

JAVA 프로그래밍

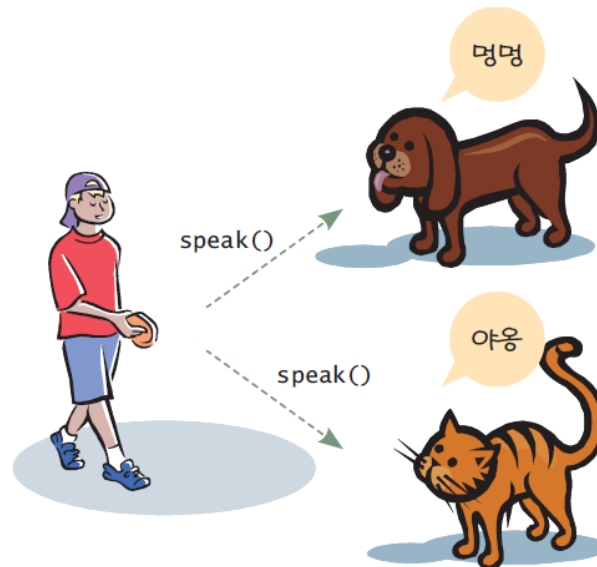
상속 & 다형성

다형성(polymorphism)

- 다형성(polymorphism)

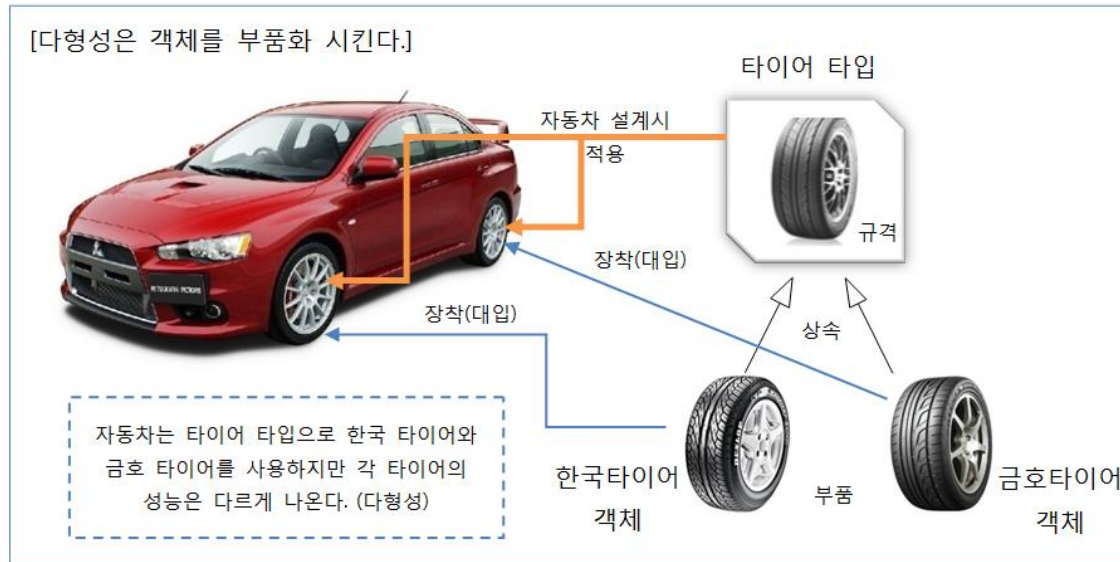
: 객체들의 타입이 다르면 똑같은 메시지가 전달되더라도 서로 다른 동작을 하는 것

- 예를 들어, 강아지의 `speak()` 메서드에서는 "멍멍"이라고 동작하고, 고양이의 `speak()` 메서드에서는 "야옹"이라고 동작하도록 구현하는 것



다형성(polymorphism)

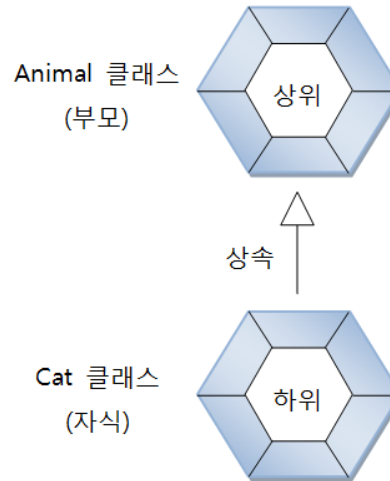
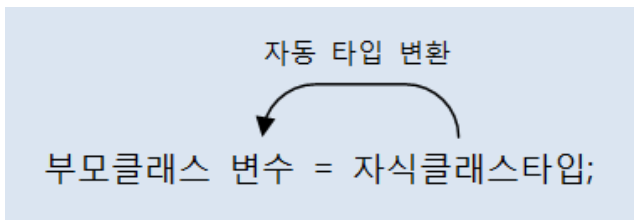
- 하나의 코드가 여러 가지 자료형으로 구현되어 실행되는 것
- 정보은닉, 상속과 더불어 객체지향 프로그래밍의 가장 큰 특징 중 하나
- 객체지향 프로그래밍의 유연성, 재활용성, 유지보수성에 기본이 되는 특징임



[다형성의 효과 : 객체 부품화 가능]

타입 변환과 다형성(polymorphism)

- 자동 타입 변환 : 프로그램 실행 도중에 자동 타입 변환이 일어나는 것



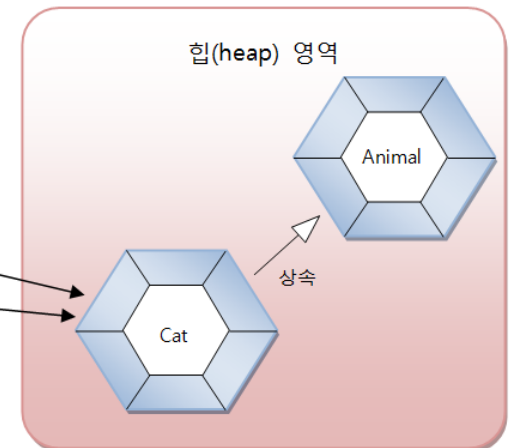
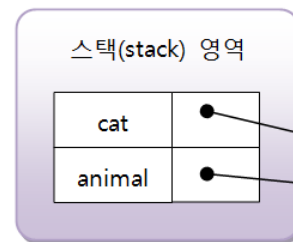
```
class Animal {  
    ...  
}
```

```
class Cat extends Animal {  
    ...  
}
```

```
Cat cat = new Cat();  
Animal animal = cat;
```

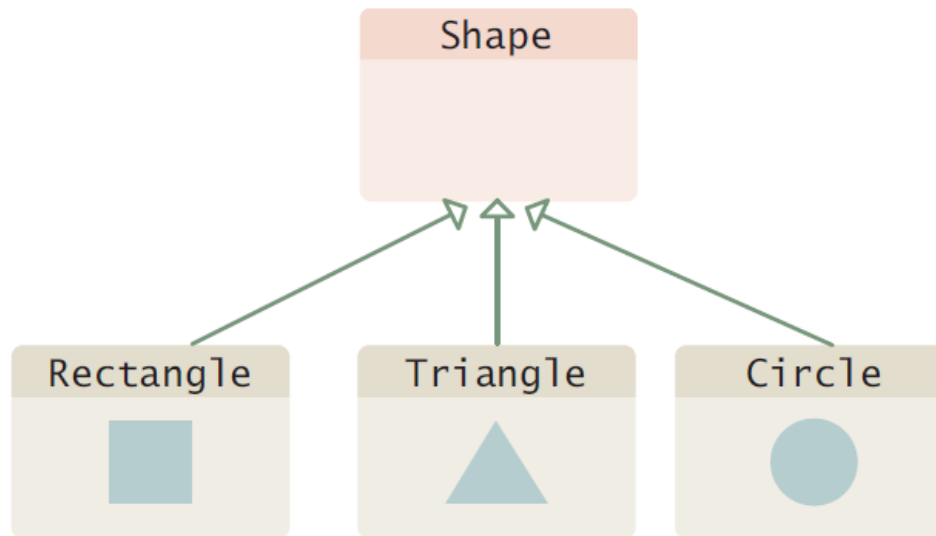
Animal animal = new Cat(); 도 가능하다.

```
cat == animal //true
```



상향 형 변환(업캐스팅)

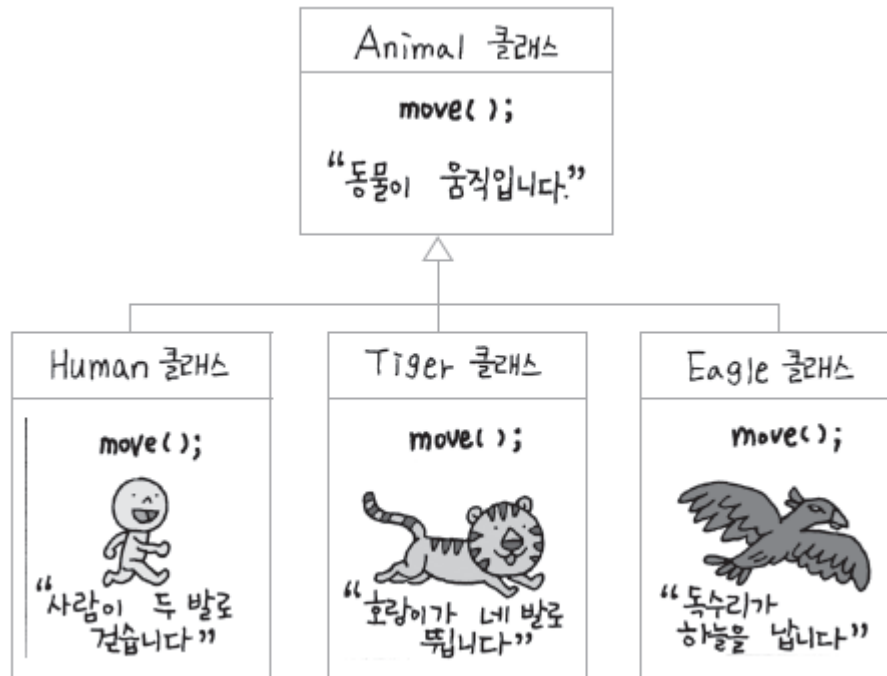
- 부모 타입의 참조변수로 자식 타입의 객체를 다룰 수 있도록 하는 것
- 서로 상속관계에 있는 타입 간의 형 변환만 가능
- 자식 타입에서 부모 타입으로 형 변환하는 경우에는 형 변환 생략 가능



```
Shape s;  
s = new Shape();           // ❶ 당연하다.  
s = new Rectangle();       // ❷ OK, Rectangle 객체도 가리킬 수 있다.
```

다형성 구현하기

- 하나의 클래스를 상속 받은 여러 클래스가 있는 경우
- 각 클래스마다 같은 이름의 서로 다른 메서드를 재정의함
- 상위 클래스 타입으로 선언된 하나의 변수가 여러 인스턴스에 대입되어 다양한 구현이 실행될 수 있음



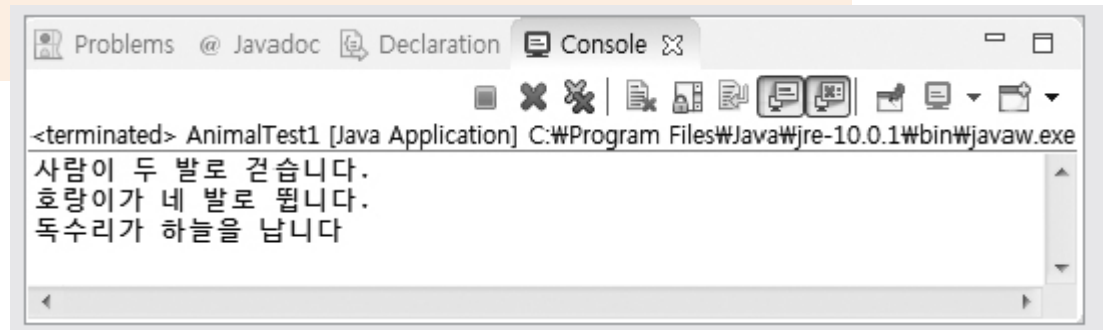
다형성 구현하기

```
public class AnimalTest1 {  
    public static void main(String[ ] args) {  
        AnimalTest aTest = new AnimalTest( );  
        aTest.moveAnimal(new Human( ));  
        aTest.moveAnimal(new Tiger( ));  
        aTest.moveAnimal(new Eagle( ));  
    }  
  
    public void moveAnimal(Animal animal) {  
        animal.move( );  
    }  
}
```

매개변수의 자료형이 상위 클래스

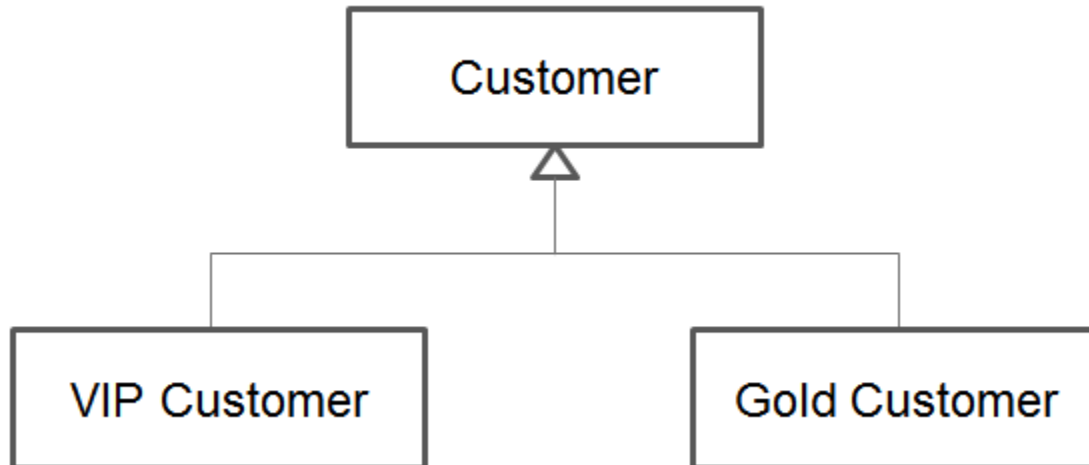
재정의된 메서드가 호출됨

동일한 animal.move()
코드에 대해 각기 다른
구현이 실행



다형성 활용하기

- 일반 고객과 VIP 고객의 중간 등급(Gold)의 고객을 생성
- 5명의 고객을 ArrayList에 생성하여 저장한 다음
- 각 고객이 물건을 샀을 때 가격과 보너스 포인트를 계산
 - 제품 구매 시 항상 10% 할인
 - 보너스 포인트 2% 적립
 - 담당 전문 상담원은 없음



상속을 언제 사용할까?

- 여러 클래스를 생성하지 않고 하나의 클래스에 공통적인 요소를 모으고 나머지 클래스는 이를 상속받은 다음 각각 필요한 특성과 메서드를 구현하는 방법
- 하나의 클래스에 여러 특성을 한꺼번에 구현하는 경우, 많은 코드 내에 많은 if문이 생길 수 있음

```
if(customerGrade == "VIP") { //할인해 주고, 적립도 많이 해주고
}
else if(customerGrade == "GOLD") { //할인해 주고, 적립은 적당히
}
else if(customerGrade == "SILVER") { //적립만 해준다
}
```

상속은 언제 사용할까?

- IS-A 관계(is a relationship : inheritance)
 - 일반적인(general) 개념과 구체적인(specific) 개념과의 관계
 - 상위 클래스 : 일반적인 개념 클래스 (예: 포유류)
 - 하위 클래스 : 구체적인 개념 클래스(예: 사람, 원숭이, 고래...)
 - 단순히 코드를 재사용하는 목적으로 사용하지 않음
- HAS-A 관계(composition) : 한 클래스가 다른 클래스를 소유한 관계

코드 재사용의 한 방법
Student 가 Subject를
포함한 관계

```
class Student {  
    Subject majorSubject;  
}
```

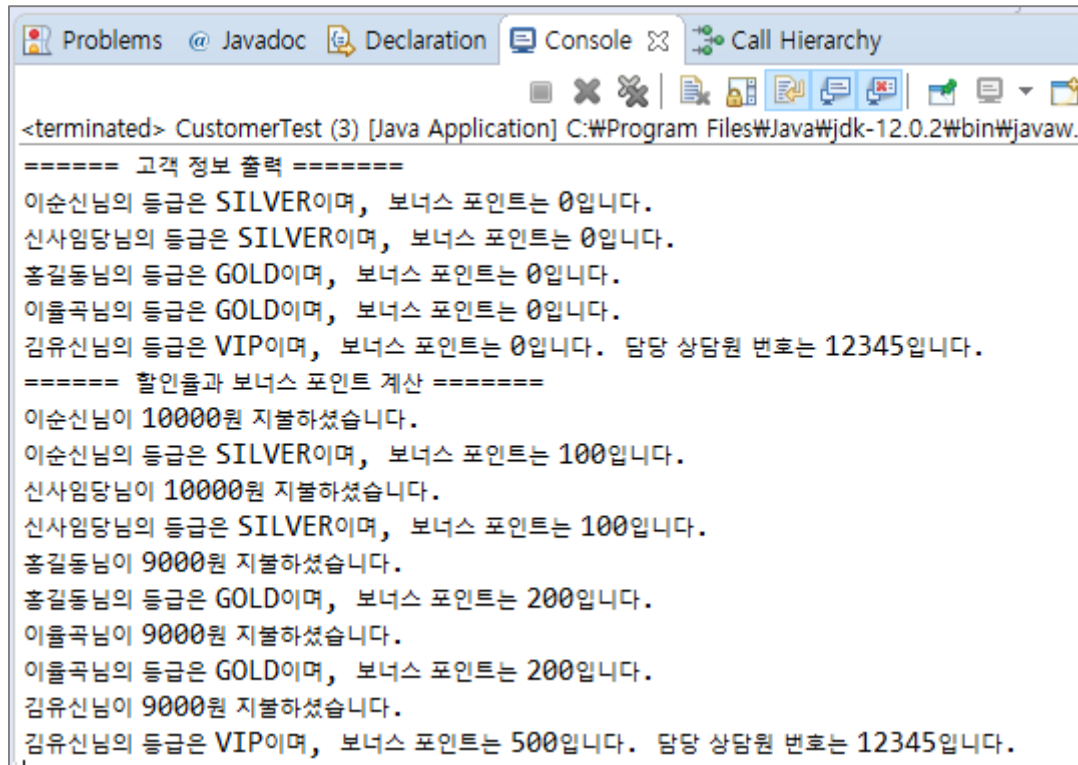
❖ 상속을 사용하면 클래스 간의 결합도가 높아져 상위 클래스의 변화가 하위 클래스에 미치는 영향이 큼

새로운 고객 등급 추가하기

```
public class GoldCustomer extends Customer {  
    private double saleRatio;    // 할인을  
  
    public GoldCustomer(int customerID, String customerName) {  
        super(customerID, customerName);  
        customerGrade = "GOLD";  
        bonusRatio = 0.02;  
        saleRatio = 0.1;  
    }  
  
    public int calcPrice(int price) {    // overriding method  
        bonusPoint += price * bonusRatio;  
        return price - (int) (price * saleRatio);  
    }  
}
```

배열을 활용한 고객 관리 프로그램 구현

[출력 결과]



```
<terminated> CustomerTest (3) [Java Application] C:\Program Files\Java\jdk-12.0.2\bin\javaw.  
===== 고객 정보 출력 =====  
이순신님의 등급은 SILVER이며, 보너스 포인트는 0입니다.  
신사임당님의 등급은 SILVER이며, 보너스 포인트는 0입니다.  
홍길동님의 등급은 GOLD이며, 보너스 포인트는 0입니다.  
이율곡님의 등급은 GOLD이며, 보너스 포인트는 0입니다.  
김유신님의 등급은 VIP이며, 보너스 포인트는 0입니다. 담당 상담원 번호는 12345입니다.  
===== 할인율과 보너스 포인트 계산 =====  
이순신님이 10000원 지불하셨습니다.  
이순신님의 등급은 SILVER이며, 보너스 포인트는 100입니다.  
신사임당님이 10000원 지불하셨습니다.  
신사임당님의 등급은 SILVER이며, 보너스 포인트는 100입니다.  
홍길동님이 9000원 지불하셨습니다.  
홍길동님의 등급은 GOLD이며, 보너스 포인트는 200입니다.  
이율곡님이 9000원 지불하셨습니다.  
이율곡님의 등급은 GOLD이며, 보너스 포인트는 200입니다.  
김유신님이 9000원 지불하셨습니다.  
김유신님의 등급은 VIP이며, 보너스 포인트는 500입니다. 담당 상담원 번호는 12345입니다.
```

배열을 활용한 고객 관리 프로그램 구현

```
import java.util.ArrayList;
```

```
public class CustomerTest {  
    public static void main(String[] args) {  
        ArrayList<Customer> customerList = new ArrayList<Customer>();  
  
        Customer customerLee = new Customer(10010, "이순신");  
        Customer customerShin = new Customer(10020, "신사임당");  
        GoldCustomer customerHong = new GoldCustomer(10030, "홍길동");  
        GoldCustomer customerYul = new GoldCustomer(10040, "이율곡");  
        VIPCustomer customerKim = new VIPCustomer(10050, "김유신", 12345);
```

```
        customerList.add(customerLee);  
        customerList.add(customerShin);  
        customerList.add(customerHong);  
        customerList.add(customerYul);  
        customerList.add(customerKim);
```

← ArrayList 객체 배열에 고객 추가

```
        System.out.println("===== 고객 정보 출력 =====");
```

```
        for (Customer customer : customerList) {  
            System.out.println(customer.showCustomerInfo());  
        }
```

```
        System.out.println("===== 할인율과 보너스 포인트 계산 =====");
```

```
        int price = 10000;
```

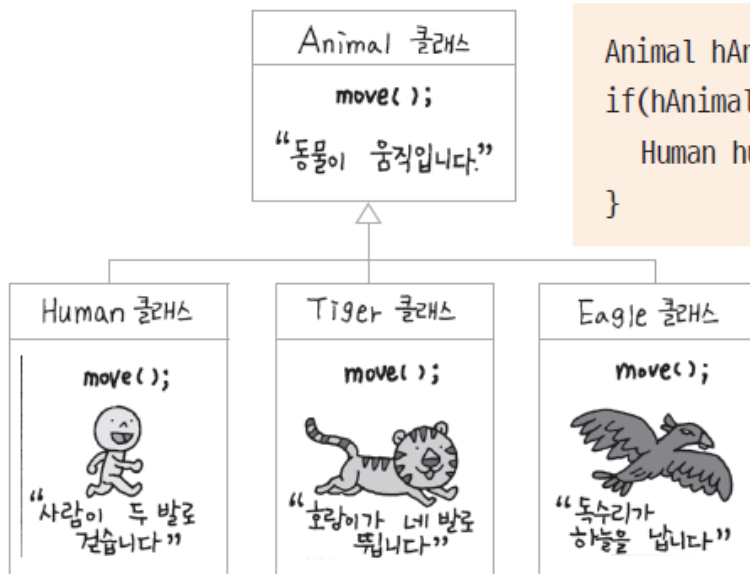
```
        for (Customer customer : customerList) {  
            int cost = customer.calcPrice(price);  
            System.out.println(customer.getCustomerName() + "님이 " + cost + "원 지불하셨습니다.");  
            System.out.println(customer.showCustomerInfo());  
        }
```

← 다형성 구현

```
    }  
}
```

다운 캐스팅 - instanceof

- 하위 클래스가 상위 클래스로 형 변환 되는 것은 묵시적으로 이루어짐
- 다시 원래 자료형인 하위 클래스로 형 변환 하려면, 명시적으로 다운 캐스팅을 해야 함
- 이때 원래 인스턴스의 타입을 체크하는 예약어가 **instanceof** 임



```
Animal hAnimal = new Human( );
if(hAnimal instanceof Human) {    // hAnimal 인스턴스 자료형이 Human형이라면
    Human hunam = (Human)hAnimal;  // 인스턴스 hAnimal을 Human형으로 다운 캐스팅
}
```

- ✓ 참조 변수의 원래 인스턴스 형을 정확히 확인하고
다운 캐스팅을 해야 안전한 코드

원래 인스턴스 형 확인 후 다운 캐스팅하기

```
class Animal {  
    public void move() {  
        System.out.println("동물이 움직입니다.");  
    }  
}  
  
class Human extends Animal {  
    @Override  
    public void move() {  
        System.out.println("사람이 두 발로 걷습니다.");  
    }  
  
    public void readBook() {  
        System.out.println("사람이 책을 읽습니다. ");  
    }  
}
```

```
class Tiger extends Animal {  
    @Override  
    public void move() {  
        System.out.println("호랑이가 네 발로 뜀니다.");  
    }  
  
    public void hunting() {  
        System.out.println("호랑이가 사냥을 합니다. ");  
    }  
}  
  
class Eagle extends Animal {  
    @Override  
    public void move() {  
        System.out.println("독수리가 하늘을 날니다 ");  
    }  
  
    public void flying() {  
        System.out.print("독수리가 날개를 쭉 펴고 멀리 날아갑니다.");  
    }  
}
```

원래 인스턴스 형 확인 후 다운 캐스팅하기

```
import java.util.ArrayList;

public class AnimalTest {
    // 배열의 자료형은 Animal 타입
    ArrayList<Animal> aniList = new ArrayList<Animal>();

    public static void main(String[] args) {
        AnimalTest aTest = new AnimalTest();
        aTest.addAnimal();
        System.out.println("원래 타입으로 다운 캐스팅 ");
        aTest.testCasting();
    }

    public void addAnimal() {
        // ArrayList에 추가되면서 Animal타입으로 형 변환
        aniList.add(new Human());
        aniList.add(new Tiger());
        aniList.add(new Eagle());

        for(Animal ani : aniList) { // 배열의 요소들을 Animal형으로 꺼내서
            ani.move(); // move()를 호출하면 오버라이딩 함수가 호출됨
        }
    }
}
```


원래 인스턴스 형 확인 후 다운 캐스팅하기

```
public void testCasting() {  
    for(int i = 0; i < aniList.size(); i++) { // 모든 배열 항목들을 하나씩 돌면서  
        Animal ani = aniList.get(i); // Animal 타입으로 i번째 요소를 얻어옴  
        if (ani instanceof Human) {  
            Human h = (Human)ani;  
            h.readBook();  
        } else if (ani instanceof Tiger) {  
            Tiger t = (Tiger)ani;  
            t.hunting();  
        } else if (ani instanceof Eagle) {  
            Eagle e = (Eagle)ani;  
            e.flying();  
        } else {  
            System.out.println("지원되지 않는 타입입니다.");  
        }  
    }  
}
```

Quiz 다형성

<ShapeTest.java>

- 상위 클래스 : Shape, draw()메서드
- 하위 클래스 : Circle, Triangle, draw()메서드 override
- ArrayList에 세 종류의 객체를 추가하고 출력 결과와 같이 구현
(단, 출력은 Enhanced for 사용)

[출력 결과]

Circle

Triangle

Shape