



java 9강 - 다형성

양 명 속

[now4ever7@gmail.com]



목차

- 다형성
- 참조변수의 형변환
- instanceof

- 매개변수의 다형성
- 여러 종류의 객체를 하나의 배열로 다루기



객체지향언어의 3대 특징

- 객체 지향 언어의 3대 특징
 - 캡슐화(은닉성)
 - 상속성
 - 다형성
- [1] 캡슐화(은닉성)
 - 클래스 내부에서 노출해야 되는 최소한의 부분을 제외한 나머지를 숨기는 특징
 - 필요한 기능만 노출하고 나머지를 감추는 것
 - 구성요소와 행위가 객체에 의해서 포장되어 있음



객체지향언어의 3대 특징

■ [2] 상속성

- 객체지향언어 – 클래스를 만들어 놓고, 필요할 때 객체를 생성해서 사용하기만 하면 됨, 한 번 만들어 놓으면 재사용이 용이
- 상속성 – 상위 클래스의 구성요소, 행위를 그대로 물려받아 사용하고, 자신만의 구성요소와 행위는 추가해서 사용
- 예) 사람 클래스 – 남자 클래스, 여자 클래스로 구분하여 생성
 - 남자 클래스도 사람클래스의 보다, 숨쉬다, 말하다를 똑같이 만들어야 되는 경우
 - 사람 클래스 밑에 오는 클래스는 사람 클래스의 구성요소와 행위를 그대로 불러서 사용할 수 있게 하는 것 – 상속



객체지향언어의 3대 특징

■ [3] 다형성

- 같은 행위를 상속 받았지만, 방식이 다를 때는 다시 정의해서 사용하는 것 즉, 재정의(오버라이딩)를 통해서 다형성을 보장해줌
- 부모 클래스 타입의 참조변수로 자식 클래스의 인스턴스를 참조할 수 있도록 함으로써 다형성을 구현



다형성(polymorphism)

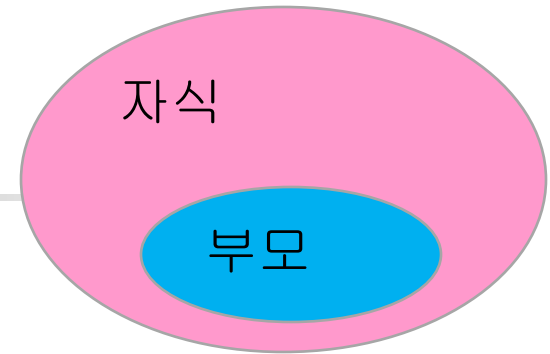
■ 다형성이란

- 여러가지 형태를 가질 수 있는 능력을 의미함
- 한 타입의 참조변수로 여러 타입의 객체를 참조할 수 있도록 함으로써 다형성을 구현
- 부모 클래스 타입의 참조변수로 자식 클래스의 인스턴스를 참조할 수 있도록 함으로써 다형성을 구현
- 여러 개의 개별적인 클래스를 하나의 부모 클래스 객체로 통합 관리하여 그 효율성을 높인 것

예제

```
class Parent
{
    public void parentFunc()
    {
        System.out.println("나는 부모클래스야");
    }
    public void display()
    {
        System.out.println("부모 display()메서드");
    }
}
class Child extends Parent
{
    public void display()
    {
        System.out.println("자식 display()메서드");
    }
    public void childFunc()
    {
        System.out.println("나는 자식클래스야");
    }
}
```

예제



```
class CastingTest
{
    public static void main(String[] args)
    {
        Parent p = new Child();
        p.parentFunc(); //Parent 클래스의 ParentFunc() 호출
        p.display(); //Child 클래스의 오버라이딩된 display() 호출
        //p.childFunc(); //에러
    }
}
```




다형성

- Parent와 Child 클래스의 인스턴스를 생성하고 사용하기 위해서는 인스턴스의 타입과 일치하는 타입의 참조변수만을 사용했다

```
Parent p = new Parent();  
Child c = new Child();
```

- 서로 상속관계에 있을 경우, 부모 클래스 타입의 참조변수로 자식 클래스의 인스턴스를 참조하도록 하는 것도 가능

```
Parent p = new Child();
```

다형성

- 같은 타입의 참조변수로 참조하는 것과 부모 타입의 참조변수로 참조하는 것의 차이

```
Parent p = new Child();  
Child c = new Child();
```

- **Parent** 타입의 참조변수로 **Child** 인스턴스 중에서 **Parent** 클래스의 멤버들(상속받은 멤버 포함)만 사용할 수 있음
- 단, 오버라이딩된 메서드의 경우는 **자식의 오버라이딩 메서드** 사용
- **Parent** 클래스에 정의되지 않은 멤버는 사용 불가능
예) p.childFunc(); //불가

- 반대로 자식 타입의 참조변수로 부모 타입의 인스턴스를 참조하는 것은 불가능

```
Child c = new Parent(); //불가
```



다형성

- 부모 타입의 참조변수로 자식 타입의 인스턴스를 참조할 수 있다.
 - 이때는 자식의 오버라이딩 메서드가 호출됨
- 반대로 자식 타입의 참조변수로 부모 타입의 인스턴스를 참조할 수는 없다.



다형성

- 다형적인 표현에서 멤버에 대한 규정
 - 부모의 참조변수에 자식 클래스의 인스턴스를 대입했을 때
 - 부모 타입의 참조변수로는 자식 인스턴스 중에서 부모 클래스의 멤버들(상속 받은 멤버포함)만 사용할 수 있음
 - 단, 자식 클래스에서 메서드 오버라이딩을 했다면 오버라이딩된 자식 클래스의 메서드가 실행됨
- 부모 자식 객체 간의 타입변환
 - 객체의 집합을 관리하는데 편리함
 - 부모 타입이 파생된 모든 자식 타입을 가리킬 수 있으므로 부모 타입의 변수로 모든 자식 타입을 일관되게 관리할 수 있음



예제 1-다형성

```
import java.util.Scanner;
class Shape {
    public void draw() {
        System.out.println("모양을 그립니다");
    }
    public void delete() {
        System.out.println("모양을 지웁니다");
    }
    public void display() {
        System.out.println("부모 - Shape");
    }
}
class Circle extends Shape {
    public void draw() {
        System.out.println("원을 그립니다");
    }
    public void delete() {
        System.out.println("원을 지웁니다");
    }
    public void sayCircle() {
        System.out.println("안녕하세요 원입니다");
    }
}
```



예제 1-계속

```
class Triangle extends Shape {  
    public void draw() {  
        System.out.println("삼각형을 그립니다.");  
    }  
    public void delete() {  
        System.out.println("삼각형을 지웁니다");  
    }  
    public void sayTriangle() {  
        System.out.println("안녕하세요 삼각형입니다");  
    }  
}
```

예제 1 - 계속

```
class Upcasting {
    public static void main(String[] args) {
        Shape s = new Shape();
        s.draw();
        s.delete();
        System.out.println();

        //클래스의 기본적인 사용법
        Circle c = new Circle();
        c.draw();
        c.delete();
        c.sayCircle();
        System.out.println();

        //다형성 이용
        System.out.println("-----Upcasting---");
        Shape c1 = new Circle();
        c1.draw(); //오버라이딩한 메서드
        c1.delete();
        c1.display(); //부모로 부터 상속받은 멤버
        //c1.sayCircle(); //에러-자식의 멤버는 접근 불가
        System.out.println();

        Shape t1 = new Triangle();
        t1.draw();
        t1.delete();
        //t1.sayTriangle(); 에러
        System.out.println();
    }
}
```

```
모양을 그립니다
모양을 지웁니다

원형을 그립니다
원형을 지웁니다
안녕하세요 원입니다

-----Upcasting---
원형을 그립니다
원형을 지웁니다
모 - Shape

삼각형을 그립니다.
삼각형을 지웁니다
```

예제 1- 계속

사용자로부터 원, 삼각형 중 입력 받아서
해당 객체 생성하기

```
Scanner sc = new Scanner(System.in);
System.out.println("도형을 선택하세요(1:원, 2: 삼각형)");
int s1 = sc.nextInt();
```

```
Shape obj=null;
if (s1 == 1)
{
    obj = new Circle();
}
else if (s1 == 2)
{
    obj = new Triangle();
}
else
{
    System.out.println("잘못 선택했습니다!");
    return;
}
obj.draw();
obj.delete();
```

```
도형을 선택하세요<1:원, 2: 삼각형>
2
삼각형을 그립니다!
삼각형을 지웁니다.
```




메서드 이용

```
//-----메서드 이용
System.out.println("도형을 선택하세요(1:원, 2: 삼각형)");
s1 = sc.nextInt();
```

```
public static Shape createShape(int s)
{
    Shape obj = null;
    if (s == 1)
    {
        obj = new Circle();
    }
    else if (s == 2)
    {
        obj = new Triangle();
    }
    return obj;
}
```

1, 2 중 하나를 매개변수로 받아서 1 이면 **Circle** 객체를 생성해서 리턴하고,
2 이면 **Triangle** 객체를 생성해서 리턴하는 메서드

다형성을 이용하지 않으면, 기존 방식으로는 반환형을 지정할 수가 없다

예제1- 계속

배열 이용

//똑같은 호출문이라도 변수가 실행 중에 가리키는 타입에 따라

//실제 동작이 달라질 수 있는 다형성이 성립함

```
System.out.println("-----다형성-----");
```

```
Shape[] sh = {new Circle(), new Triangle(), new Rectangles()};
```

```
for (int i=0; i<sh.length;i++ )
```

```
{
```

```
    sh[i].draw(); //호출하기만 하면 알아서 적당한 메서드가 호출되어 동작함
```

```
}//for
```

```
}
```

```
}
```

```
Shape s1 = new Circle();
```

```
Shape s2 = new Triangle();
```

```
Shape s3 = new Rectangles();
```

```
Shape s[] = new Shape[3];
```

```
s[0] = new Circle();
```

```
s[1] = new Triangle();
```

```
s[2] = new Rectangles();
```

```
-----다형성-----  
원을 그립니다.  
삼각형을 그립니다.  
사각형을 그립니다.
```



예제2-다형성

```
class Polymo{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("1. 고양이, 2. 강아지, 3. 소 중에서 하나 선택!!");
        int type = sc.nextInt();

        Animal ani=null;
        switch (type){
            case 1 : ani = new Cat();
                    break;
            case 2: ani=new Dog();
                    break;
            case 3: ani=new Cow();
                    break;
            default:
                System.out.println("잘못 입력함!!!");
                return;
        }//switch

        //자식의 오버라이딩된 메서드 호출
        ani.bark();
    }
}
```



예제2-다형성 계속

```
        System.out.println("=====예제2");
        Animal obj = Polymo(2); // Polymo(1), Polymo(3)
        obj.bark();
    }

    static Animal Polymo(int i) //예제2
    {
        Animal obj = null;
        switch(i){
            case 1: //부모를 참조하는 참조변수는 자식을 참조할 수 있다 - 다형성 구현됨
                    obj = new Cat();break;
            case 2:
                    obj = new Cow();break;
            case 3:
                    obj = new Dog();break;
        }//switch

        return obj;
    }
}
```



예제2-다형성 계속

```
class Cat extends Animal
{
    public void bark()
    {
        System.out.println("야옹야옹");
    }
    public void child()
    {
        System.out.println("난 자식-고양이");
    }
}
class Cow extends Animal
{
    public void bark()
    {
        System.out.println("음메음메");
    }
}
class Dog extends Animal
{
    public void bark()
    {
        System.out.println("멍멍");
    }
}
```

```
class Animal
{
    public void bark(){
        System.out.println("울다");
    }
    public void parent(){
        System.out.println("난 부모-동물");
    }
}
```



실습

- 차 클래스(부모)
 - 메서드
 - 엔진
 - 차종
- 에쿠스 클래스- Equus
 - 메서드 오버라이딩
 - 엔진 - 에쿠스 엔진
 - 차종 - 에쿠스
- 벤츠 클래스- Benz
 - 메서드 오버라이딩
 - 엔진 - 벤츠 엔진
 - 차종 - 벤츠
- main() 메서드에서 사용자로부터 차종을 입력 받고, 엔진과 차종을 출력

```
차종을 선택하세요(1:에쿠스, 2:벤츠)
2
벤츠!
벤츠 엔진
```



배열 이용

```
class PolymoArray {  
    public static void main(String[] args) {  
        //Shape 배열에 Shape 객체 넣기  
        Shape sh1=new Shape();  
        Shape sh2=new Shape();  
        Shape sh3=new Shape();  
  
        Shape[] shArr = new Shape[3];  
        shArr[0] = sh1;  
        shArr[1] = new Shape();  
        shArr[2] = new Shape();  
  
        //다형성 이용 - 부모타입의 배열에 자식 인스턴스를 넣자  
        Shape sh4 = new Shape();  
        Circle c1 = new Circle();  
        Triangle t1 = new Triangle();  
        Shape c2 = new Circle();  
    }  
}
```



배열 이용

//부모 배열 선언

```
Shape[] shArr2 = new Shape[5];
```

```
shArr2[0] = sh4;
```

```
shArr2[1] = c1;
```

```
shArr2[2] = new Triangle();
```

```
shArr2[3] = new Circle();
```

```
shArr2[4] = c2;
```

```
for(int i=0;i<shArr2.length;i++)  
{
```

```
    Shape s= shArr2[i];
```

```
    s.draw();
```

```
    s.delete();
```

```
    //shArr2[i].draw();
```

```
    //shArr2[i].delete();
```

```
}//for
```

```
}
```

```
}
```


다형성-배열 이용

//기존의 클래스로부터 공통된 부분을 뽑아내어 부모클래스를 만들어 보자

```
class Marine{                                //보병
    int x, y;                                //현재 위치
    void move(int x, int y){/* 지정된 위치로 이동 */}
    void stop(){/* 현재 위치에 정지 */}
    void stimPack(){/* 스팀팩을 사용한다 */}
}
```

• **Starcraft**에 나오는 유닛들을 클래스로 정의
이 유닛들은 각자 나름대로의 기능을 가지고 있지만
공통부분을 뽑아내어 하나의 클래스로 만들고,
이 클래스로부터 상속받도록 변경

```
class Tank{                                  //탱크
    int x, y;                                //현재 위치
    void move(int x, int y){/* 지정된 위치로 이동 */}
    void stop(){/* 현재 위치에 정지 */}
    void changeMode(){/* 공격모드를 변환한다 */}
}
```

```
class Dropship{                              //수송선
    int x, y;                                //현재 위치
    void move(int x, int y){/* 지정된 위치로 이동 */}
    void stop(){/* 현재 위치에 정지 */}
    void load(){/* 선택된 대상을 태운다 */}
    void unload(){/* 선택된 대상을 내린다 */}
}
```

```

class Unit{
    protected int x, y;          //현재 위치
    public void move(int x, int y){} //Unit 클래스를 상속받아서 작성되는 클래스는
    //move 메서드를 자신의 클래스에 알맞게 구현해야 한다
    public void stop(){/* 현재 위치에 정지 */}
}

```

```

class Marine extends Unit{      //보병
    public void move(int x, int y){
        /* 지정된 위치로 이동 */
        this.x=x;
        this.y=y;
        System.out.println("보병이 "+x+", "+y+"위치로 이동한다.");
    }
    public void stimPack(){/* 스팀팩을 사용한다 */}
}

```

```

class Tank extends Unit{        //탱크
    public void move(int x, int y){
        this.x=x;
        this.y=y;
        System.out.println("탱크가 "+x+", "+y+"위치로 이동한다.");
    }
    public void changeMode(){/* 공격모드를 변환한다 */}
}

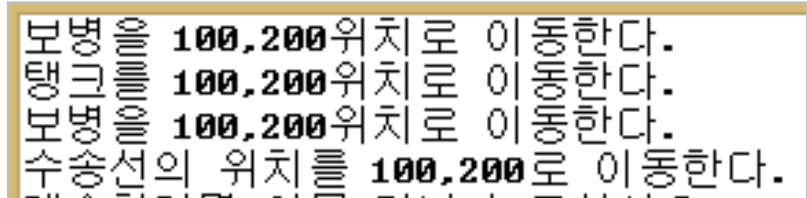
```

- 각 클래스의 공통부분을 뽑아내서 **Unit** 클래스를 정의하고 이로부터 상속받도록 함
- 이 **Unit** 클래스는 다른 유닛을 위한 클래스를 작성하는데 재활용될 수 있다
- **Marine, Tank**는 지상유닛이고, **Dropship**은 공중 유닛이기 때문에 이동하는 방법이 서로 달라서 **move** 메서드의 실제 구현 내용이 다름

```

class Dropship extends Unit{
    //수송선
    public void move(int x, int y){
        this.x=x;
        this.y=y;
        System.out.println("수송선의 위치를 "+x+", "+y+"로 이동한다.");
    }
    public void load(){/* 선택된 대상을 태운다 */}
    public void unload(){/* 선택된 대상을 내린다 */}
}

```



보병을 100,200위치로 이동한다.
 탱크를 100,200위치로 이동한다.
 보병을 100,200위치로 이동한다.
 수송선의 위치를 100,200로 이동한다.

```

class UnitTest {
    public static void main(String[] args) {
        Unit[] group = new Unit[4];
        group[0]=new Marine();
        group[1]=new Tank();
        group[2]=new Marine();
        group[3]=new Dropship();

        for (int i=0;i<group.length ;i++ ) {
            group[i].move(100, 200);
            //Unit 배열의 모든 유닛을 좌표(100, 200)의 위치로 이동한다
        }
    }
}

```

예제-여러 종류의 객체를 하나의 배열로 다루기

- 10개의 도형을 입력 받는다.
 - 그 도형은 원과 사각형 중 어느 것이어도 됨
 - 사용자가 원하는 도형을 입력할 수 있도록 함
- 입력 받는 도중에 사용자가 현재까지 입력된 도형을 보려는 경우 보여 주어야 함
- 언제든지 프로그램은 종료될 수 있어야 함

```
1.원 2.사각형 3.보기 4.종료 ==> 1
r = 10
1.원 2.사각형 3.보기 4.종료 ==> 2
w = 3
h = 5
1.원 2.사각형 3.보기 4.종료 ==> 1
r = 7
1.원 2.사각형 3.보기 4.종료 ==> 3
----- 보기 -----
원의 면적 : 314.0
사각형의 면적 : 15
원의 면적 : 153.86
-----
1.원 2.사각형 3.보기 4.종료 ==> 4
프로그램을 종료합니다.
```



예제-여러 종류의 객체를 하나의 배열로 다루기

■ 1단계

- main()에서 사용자로부터 원, 사각형, 종료 중 선택하게 하고,
- 원을 선택하면 반지름을 입력 받아서 원의 객체 생성
- 사각형을 선택하면 가로,세로를 입력 받아서 사각형의 객체 생성
- 보기를 선택하면, 면적을 구하는 메서드를 호출하여 모든 객체의 면적 출력
- 종료를 선택하면 메서드 종료(return)
- 단, 다형성을 이용하여 구현할 것

■ 2단계

- 10번 반복하여 사용자로부터 입력 받을 것

■ 3단계

- 배열을 이용하여 부모배열에 자식 객체들을 저장



예제-여러 종류의 객체를 하나의 배열로 다루기

- Shape 클래스-부모
 - 메서드
 - findArea() – “도형의 면적을 구한다” 출력
- Circle 클래스
 - 필드 : 반지름
 - 생성자
 - 반지름 초기화
 - 메서드
 - findArea() – 원의 면적 구한후 출력 (반지름*반지름*3.14)
- Rect 클래스
 - 필드 : 가로, 세로
 - 생성자
 - 가로, 세로값 초기화
 - 메서드
 - findArea() – 사각형 면적 구한후 출력



예제

```
import java.util.Scanner;
class Shape {
    public void findArea() {
        System.out.println("도형의 면적을 구한다!");
    }
}
//
class Circle1 extends Shape {
    final double PI = 3.14;
    private int r;

    public Circle1(int r){
        this.r = r;
    }

    public void findArea() {
        System.out.println("원의 면적 : " + PI*r*r);
    }
}
//
class Rect1 extends Shape {
    private int w;
    private int h;

    public Rect1(int w, int h){
        this.w =w;
        this.h = h;
    }
}
```



예제

```
public void findArea() {
    System.out.println("사각형의 면적 : "+ w*h);
}
}
}

class PolymoArray2{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        final int MAX_COUNT=100;
        Shape[] mp = new Shape[MAX_COUNT];
        int cnt=0;

        while(true) {
            System.out.print("1.원 2.사각형 3.보기 4.종료 ==> ");
            int x = sc.nextInt();

            if (cnt==MAX_COUNT && (x==1 || x==2)){
                System.out.print("데이터를 더 이상 입력할 수 없습니다.WnWn");
                continue;
            }

            switch (x){
                case 1:
                    System.out.print("반지름은?");
                    int r = sc.nextInt();
                    mp[cnt++] = new Circle1(r);
                    break;
```



```

case 2:
    System.out.print("가로는?");
    int w = sc.nextInt();
    System.out.print("세로는?");
    int h = sc.nextInt();
    mp[cnt++] = new Rect1(w, h);
    break;

case 3:
    if (cnt==0){
        System.out.println("\n조회할 데이터가 없습니다!!\n");
        break;
    }

    System.out.println("\n----- 보기 -----");
    for (int k = 0; k < cnt; k++) {
        mp[k].findArea();
    }//for
    System.out.println("-----\n");
    break;

case 4:
    System.out.println("\n프로그램을 종료합니다.");
    return;

default:
    System.out.println("\n잘못 입력하셨습니다.");
    continue;

```

```

    }

```

```

} //while

```

```

} //main

```

```

} //class

```

```

class ShapeManager{
    private Scanner sc = new Scanner(System.in);
    final int MAX_COUNT=100;
    private Shape[] mp = new Shape[MAX_COUNT];
    private int cnt=0;

    public void inputCircle(){
        if (cnt==MAX_COUNT){
            System.out.println("데이터를 더 이상 입력할 수 없습니다.WnWn");
            return;
        }
        System.out.print("반지름은?");
        int r = sc.nextInt();
        mp[cnt++] = new Circle1(r);
    }
    public void inputRect(){
        if (cnt==MAX_COUNT){
            System.out.println("데이터를 더 이상 입력할 수 없습니다.WnWn");
            return;
        }
        System.out.print("가로는?");
        int w = sc.nextInt();
        System.out.print("세로는?");
        int h = sc.nextInt();
        mp[cnt++] = new Rect1(w, h);
    }
}

```


참조변수의 형변환

- 기본형 변수와 같이 참조형 변수도 형변환이 가능
 - 단, 서로 상속관계에 있는 클래스 사이에서만 가능
 - 자식 타입의 참조변수를 부모 타입의 참조변수로, 부모 타입의 참조변수를 자식 타입의 참조변수로의 형변환만 가능
- 참조형 변수의 형변환에서는 자식 타입의 참조변수를 부모 타입으로 형변환하는 경우에는 형변환 생략 가능(자동 형변환)
- 참조변수간의 형변환도 캐스트 연산자 사용 => (클래스명)

부모 타입 <- 자식 타입(Up-casting) : 자동 형변환

자식 타입 <- 부모 타입(Down-casting) : 명시적 형변환 (형변환 생략불가)

```
Parent p = (Parent)new Child(); //생략가능
```

```
Child c = (Child)p; //생략불가
```

```
Child c = (Child)new Parent(); //runtime 에러 (컴파일은 됨)
```



참조변수의 형변환

- 대입 연산자의 좌우변 양쪽 타입이 같지 않더라도 암시적으로 변환 가능하면 대입 가능

```
int i = 123;  
long k = i;
```

- 암시적, 명시적 변환 관계가 상속 계층의 클래스끼리도 허용됨
 - 자식 타입의 객체는 부모 타입으로 암시적으로 형변환됨
 - 부모 타입 변수에 자식 타입의 객체를 대입할 수 있음

```
Parent p = new Child();  
p.display();
```

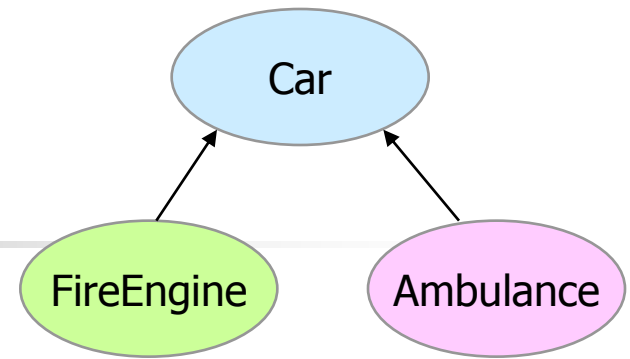
예제

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

    void drive() {                // 운전하는 기능
        System.out.println("drive, Brrrr~");
    }


    void stop() {                 // 멈추는 기능
        System.out.println("stop!!!");
    }
}

class FireEngine extends Car {   // 소방차
    void water() {               // 물을 뿌리는 기능
        System.out.println("water!!!");
    }
    void drive() {               // 운전하는 기능
        System.out.println("소방차를 운전합니다~");
    }
}
```



FireEngine타입의 참조변수와
Ambulance 타입의 참조변수
간에는 서로 형변환 불가능

```
class Ambulance extends Car {   // 앰불런스
    void siren() {              // 사이렌을 울리는 기능
        System.out.println("siren~~");
    }
    void drive() {              // 앰불런스를 운전합니다~
        System.out.println("앰불런스를 운전합니다~");
    }
}
```



```
Exception in thread "main" java.lang.ClassCastException: Car cannot be cast to FireEngine
    at CastingTest1.main(CastingTest1.java:12)
```

예제

```
class CastingTest1 {
    public static void main(String args[]) {
        Car c = new FireEngine(); //자동 형변환
        c.drive();
        //c.water(); //error

        FireEngine f = (FireEngine)c; //명시적 형변환
        f.water(); //자식만의 메서드도 호출 가능해짐

        //FireEngine f2 = (FireEngine)new Car(); //실행에러 (Class Cast Exception)
```

자식 타입으로의 형변환은 생략할 수 없으며, 형변환을 수행하기 전에 instanceof 연산자를 사용해서 참조변수가 참조하고 있는 실제 인스턴스의 타입을 확인하는 것이 안전함

- 형변환은 참조변수의 타입을 변환하는 것이지 인스턴스를 변환하는 것은 아니기 때문에 참조변수의 형변환은 인스턴스에 아무런 영향을 미치지 않음
- 단지, 참조변수의 형변환을 통해서, 참조하고 있는 인스턴스에서 사용할 수 있는 **멤버의 범위(개수)를 조절**하는 것뿐



예제

```
FireEngine fe = (FireEngine)new Car(); //실행에러
```

- 캐스트 연산자를 사용하면 서로 상속관계에 있는 클래스 타입의 참조변수간의 형변환은 양방향으로 자유롭게 수행될 수 있다.

그러나

- 참조변수가 참조하고 있는 인스턴스의 자식타입으로 형변환을 하는 것은 허용되지 않는다.

instanceof 연산자

참조변수 instanceof 타입(클래스명)

- 참조변수가 참조하고 있는 인스턴스의 실제 타입을 알아보기 위해 instanceof 연산자를 사용함
- 주로 조건문에 사용됨
- 연산의 결과로 boolean 값인 true, false 중의 하나를 반환
- instanceof를 이용한 연산결과로 true를 얻었다는 것은 참조변수가 검사한 타입으로 형변환이 가능하다는 것을 뜻함

```
Car c = new FireEngine();
```

```
if(c instanceof FireEngine) //c는 Car 타입의 참조변수
{
    FireEngine fe = (FireEngine)c;
    fe.water();
    //....
}
```

instanceof 연산자로 Car 타입의 참조변수 c가 FireEngine타입의 인스턴스를 참조하고 있는지를 검사



예제

```
소방자를 운전합니다~  
water!!!  
c2를 FireEngine으로 형변환 불가!!  
water!!!
```

//객체(참조변수) instanceof 클래스 => true, false => 객체의 타입을 확인

```
Car c2 = new Car();
```

```
if (c2 instanceof FireEngine){  
    FireEngine f3 = (FireEngine)c2;  
    f3.water();
```

```
}else{
```

```
    System.out.println("c2를 FireEngine으로 형변환 불가!!");
```

```
}
```

```
if (c instanceof FireEngine){
```

```
    FireEngine f4 = (FireEngine)c;  
    f4.water();
```

```
}else{
```

```
    System.out.println("c를 FireEngine으로 형변환 불가!!");
```

```
}//if
```

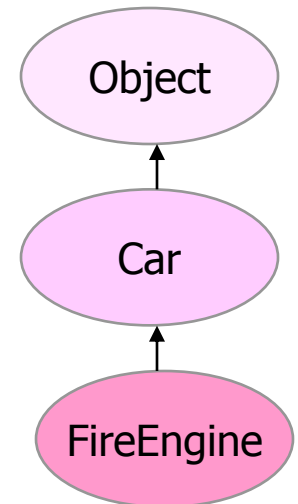
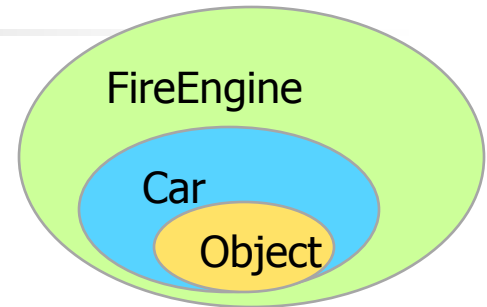
```
Car c = new FireEngine();
```

예제

- 자식객체 instanceof 부모클래스 : true
- 부모타입의 instanceof 연산에도 true
=> 자식은 부모의 인스턴스이기도 하므로

```
FireEngine f5 = new FireEngine();
if (f5 instanceof FireEngine)
{
    System.out.println("f5는 FireEngine 의 인스턴스!");
}
if (f5 instanceof Car) // 자식객체 instanceof 부모클래스 : true
{
    System.out.println("f5는 Car 의 인스턴스!");
}
if (f5 instanceof Object)
{
    System.out.println("f5는 Object 의 인스턴스!");
}
}
```

```
f5는 FireEngine 의 인스턴스!
f5는 Car 의 인스턴스!
f5는 Object 의 인스턴스!
```





예제

생성된 인스턴스는 **FireEngine** 타입일지라도, **Object** 타입과 **Car** 타입의 **instanceof** 연산에서도 **true**

=> 이유 : **FireEngine** 클래스는 **Object** 클래스와 **Car** 클래스의 자식 클래스이므로 부모의 멤버들을 상속받았기 때문에, **FireEngine** 인스턴스는 **Object** 인스턴스와 **Car** 인스턴스를 포함하고 있는 셈이므로

- 실제 인스턴스와 같은 타입의 instanceof 연산 이외에 **부모타입의 instanceof 연산에도 true**를 결과로 얻음
- instanceof 연산의 결과가 true라는 것은 검사한 타입으로 형변환이 가능하다는 것



참조변수와 인스턴스의 연결

- 멤버변수가 부모 클래스와 자식 클래스에 중복으로 정의된 경우
 - 부모 타입의 참조변수를 사용했을 때는 부모 클래스에 선언된 멤버변수가 사용되고, 자식 타입의 참조변수를 사용했을 때는 자식 클래스에 선언된 멤버변수가 사용됨
- 메서드
 - 부모 클래스의 메서드를 자식의 클래스에서 오버라이딩한 경우에도, 참조변수의 타입에 관계없이 항상 실제 인스턴스의 메서드(오버라이딩된 메서드)가 호출됨

예제 1

```
p.x = 100  
Child Method  
c.x = 200  
Child Method
```

```
class BindingTest{  
    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();  
    }  
}
```

```
class Parent {  
    int x = 100;  
  
    void method() {  
        System.out.println("Parent Method");  
    }  
}
```

```
class Child extends Parent {  
    int x = 200;  
  
    void method() {  
        System.out.println("Child Method"); //오버라이딩 메서드  
    }  
}
```

- 메서드 : 자식의 오버라이딩된 메서드가 호출됨
- 인스턴스변수 : 참조변수의 타입에 따라서 달라짐



예제2

```
p.x = 100  
Parent Method  
c.x = 100  
Parent Method
```

```
class BindingTest2 {  
    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();  
    }  
}  
  
class Parent {  
    int x = 100;  
  
    void method() {  
        System.out.println("Parent Method");  
    }  
}  
  
class Child extends Parent { }
```

자식 클래스에서 부모 클래스의 멤버를 중복으로 정의하지 않았을 때는 참조변수의 타입에 따른 변화는 없다.



정리

- 참조자료형도 상속관계에서는 형변환 가능
- 자식 타입의 객체는 부모 타입으로 자동 형 변환됨
- 부모 타입의 참조변수를 자식 타입으로 형변환할 때는 명시적 형변환
 - 단, 부모 타입의 실제 객체는 자식 타입이어야 함
- `instanceof` - 객체의 타입을 확인할 수 있음
- `instanceof` 로 타입확인을 할 때 부모 타입도 `true` 결과가 나옴
 - if문에서 자식 타입부터 확인해야 함



실습

```
춤을 춥니다.  
노래를 합니다.  
그림을 그립니다.
```

- 다음과 같은 실행결과를 얻도록 코드를 완성하십시오.
 - [Hint] instanceof 연산자를 사용해서 형변환한다.
 - 메서드명 : action
 - 기능 : 주어진 객체의 메서드를 호출한다.
 - DanceRobot인 경우, dance()를 호출하고,
 - SingRobot인 경우, sing()을 호출하고,
 - DrawRobot인 경우, draw()를 호출한다.

```

class Exam1 {
    (1) action() 메서드를 작성하시오.

    public static void main(String[] args) {
        Robot[] arr = { new DanceRobot(), new SingRobot(), new DrawRobot()};
        for(int i=0; i< arr.length;i++)
            action(arr[i]);
    } // main
}

```

```

class Robot {}
class DanceRobot extends Robot {
    void dance() {
        System.out.println("춤을 춥니다.");
    }
}
class SingRobot extends Robot {
    void sing() {
        System.out.println("노래를 합니다.");
    }
}
class DrawRobot extends Robot {
    void draw() {
        System.out.println("그림을 그립니다.");
    }
}

```



예제-매개변수의 다형성

```
class Person
{
    public String kind()
    {
        return "사람";
    }
};

class Student extends Person
{
    public String kind()
    {
        return "학생";
    }
};

class Graduate extends Student
{
    public String kind()
    {
        return "대학원생";
    }
};
```

```
class Assistant extends Person
{
    public String kind()
    {
        return "조교";
    }
};

class Professor extends Person
{
    public String kind()
    {
        return "교수";
    }
}

//
```



예제-매개변수의 다형성

```
class PolyArgs
{
    /*
        public static void register(Student s)
        {
            System.out.println(s.kind() + " 이/가 등록합니다.");
        }
        public static void register(Graduate g)
        {
            System.out.println(g.kind() + " 이/가 등록합니다.");
        }
        public static void register(Assistant a)
        {
            System.out.println(a.kind() + " 이/가 등록합니다.");
        }
    */
    */
}
```

예제-매개변수의 다형성

```
class PolyArgs
```

```
{
```

```
    public static void register(Person p)
```

```
    {
```

```
        if (p instanceof Student || p instanceof Assistant)
```

```
        {
```

```
            System.out.println(p.kind() + " 이/가 등록합니다." );
```

```
        }
```

```
    else
```

```
    {
```

```
        System.out.println(p.kind() + " 은/는 등록할 수 없습니다." );
```

```
    }
```

```
}
```

```
학생 이/가 등록합니다.  
대학원생 이/가 등록합니다.  
조교 이/가 등록합니다.  
교수 은/는 등록할 수 없습니다.  
true  
true  
true
```

- instanceof 연산자 (**A**참조변수 instanceof **B**타입)

- 참조변수가 우변의 타입과 맞는지를 조사
- 맞으면 **true**, 그렇지 않으면 **false**를 리턴
- **A**가 **B**의 자식 타입이면 이때도 **true** 리턴
- 타입에 따라 차별화된 처리를 할 수 있음



예제-매개변수의 다형성

```
public static void main(String[] args)
{
    Student kang = new Student();
    Graduate kim = new Graduate();
    Assistant lee = new Assistant();
    Professor park = new Professor();

    register(kang);
    register(kim);
    register(lee);
    register(park);

    //instanceof 연산자 : 참조변수 instanceof 자료형 (결과값=>true/false)
    System.out.println(kang instanceof Student);
    System.out.println(kim instanceof Student); //자식변수 is 부모자료형
    System.out.println(lee instanceof Assistant);
    //System.out.println(park instanceof Student);
}
} //
```



실습1

```
이름, 기본급여를 입력하세요  
홍길동  
3000000
```

```
=====  
이름: 홍길동  
기본급여: 3000000
```

- 급여관리 시스템
 - 직원의 근무형태 - 고용직
 - 직원의 이름과 급여정보를 저장하기 위한 클래스
 - class Permanent
 - 필드 : 이름, 기본급여
 - 생성자
 - getter/setter
 - 메서드 : 급여계산
 - 메인 메서드
 - 인자를 넘겨 객체생성
 - 이름, 기본 급여를 getter로 불러서 화면에 출력



실습1-상속, 다형성 이용

- 고용형태
 - 고용직, 판매직(기본급여+판매실적에 따른 인센티브), 임시직
 - 급여계산방식의 차이
 - 고용직(Permanent) – 연봉제(매달 기본급여가 정해져 있다)
 - 판매직(SalesPerson) – 연봉제(기본급여)+인센티브(판매수익*0.15)
 - 임시직(Temporary) – 일한시간 * 시간당 급여
- 고용인들이 공통적으로 지니고 있어야 하는 멤버들을 모아서 Employee 클래스로 추상화시킨다
- 고용직 클래스 (Permanent) – Employee클래스를 상속받음
- 임시직 클래스 정의 (Temporary) –Employee클래스를 상속받음
 - 필드 : 일한시간(time), 시간당 급여(pay)
 - 생성자, getter/setter
 - 메서드 : 급여계산 => time*pay
- 판매직 클래스 정의 (SalesPerson) – Permanent 클래스를 상속받음
 - 필드 : 판매수익(earnings)
 - 생성자, getter/setter
 - 메서드 : 급여계산 => salary+(earnings*RATE), RATE=0.15 : 상수선언



실습1

- class Employee
 - 필드 : 이름
 - 생성자, getter/setter
 - 메서드 : 급여계산 findPay(){ return 0; }
- class Permanent – 고용직
 - 필드 : 기본급여(salary)
 - 생성자, getter/setter
 - 메서드 : 급여계산 findPay() => salary
- Class Temporary – 임시직
 - 필드 : 일한시간(time), 시간당 급여(pay)
 - 생성자, getter/setter
 - 메서드 : 급여계산 => time*pay
- Class SalesPerson – 판매직 : **Permanent** 클래스를 상속받음
 - 필드 : 판매수익(earnings)
 - 생성자, getter/setter
 - 메서드 : 급여계산 => salary+(earnings*RATE)
 - RATE=0.15 : 상수선언

고용형태 - 고용직<P>, 임시직<T>, 판매직<S>를 입력하세요

p

이름, 기본급여를 입력하세요

박지성

3400000

=====

고용형태: 고용직

이름: 박지성

급여: 3400000

고용형태 - 고용직<P>, 임시직<T>, 판매직<S>를 입력하세요

t

이름, 일한시간, 시간당급여를 입력하세요

이영표

10

70000

=====

고용형태: 임시직

이름: 이영표

급여: 700000

고용형태 - 고용직<P>, 임시직<T>, 판매직<S>를 입력하세요

s

이름, 기본급여, 판매수익을 입력하세요

홍길동

2000000

500000

=====

고용형태: 판매직

이름: 홍길동

급여: 2075000



실습2 - 계좌추가

- 은행에서 새로운 계좌 신설
 - 신용계좌(FaithAccount) : Account 클래스를 상속받는다
 - 신용등급이 높고 앞으로 좋은 거래실적이 예상되는 고객만을 대상으로 만들어 주는 계좌
 - 입금시 바로 1%의 이자가 추가로 더해짐($\text{balance} + \text{balance} * 0.01$)
 - 생성자 : ($\text{balance} + \text{balance} * 0.01$)
 - 메서드 : 입금하다($\text{balance} += \text{money} + \text{money} * 0.01$) 오버라이드
 - 기부계좌(ContriAccount) : Account 클래스를 상속받는다
 - 입금금액의 1%에 해당하는 금액이 사회기부금으로 기여됨
 - ($\text{balance} - \text{balance} * 0.01$)
 - 기부계좌에는 기부된 금액의 총액에 대한 정보가 존재함
 - 필드 : 기부금 총액(contribution)
 - 생성자 : ($\text{balance} - \text{balance} * 0.01$), $\text{contribution} = \text{balance} * 0.01$
 - 메서드 : 입금하다 오버라이딩
 - $\text{super.deposit}(\text{money} - \text{money} * 0.01);$
 - $\text{contribution} += \text{money} * 0.01;$
 - 화면출력 오버라이딩 - $\text{System.out.println}(\text{"총 기부액:"} + \text{contribution});$ 추가
 - 현재 남아있는 잔액정보를 조회할 때 기부된 총액도 출력되어야 함



실습2 - 계좌추가

- Account 클래스 (부모)
 - 필드: 계좌번호, 잔액
 - 생성자
 - 메서드 : 입금하다, 출금하다, 화면출력



실습2 - Account 클래스

```
class Account
{
    protected String accNo;
    protected double balance;

    public Account(String accNo, double balance)
    {
        this.accNo = accNo;
        this.balance = balance;
    }

    public void deposit(double money) //입금
    {
        balance += money;
    }
    public void withdraw(double money) //출금
    {
        balance -= money;
    }
    public void display()
    {
        System.out.println("계좌번호:"+accNo);
        System.out.println("계좌잔액:"+balance);
    }
}
```

개설할 계좌종류-일반계좌<A>,신용계좌<F>,기부계좌<C>, 계좌번호, 잔액을 입력하세요

a
100-20-3541
540000

=====

계좌번호:100-20-3541
계좌잔액:540000
입금할 금액을 입력하세요
10000

계좌번호:100-20-3541
계좌잔액:550000

개설할 계좌종류-일반계좌<A>,신용계좌<F>,기부계좌<C>, 계좌번호, 잔액을

f
100-200-1205
800000

=====

계좌번호:100-200-1205
계좌잔액:808000
입금할 금액을 입력하세요
10000
계좌번호:100-200-1205
계좌잔액:818100

개설할 계좌종류-일반계좌<A>,신용계좌<F>,기부계좌<C>, 계좌번호, 잔

c
120-51-64785
900000

=====

계좌번호:120-51-64785
계좌잔액:891000
총기부액:9000
입금할 금액을 입력하세요
10000

계좌번호:120-51-64785
계좌잔액:900900
총기부액:9100



과제 - 전화번호 관리 프로그램 4단계

- PhoneInfo 클래스의 멤버변수인 birth 삭제
- PhoneInfo 를 상속받는 두 자식 클래스 추가
 - PhoneUnivInfo - 대학동기들의 전화번호 저장
 - PhoneCompanyInfo - 회사 동료들의 전화번호 저장
- PhoneUnivInfo
 - name (이름)
 - phoneNumber (전화번호)
 - major (전공)
 - year (학번 - 년도)
- PhoneCompanyInfo
 - name (이름)
 - phoneNumber (전화번호)
 - company (회사)
- 데이터 입력 메서드만 변경한다
 - 상속, 다형성을 이용해서 PhoneBookManager 클래스의 변경이 최소한으로 이뤄지도록 한다



전화번호 관리 프로그램 4단계

- 데이터를 입력받아서 객체를 생성하는 메서드를 3개 만든다.
 - 일반 : readFriendInfo() – PhoneInfo객체를 리턴
 - 대학 : readUnivFriendInfo() – PhoneUnivInfo객체를 리턴
 - 회사 : readCompanyFriendInfo() – PhoneCompanyInfo객체를 리턴
 - 이들의 반환타입은 모두 PhoneInfo 클래스
- PhoneInfo[] 배열에 PhoneInfo, PhoneUnivInfo, PhoneCompanyInfo 객체도 넣는다.
 - 부모배열에 자식 객체들을 저장
 - 위의 3개의 메서드에서 반환한 PhoneInfo객체(부모 객체)를 PhoneInfo[]배열에 넣는다

선택하세요...

1. 데이터 입력
2. 전체 데이터 조회
3. 데이터 검색
4. 데이터 삭제
5. 프로그램 종료

선택: 1

데이터 입력을 시작합니다..

1. 일반, 2. 대학, 3. 회사

선택>> 1

이름: 김기수

전화번호: 010-700-8000

데이터 입력이 완료되었습니다.

선택하세요...

1. 데이터 입력
2. 전체 데이터 조회
3. 데이터 검색
4. 데이터 삭제
5. 프로그램 종료

선택: 1

데이터 입력을 시작합니다..

1. 일반, 2. 대학, 3. 회사

선택>> 2

이름: 홍길동

전화번호: 010-100-2000

전공: 컴공

학번<연도>: 2011

데이터 입력이 완료되었습니다.

선택하세요...

1. 데이터 입력
2. 전체 데이터 조회
3. 데이터 검색
4. 데이터 삭제
5. 프로그램 종료

선택: 1

데이터 입력을 시작합니다..

1. 일반, 2. 대학, 3. 회사

선택>> 3

이름: 박길도

전화번호: 010-500-9000

회사: mbc

데이터 입력이 완료되었습니다.

선택하세요...

1. 데이터 입력
2. 전체 데이터 조회
3. 데이터 검색
4. 데이터 삭제
5. 프로그램 종료

선택: 2

-----전체 데이터 조회-----

name: 홍길동

phone: 010-100-2000

major: 컴공

year: 2011

name: 김기수

phone: 010-700-8000

name: 박길도

phone: 010-500-9000

company: mbc