부 호출 되어진다.

```
public Test(){ -----> default 생성자.
    this(30);    --->    public Test(int a)를 호출 한다.
}
public Test(int a){
```

다음은 Dog란 클래스로 생성자를 overload한 예이다.

```
class Dog {
    private String name;
        Dog () { -----> () 안에 매개 인자 없는 default 생성자
            name = "Anonymonus";
        }
        Dog (String aName) { -----> overload 된 생성자.
            name = aName;
        }
}
```

5. 생성자는 user 가 명시하지 않으면 default 생성자가

내부적으로 호출되어 메모리 할당 시 변수를 초기화 한다.

초기화는 다음과 같이 데이터 타입에 따라 지정된다.

정수	0
부동 소수점	0.0
char 형	'\u0000'
boolean 형	false
참조 형	null

6. 생성자를 하나라도 명시하게 되면 명시된 생성자를 호출하면서 메모리 할당 한다.

`public Test(int a) {} -----> Test t1=new Test(100);`

`public Test(int a, int b){} -----> Test t2=new Test(100,200);`

7. 생성자는 외부, 내부 메서드처럼 호출할 수 없고 객체 생성시 단 한번만 호출 되어 진다.

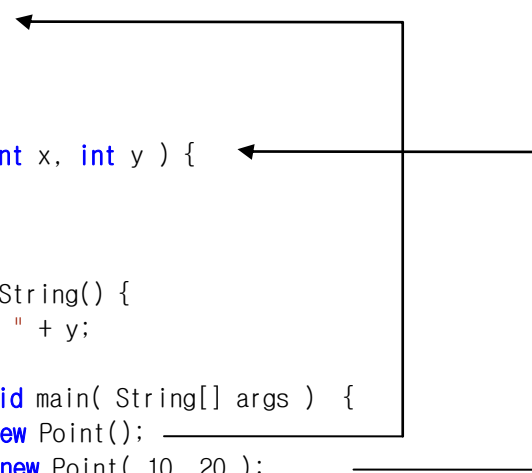
생성자의 장점은 멤버 변수를 초기화 할 인스턴스를 만들 때 인스턴스 변수의 값을 지정할 수 있다는 점이다. 즉, 인스턴스화 하는 객체마다 고유 한 초기 값을 가질 수 있다. 클래스를 사용하는 다른 클래스에서 보면 전용 초기화 메소드를 일부러 호출 할 필요가 없다

다음 프로그램을 살펴 보자

X, Y 좌표를 이용하여 위치를 리턴 하는 Point 클래스이다

한번은 기본값 0,0 의 좌표를 리턴 하고 한번은 사용자가 좌표값을 10,20 으로 지정해서 좌표표시를 한다음 값을 리턴 받고 싶은 상황이다.

```
public class Point {  
    int x;  
    int y;  
    public Point() {  
        x = 0;  
        y = 0;  
    }  
    public Point ( int x, int y ) {  
        this.x = x;  
        this.y = y;  
    }  
    public String toString() {  
        return x + " " + y;  
    }  
    public static void main( String[] args ) {  
        Point pt = new Point();  
        Point pt2 = new Point( 10, 20 );  
        System.out.println( pt.toString() );  
    }  
}
```



```

        System.out.println( pt2.toString() );
    }
}

```

[실행결과]

```

0 0
10 20

```

객체를 pt 로 생성하게 되면 default 생성자 `public Point()` 를 호출하면서 객체가 생성되어 x,y 에 각 0 을 대입되어 생성되며 pt2 는 `public Point (int x, int y)` 를 호출하면서 객체가 생성되어 10, 20 을 x, y 에 전달하게 되면서 객체가 생성된다.

2. 상속(inheritance)

객체지향개념의 가장 중요한 코드의 재사용을 프로그램으로 구현하는 부분이다.

새로운 클래스를 생성할 때 처음부터 새로 작성하는 것이 아니라, 기존에 정의된 클래스로부터 중복되는 부분을 물려받아서 사용할 수 있다. 이렇게 되면 우리는 새로운 클래스에서 추가되는 부분만 고심해서 프로그래밍하면 된다.

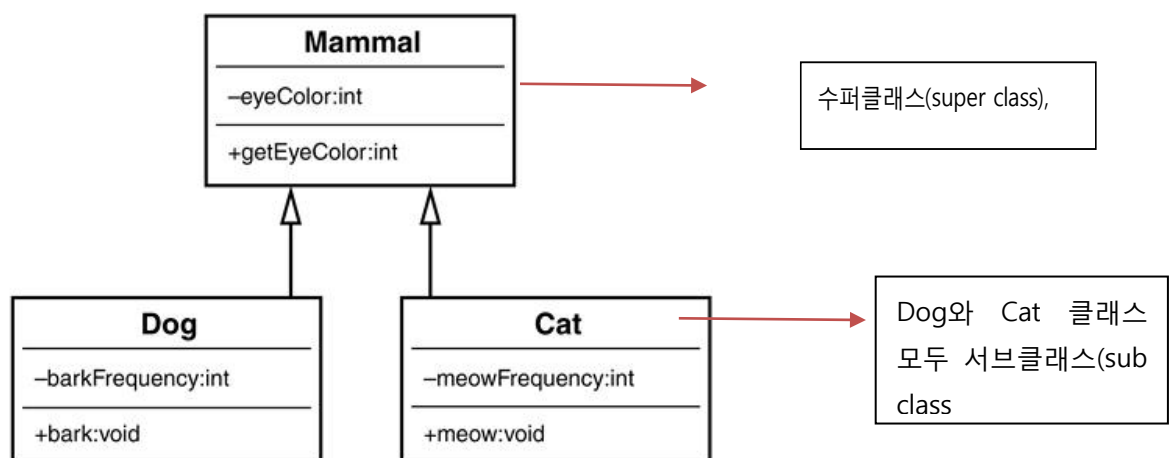
프로그램을 전체를 새로 개발한 것이 아니므로 프로그램 개발기간의 단축과, 비용의 감소를 가져올 수 있고, 이미 실 시스템에서 안정적으로 돌아가는 코드부분을 사용하기 때문에 시스템이 더 안정적이다. 새로 작성해서 기존의 코드를 가져다 쓰는 클래스가 상속을 받는 클래스

-> 서브클래스(sub class)또는 자식클래스(child class) 또는 파생클래스(derived class)

기존에 이미 만들어져서 상속해주는 클래스

->슈퍼 클래스(super class)또는 베이스클래스(base class)또는 부모클래스(parent class)

자바 프로그램에서 상속은 **extends**라는 키워드를 써서 표현한다.

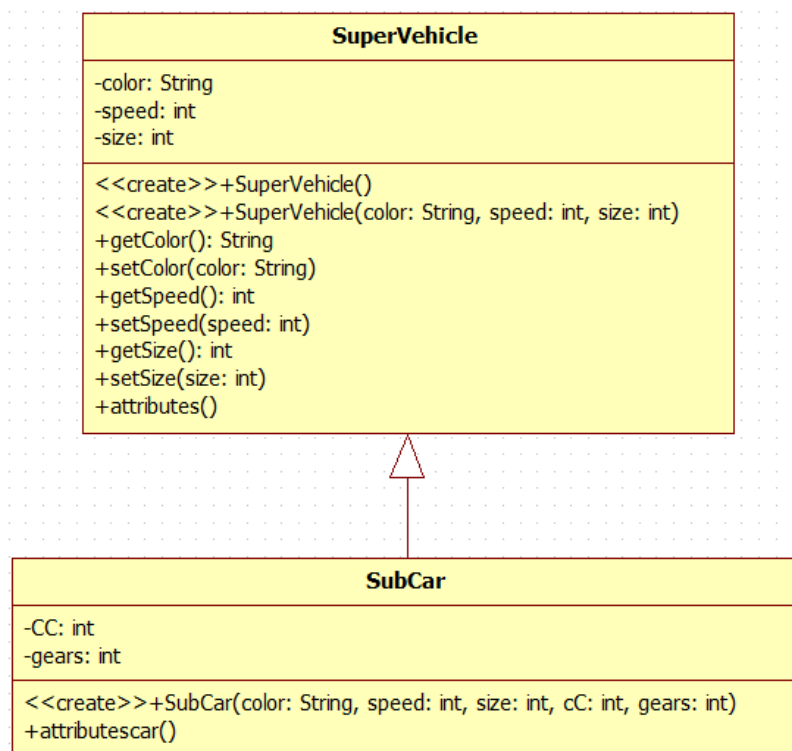


```
class Dog extends Mammal{ //코드 생략 }
class Cat extends Mammal{ //코드 생략 }
```

상속시 특징은 다음과 같다.

- ① 상속은 단일 상속을 원칙으로 한다 즉, 후손이 슈퍼 클래스는 하나 밖에 가질 없다
- ② 서브 클래스는 몇 단계로 만들 수 있다
- ③ 멤버 변수, 메소드는 상속됩니다.
- ④ 그러나, 후손클래스의 변수와 메소드는 사용할 수 없다. 즉, 인스턴스화 한 클래스가 슈퍼 클래스에서 상속 된 기능 밖에 사용할 수 없다.
- ⑤ 생성자는 상속되지 않는다. 클래스 고유의 것이다
- ⑥ 후손의 멤버 변수 이름, 메소드 이름이 겹치면 인스턴스화 된 클래스에서 가장 계층이 가까운 선조 클래스의 메소드가 호출된다. 즉, 상속시에 기존 멤버 변수 방법을 변경할 수 있다.
- ⑦ 후속이 선조의 멤버 혹은 생성자를 호출할 때 super.멤버, super()로 호출한다.

다음 프로그램을 보고 상속에 대한 구문을 살펴 보자.



차량에 관한 슈퍼 클래스로 SuperVehicle 이 색상, 스피드, 사이즈 등을 관리하는 클래스가 있다면 배기량(cc)와 기어만 가지는 SubCar 클래스는 상속을 받게 되면

SuperVehicle 가 가진 메소드를 자유롭게 호출해서 사용할 수 있다.<<
Testinheritance.java>>

```
class SuperVehicle { -----> 선조 클래스
    private String color;
    private int speed;
    private int size; -----> 멤버 변수 3개를 private로 은닉화 선언

    public SuperVehicle() { --> 기본생성자
        super();
    }

    public SuperVehicle(String color, int speed, int size) {
        -----> 주어진 값으로 초기화
        super();
        this.color = color;
        this.speed = speed;
        this.size = size;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }

    public int getSpeed() {
        return speed;
    }

    public void setSpeed(int speed) {
        this.speed = speed;
    }

    public int getSize() {
        return size;
    }

    public void setSize(int size) {
        this.size = size;
    }

    public void attributes() {-----> 내용을 출력
        System.out.println("Color : " + color);
    }
}
```

```

        System.out.println("Speed : " + speed);
        System.out.println("Size : " + size);
    }
}

```

```

class SubCar extends SuperVehicle { -----> 상속관계로 후손 클래스 선언
    private int CC;
    private int gears; -----> 멤버변수 선언

```

```

    public SubCar(String color, int speed, int size, int cC, int gears) {
-----> 총 5개의 값을 받아 3개는 선조에게 super 키워드를 통해 전달하면
        선조의 생성자인 public SuperVehicle(String color, int speed, int size)이
호출되어 초기값을 지정하면서 선조 클래스의 메모리에 생성되고
        2개의 값은 후손클래스의 멤버에게 전달하고 객체가 메모리에 생성된다
        super(color, speed, size);
        CC = cC;
        this.gears = gears;
    }

```

```

    public void attributescar() {
--> 선조의 멤버변수가 은닉화 되어 있어 리턴형 메소드를 호출 한다
getColor(),getSpeed(), getSize()를 호출한다.
        System.out.println("Color of Car : " + getColor());
        System.out.println("Speed of Car : " + getSpeed());
        System.out.println("Size of Car : " + getSize());
        System.out.println("CC of Car : " + CC);
        System.out.println("No of gears of Car : " + gears);
    }
}

```

```

public class Testinheritance{
    public static void main(String args[]) {
        SubCar b1 = new SubCar("Blue",200,22,1000,5);
-----> 후손의 객체를 생성하면서 5개의 값을 전달하게 되면 public SubCar(String
color, int speed, int size, int cC, int gears) 이 호출되어 선조와 후손의 객체를
생성하게 된다.
        b1.attributescar();
    }
}

```

[출력 결과]

```

Color of Car : Blue
Speed of Car : 200
Size of Car : 22
CC of Car : 1000
No of gears of Car : 5

```

3. 오버라이딩(overriding)

>> 오버라이딩(overriding)

오버라이딩은 상속관계에서 발생하는데, 슈퍼클래스에서 상속받은 메소드를 그대로 사용하는 것이 아니라, 서브클래스에 맞도록 메소드의 내용을 수정해서 사용하는 것을 말한다.

슈퍼 클래스와 서브클래스에 같은 이름을 가진 메소드가 존재할 때, 슈퍼 클래스의 메소드를 무시하고(override), 서브클래스의 메소드를 사용하는 것으로, 메소드를 재 정의하는 것이다.

➤ 오버라이딩시 주의점

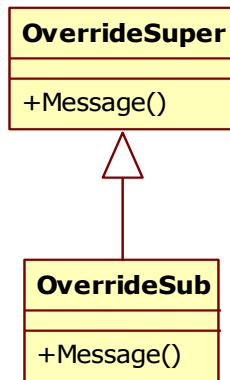
- 첫 번째 슈퍼클래스의 메소드를 오버라이딩하는 서브클래스의 메소드는, 메소드의 시그니처는 물론 리턴 타입까지 일치해야 한다. 일치하지 않으면 오버라이딩이 되지 않는다.
- 두 번째 서브클래스에서 재정의 되는 메소드의 접근제어자는 슈퍼클래스의 오버라이드되는 메소드의 접근제어자와 같거나, 제한범위가 넓은 접근제어자를 사용한다.

오버라이딩과 오버로딩은 다음과 같은 차이를 가진다.

오버라이딩과 오버로딩의 차이

오버라이딩(재정의)	구분	오버로딩(다중정의)
상속관계	적용	특정 클래스
super 클래스의 메서드보다 sub 클래스의 메서드 접근제한이 동일하거나 더 넓어야 한다. 예를 들어, protected라면 protected/public이다.	접근제한	상관없다.
기본적으로 같아야 한다.	리턴형	상관없다.
super 클래스의 메서드명과 sub 클래스의 메서드명이 같아야 한다.	메서드명	반드시 같아야 한다.
반드시 같아야 한다.	인자(타입, 개수)	반드시 달라야 한다.

다음프로그램을 살펴 보자. << TestOverride.java>>



-----> 선조와 후손의 같은 Message()메소드를 가지고 있다.

```

class OverrideSuper {
    public void Message () {
        System.out.println ( "부모의 메시지");
    }
}
class OverrideSub extends OverrideSuper {
    public void Message () {
        System.out.println ( "아이 메시지");
    }
}

public class TestOverride {
    public static void main (String [] args) {
        OverrideSub objsub = new OverrideSub ();
        objsub.Message() ; -----> 선조의 메소드를 후손이 재정의한 상태로
후손의 메소드가 호출된다.
    }
}
  
```

[실행결과]
아이 메시지

>> 클래스간의 형변환

기본 데이터 형은 형변환은 변수끼리 대입 할 경우와 불가능한 경우가 있었다. 확장형 변환이라면 자동 형 변환되었지만 축소 형 변환은 명시 적 캐스트가 필요했다.

참조 형식 변수는 메모리의 상태를 참조하는 ID가 할당되어 있다 즉 주소값을 받아 참조한다. 참조 형 변수의 대입은 참조를 식별 ID의 복사본이다. 참조 형 변수의 경우는 같은 클래스 형이면 문제없이 할당 할 수 있지만 클래스 형이 다르면 할당 할 수 없다. 단, 상속 관계에 있는 클래스 형의 경우는 형변환을 통하여 대입 할 수 있다.

① 수퍼 클래스 = 서브 클래스

서브 클래스 형 객체 참조는 슈퍼 클래스 형의 변수에 할당 할 수 있다. 자동으로 타입 변환된다. 서브 클래스는 슈퍼 클래스의 정의를 계승하고 있다. 이 서브 클래스를 인스턴스화하면 슈퍼 클래스에 정의 된 부분에 대한 인스턴스도 만들어진다.. 즉, 하위 클래스의 인스턴스는 슈퍼 클래스의 정의에 대한 인스턴스도 포함되어있는 것이다. 그래서 서브 클래스의 객체를 참조하는 변수는 슈퍼 클래스 형 변수에 할당 할 수 있다

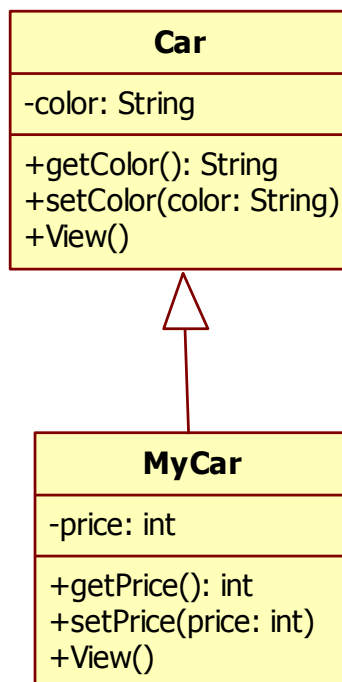
② 서브 클래스 = 슈퍼 클래스

슈퍼 클래스 형 오브젝트의 참조는 서브 클래스 형의 변수에 대입 할 수 없다 일반적으로 슈퍼 클래스 형 오브젝트는, 서브 클래스에서 직접 구현하는 정의에 해당하는 인스턴스를 가지지 않기 때문이다. 대입 하는 경우에는 캐스트가 필요하다

③ 캐스팅

슈퍼 클래스 형 서브 클래스 형의 참조를 대입하는 것은 자주 사용한다. 하위 클래스에서 메서드를 재정의하는 경우는 슈퍼 클래스 형으로 변환되어 있어도 서브 클래스에서 재정의되는 메서드의 구현으로 동작한다.

다음 구조를 살펴 보자. 선조 클래스와 후손 클래스를 이용하여 객체를 생성한 후 캐스팅한 값을 구현할 수 있다. << TestCasting.java>>



Car c = **new** Car(); -->Car 클래스 생성

MyCar my_car = **new** MyCar(); -→ Car클래스와 MyCar클래스 생성
c = my_car; --→ 슈퍼 클래스 = 서브 클래스

my_car = (MyCar) c; -----→ 캐스팅, 캐스팅하지 않으면 후손의 메소드에 접근 할 수 없다

```
if(my_car instanceof Car){ → Car의 객체가 맞는지 확인 후 메소드로 접근.  
    my_car.setColor("Yellow");  
    my_car.setPrice(50000);  
    my_car.View();  
}
```