8. 상속과 다형성

상속이란?

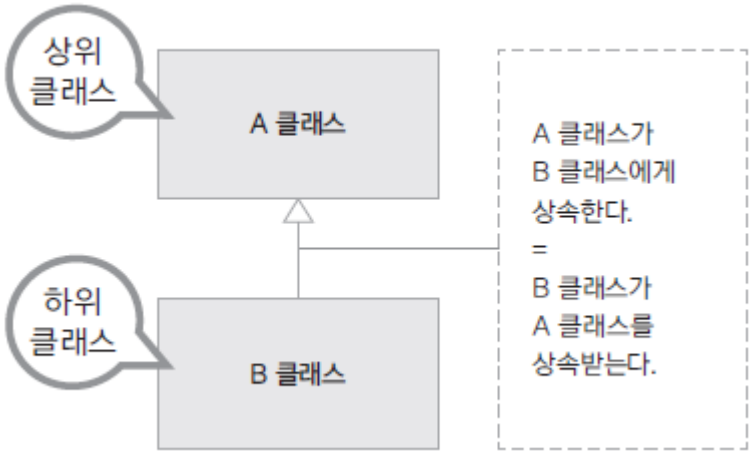
클래스를 정의 할 때 이미 구현된 클래스를 상속(inheritance) 받아서 속성이나 기능이 확장되는 클래스를 구현함

상속하는 클래스 : 상위 클래스, **parent class, base class, super class**

상속 받는 클래스 : 하위 클래스, **child class, derived class, subclass**

클래스 상속 문법

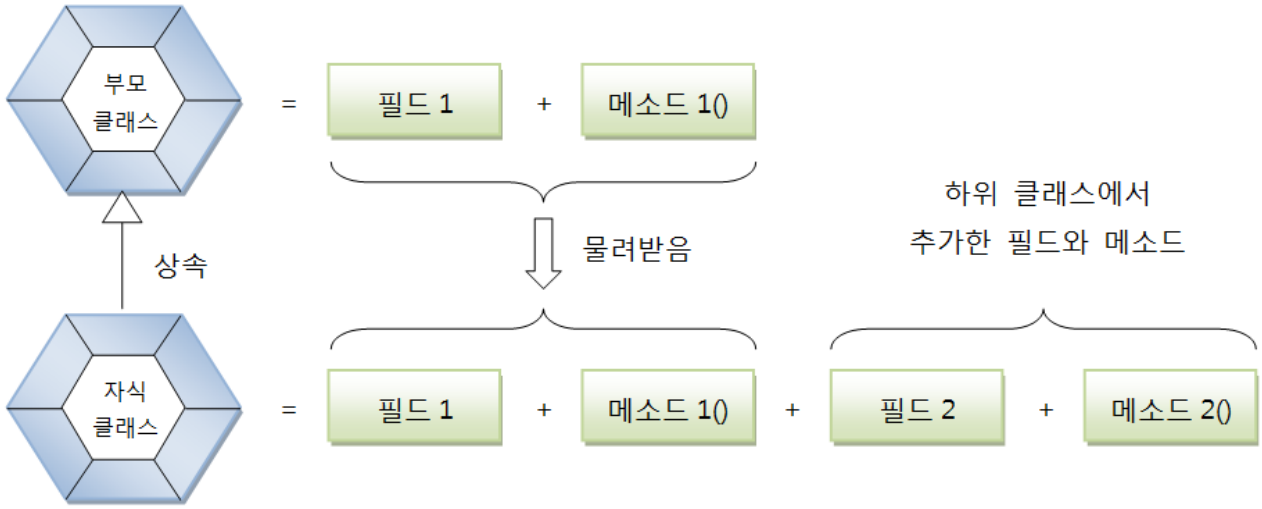
```
class B extends A {

}
```



상속이란?

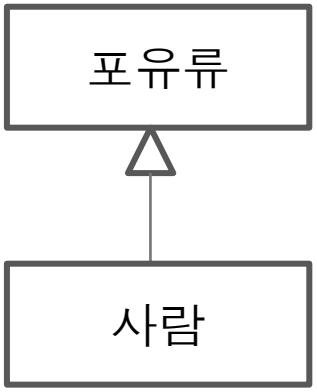
- 현실 세계:
  - 부모가 자식에게 물려주는 행위
  - 부모가 자식을 선택해서 물려줌

- 객체 지향 프로그램:
  - 자식(하위, 파생) 클래스가 부모(상위) 클래스의 멤버를 물려받는 것
  - 자식이 부모를 선택해 물려받음
  - 상속 대상: 부모의 필드와 메소드



상속이란?

상위 클래스는 하위 클래스 보다 일반적인 의미를 가짐
하위 클래스는 상위 클래스 보다 구체적인 의미를 가짐




```
class Mammal{

}

class Human extends Mammal{

}
```

extends 뒤에는 단 하나의 class 만 사용할 수 있음
자바는 single inheritance만을 지원 함

상속개념의 활용

- 상속의 효과
  - 부모 클래스 재사용해 자식 클래스 빨리 개발 가능
  - 반복된 코드 중복 줄임
  - 유지 보수 편리성 제공
  - 객체 다형성 구현 가능

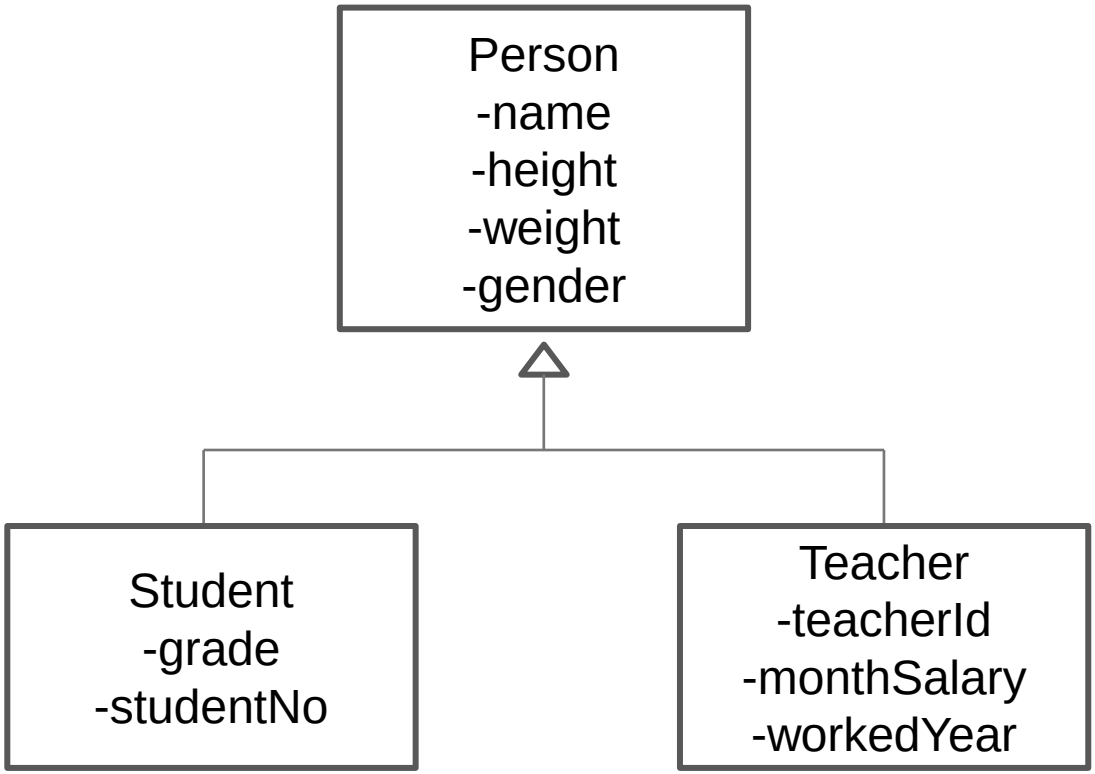
- 상속 대상 제한
  - 부모 클래스의 private 접근 갖는 필드와 메소드 제외
  - 부모 클래스가 다른 패키지에 있을 경우, default 접근 갖는 필드와 메소드도 제외

상속을 활용한 PERSON CLASS 구현

부모 클래스로 **Person class** 구현

자식 클래스로 **Student class** 와 **Teacher class** 구현

구현한 클래스는 **Scanner class**로 입력 받아서 배열을 이용하여 저장

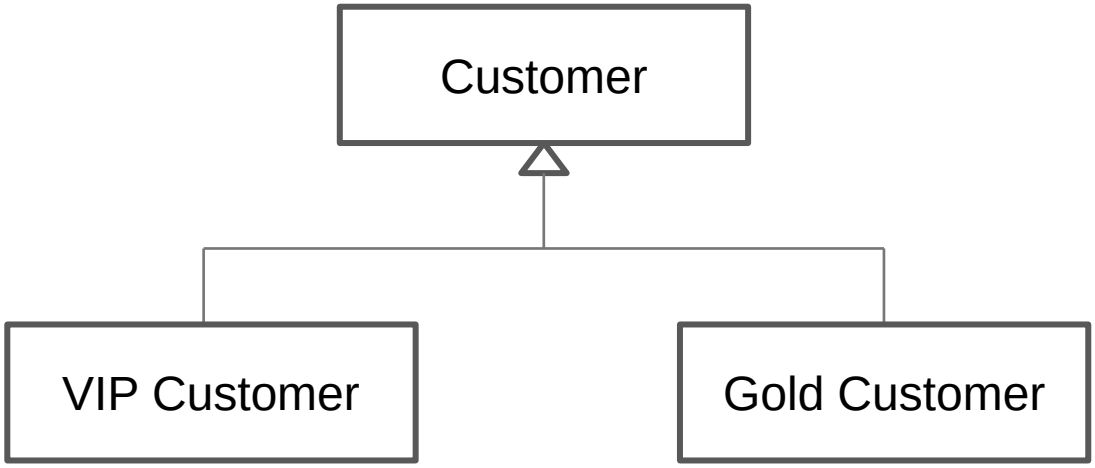


상속을 활용한 고객관리 프로그램

고객의 정보를 활용하여 고객 맞춤 서비스를 구현

고객의 등급에 따라 차별화 된 할일율과 포인트를 지급

등급에 따른 클래스를 따로 구현하는 것이 아닌 일반적인 클래스를 먼저 구현하고 그 보다 기능이 많은 클래스는 상속을 활용하여 구현함



상속을 활용한 고객관리 프로그램

Customer 클래스 속성

멤버 변수	설명
customerID	고객 아이디
customerName	고객 이름
customerGrade	고객 등급 기본 생성자에서 지정되는 기본 등급은 SILVER입니다.
bonusPoint	고객의 보너스 포인트 - 고객이 제품을 구매할 경우 누적되는 보너스 포인트입니다.
bonusRatio	보너스 포인트 적립 비율 - 고객이 제품을 구매할 때 구매 금액의 일정 비율이 보너스 포인트로 적립됩니다. 이때 계산되는 적립 비율입니다. - 기본 생성자에서 지정되는 적립 비율은 1%입니다. 즉 10,000원짜리를 사면 100원이 적립됩니다.

상속을 활용한 고객관리 프로그램

Customer 클래스 메서드

메서드	설명
Customer()	기본 생성자 고객 한 명이 새로 생성되면 CustomerGrade 는 SILVER 이고, bonusRatio 는 1% 로 지정 함
calcPrice(int price)	제품에 대해 지불해야 하는 그맥을 계산하여 반환. 할인되지 않는 경우 가격을 그대로 반환. 가격에 대해 보너스 포인트 비율을 적용하여 보너스 포인트를 적립
showCustomerInfo()	고객 정보를 출력, 고객이름, 등급, 현재 적립된 포인트 정보를 보여 줌

Customer 클래스 구현

```
public class Customer {

    private int customerID;
    private String customerName;
    private String customerGrade;
    int bonusPoint;
    double bonusRatio;

    public Customer()
    {
        customerGrade = "SILVER";
        bonusRatio = 0.01;
    //  System.out.println("Cusomer() 생성자 호출");
    }

    public Customer(int customerID, String customerName){
        this.customerID = customerID;
        this.customerName = customerName;
        customerGrade = "SILVER";
        bonusRatio = 0.01;
    //  System.out.println("Cusomer(int, String) 생성자 호출");
    }

    public int calcPrice(int price){
        bonusPoint += price * bonusRatio;
        return price;
    }

    public String showCustomerInfo(){
        return customerName + " 님의 등급은 " + customerGrade + "이며, 보너스 포인트는 " + bonusPoint + "입니다.";
    }
}
```

새로운 고객 등급이 필요한 경우

단골고객에 대한 혜택이 필요 함

우수 고객을 관리 하기 위해 다음과 같은 혜택을 줌

고객 등급 : **VIP**

제품 구매 할인율 **:10%**

보너스 포린트 **: 5%**

담당 전문 상담원 배정

VIP 클래스 정의 하기

이미 구현되어 있는 **Customer** 클래스를 상속 받고 **Customer** 클래스에 구현되지 않은 속성과 메서드를 추가적으로 구현 함

```
public class VIPCustomer extends Customer{

    private int agentID;
    double saleRatio;

    public VIPCustomer()
    {
        customerGrade = "VIP";
        bonusRatio = 0.05;
        saleRatio = 0.1;
    }

    public int calcPrice(int price){
        bonusPoint += price * bonusRatio;
        return price - (int)(price * saleRatio);
    }

    public int getAgentID(){
        return agentID;
    }
}
```

protected 예약어

상위 클래스(Customer) 에 private 으로 선언된 변수나 메서드는 하위 클래스(VIPCustomer) 에서 사용할 수 없음

상위 클래스에 변수나 메서드를 선언할 때 하위 클래스에서도 가시성이 허용되도록 protected 예약어를 사용하여 선언하면 하위 클래스를 제외한 외부 클래스에 대해서는 private 과 동일한 기능

단 동일 패키지내에서는 가시성이 허용 됨

접근 제한자 가시성

	외부 클래스	하위 클래스	동일 패키지	내부 클래스
public	O	O	O	O
protected	X	O	O	O
선언되지 않음 (default)	X	X	O	O
private	X	X	X	O

수정한 Customer 클래스

따라서 **Customer** 클래스에 선언된 **private** 변수를 **protected**로 선언함

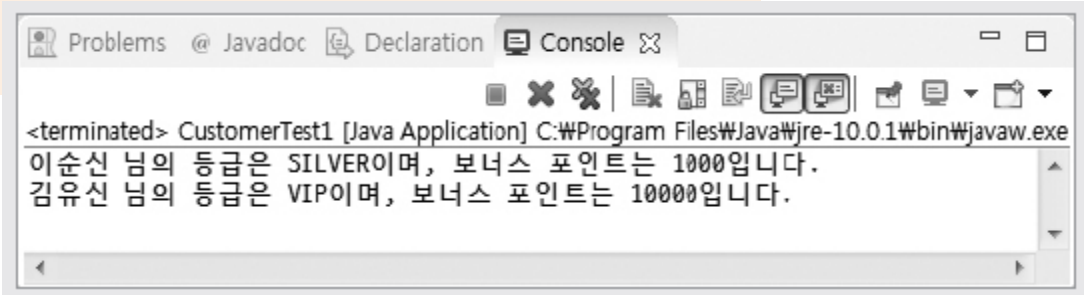
```
public class Customer {
  protected int customerID;
  protected String customerName;
  protected String customerGrade;
  int bonusPoint;
  double bonusRatio;
  ...
```

상속 클래스 테스트 하기

```
public class CustomerTest {
   public static void main(String[ ] args) {
      Customer customerLee = new Customer( );
      customerLee.setCustomerID(10010);
      customerLee.setCustomerName("이순신");
      customerLee.bonusPoint = 1000;
      System.out.println(customerLee.showCustomerInfo( ));

      VIPCustomer customerKim = new VIPCustomer( );
      customerKim.setCustomerID(10020);
      customerKim.setCustomerName("김유신");
      customerKim.bonusPoint = 10000;
      System.out.println(customerKim.showCustomerInfo( ));
   }
}
```

customerID와 customerName은 protected 변수이므로 set( ) 메서드 호출

customerID와 customerName은 protected변수이므로 set( ) 메서드 호출



상속에서 클래스 생성 과정

하위 클래스가 생성 될 때 상위 클래스가 먼저 생성 됨

상위 클래스의 생성자가 호출되고 하위 클래스의 생성자가 호출 됨

하위 클래스의 생성자에서는 무조건 상위 클래스의 생성자가 호출 되어야 함

아무것도 없는 경우 컴파일러는 상위 클래스 기본 생성자를 호출하기 위한 **super()** 를 코드에 넣어 줌

super() 호출되는 생성자는 상위 클래스의 기본 생성자 임

만약 상위 클래스의 기본생성자가 없는 경우 (매개변수가 있는 생성자만 존재 하는 경우) 하위 클래스는 명시적으로 상위 클래스르 호출해야 함

상속에서 클래스 생성 과정

상위 클래스 기본 생성자가 호출 되는 경우

```
public VIPCustomer( ) {
   super( );
   customerGrade = "VIP";
   bonusRatio = 0.05;
   saleRatio = 0.1;
   System.out.println("VIPCusomer( ) 생성자 호출");
}
```

컴파일러가 자동으로 추가하는 코드.
상위 클래스의 Customer( )가 호출됨

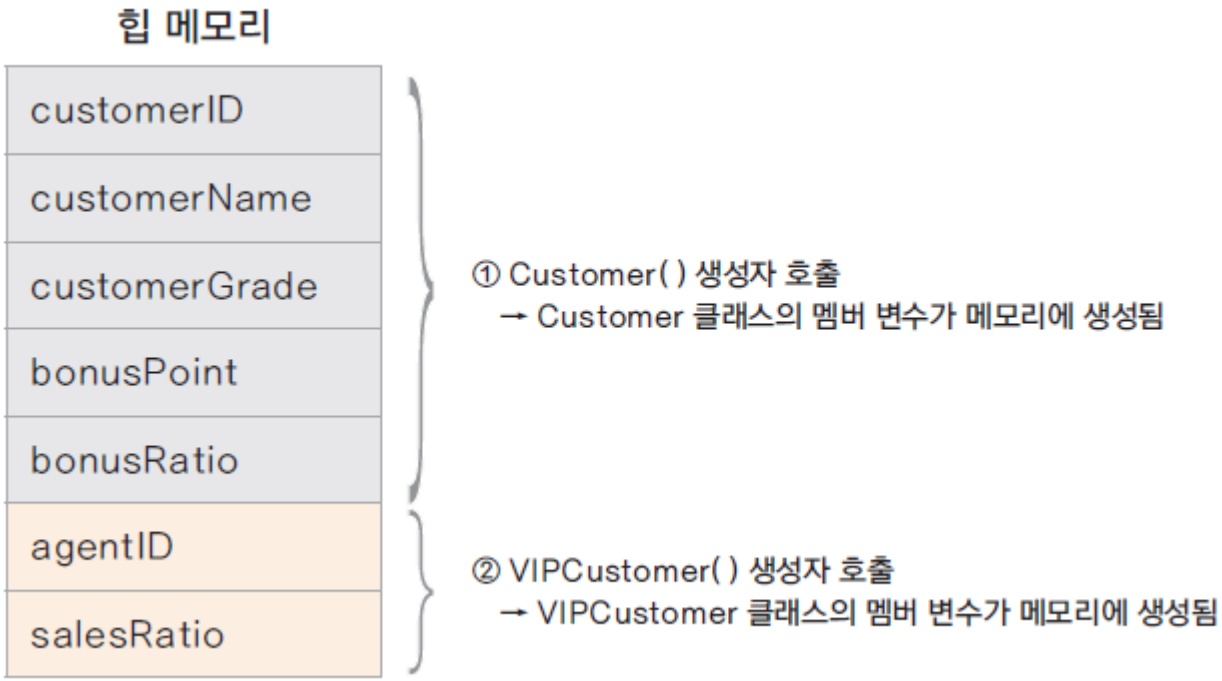
명시적으로 상위 클래스 생성자를 호출하는 경우

```
public VIPCustomer(int customerID, String customerName, int agentID) {
   super(customerID, customerName);
   customerGrade = "VIP";
   bonusRatio = 0.05;
   saleRatio = 0.1;
   this.agentID = agentID;
   System.out.println("VIPCusomer(int
}
```

```
public Customer(int customerID, String customerName) {
   this.customerID = customerID;
   this.customerName = customerName;
   customerGrade = "SILVER";
   bonusRatio = 0.01;
   System.out.println("Customer(int, String) 생성자 호출");
}
```

상속에서의 메모리 상태

상위 클래스의 인스턴스가 먼저 생성이 되고, 하위 클래스의 인스턴스가 생성 됨

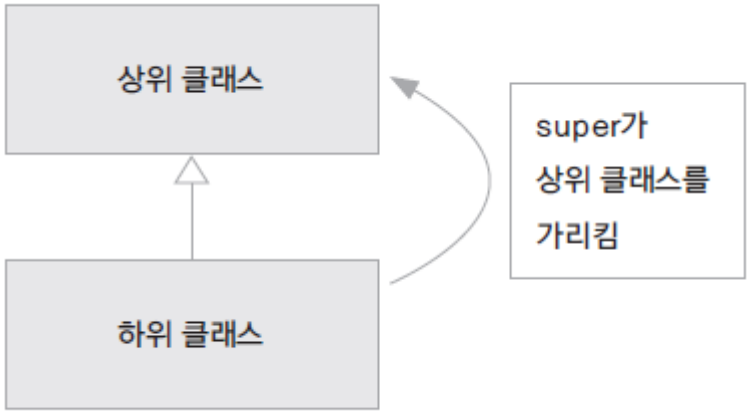


super 예약어

this 가 자기 자신의 인스턴스의 주소를 가지는 것처럼

super 는 하위 클래스가 상위 클래스에 대한 주소를 가지게 됨

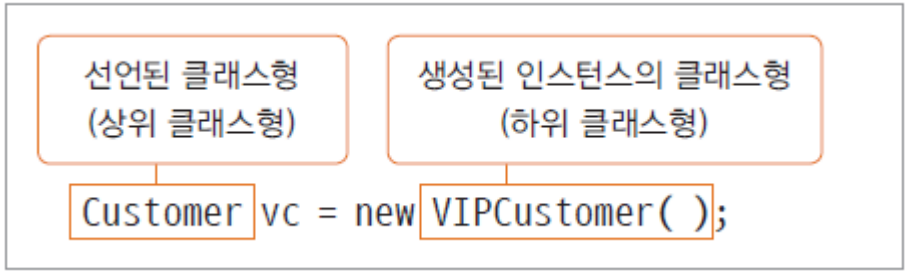
하위 클래스가 상위 클래스에 접근 할 때 사용할 수 있음



상위 클래스로의 묵시적 형 변환 (업캐스팅)

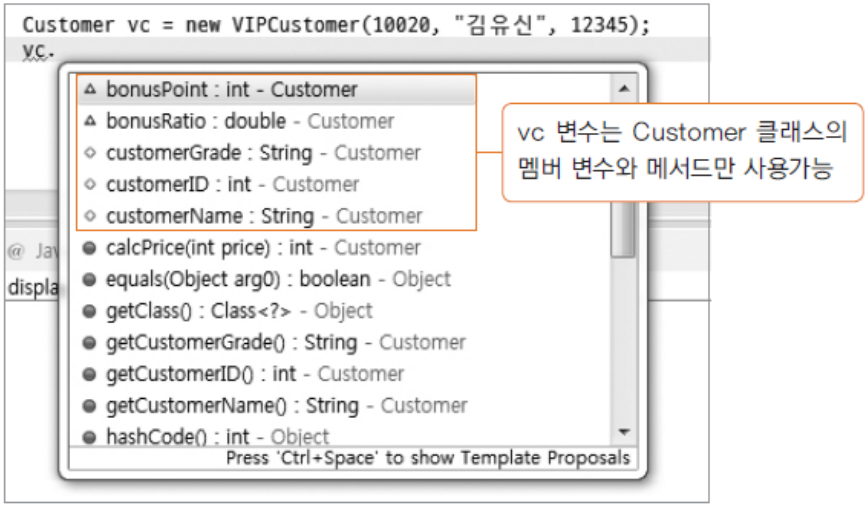
상위 클래스 형으로 변수를 선언하고 하위 클래스 인스턴스를 생성할 수 있음

하위 클래스는 상위 클래스의 타입을 내포하고 있으므로 상위 클래스로 묵시적 형변환이 가능 함



형 변환에서의 메모리

Customer vc = new VIPCustomer(); 에서 **vc** 가 가리키는 것은?



VIPCustomer() 생성자의 호출로 인스턴스는 모두 생성 되었지만

타입이 **Customer** 이므로 접근 할 수 있는 변수나 메서드는 **Customer**의 변수와 메서드 임

메서드 오버라이딩(overriding)

상위 클래스에 정의 된 메서드 중 하위 클래스와 기능이 맞지 않거나 추가 기능이 필요한 경우 같은 이름과 매개변수로 하위 클래스에서 재정의 함

VIPCustomer 클래스의 calcPrice() 메서드 재정의

```
public int calcPrice(int price){
    bonusPoint += price * bonusRatio;
    return price - (int)(price * saleRatio);
}
```

보너스 포인트 적립하고 할일률을 적용한 가격을 반환

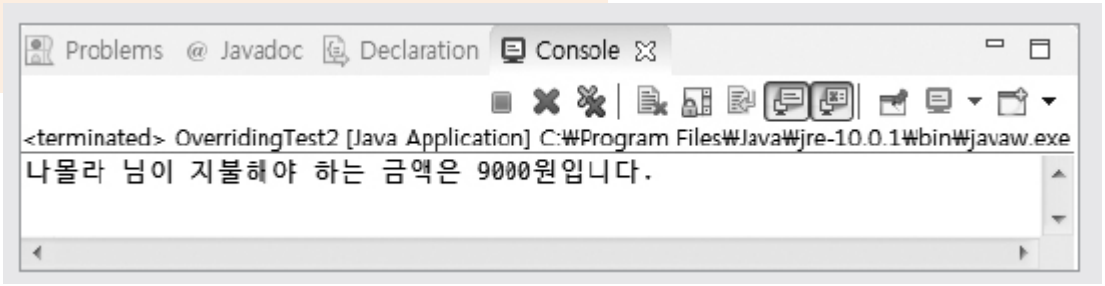
묵시적 형 변환과 재정의 된 메서드 호출

Customer vc = new VIPCustomer();

vc.calcPrice(10000);

위 코드에서 **calcPrice()** 메서드는 어느 메서드가 호출 될 것인가?

```
public class OverridingTest2 {
   public static void main(String[ ] args) {
      Customer vc = new VIPCustomer(10030, "나몰라", 2000);  //VIP 고객 생성
      vc.bonusPoint = 1000;

      System.out.println(vc.getCustomerName( ) + " 님이 지불해야 하는 금액은 "
                + vc.calcPrice(10000) + "원입니다.");
   }
}
```



가상 메서드(virtual method)

프로그램에서 어떤 객체의 변수나 메서드의 참조는 그 타입에 따라 이루어 짐. 가상 메서드의 경우는 타입과 상관없이 실제 생성된 인스턴스의 메서드가 호출 되는 원리

Customer vc = new VIPCustomer();

vc.calcPrice(10000);

vc의 타입은 **Customer** 지만, 실제 생성된 인스턴스인 **VIPCustomer** 클래스의 **calcPrice()** 메서드가 호출 됨

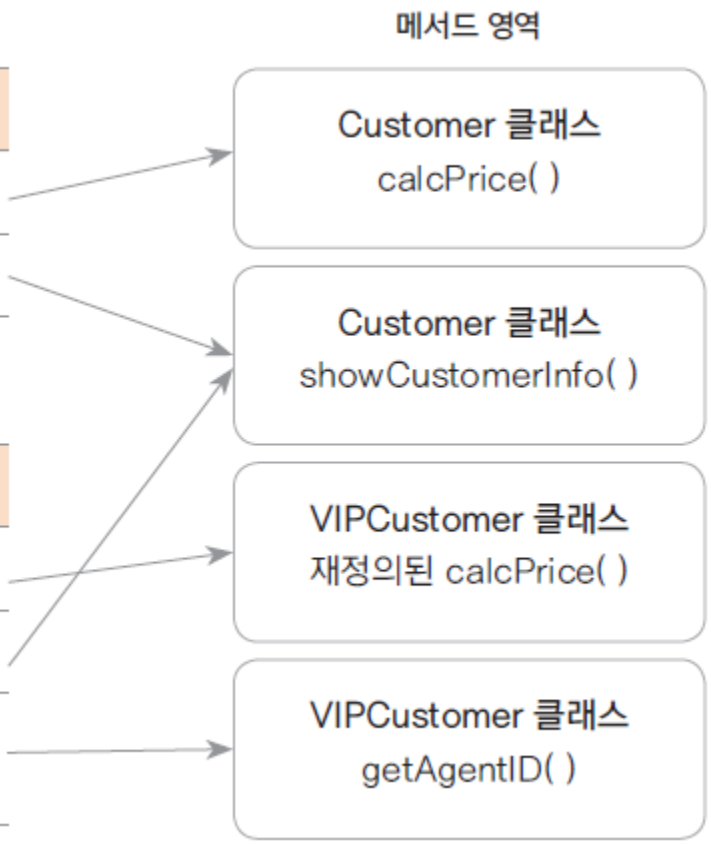
가상 메서드(virtual method)

Customer 클래스의 가상 메서드 테이블

메서드	메서드 주소
calcPrice (재정의됨)	0xFF00FFAA
showCustomerInfo (재정의되지 않음)	0x112233AA

VIPCustomer 클래스의 가상 메서드 테이블

메서드	메서드 주소
calcPrice (재정의됨)	0x00335577
showCustomerInfo (재정의되지 않음)	0x112233AA
getAgentID (하위 클래스에서 추가된 메서드)	0x8899BB33



다형성(polymorphism)

다형성 : 하나의 코드가 여러가지 자료형으로 구현되어 실행되는 것

정보은닉, 상속 과 더불어 객체지향 프로그래밍의 가장 큰 특징 중 하나

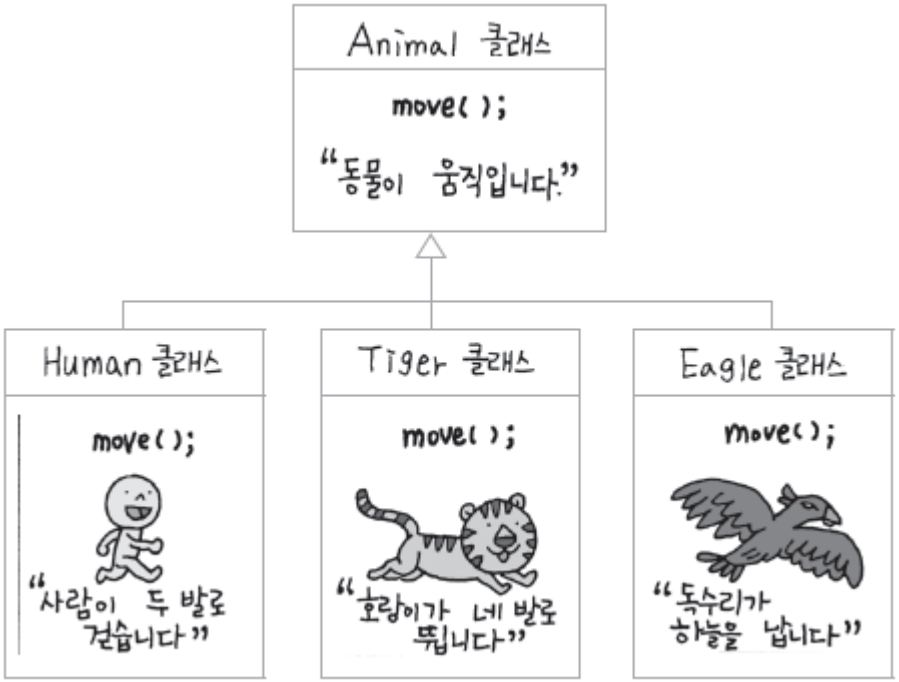
객체지향 프로그래밍의 유연성, 재활용성, 유지보수성에 기본이 되는 특징임

다형성 구현 하기

하나의 클래스를 상속 받은 여러 클래스가 있는 경우

각 클래스마다 같은 이름의 서로 다른 메서드를 재정의 함

상위 클래스 타입으로 선언된 하나의 변수가 여러 인스턴스에 대입되어 다양한 구현이 실행될 수 있음



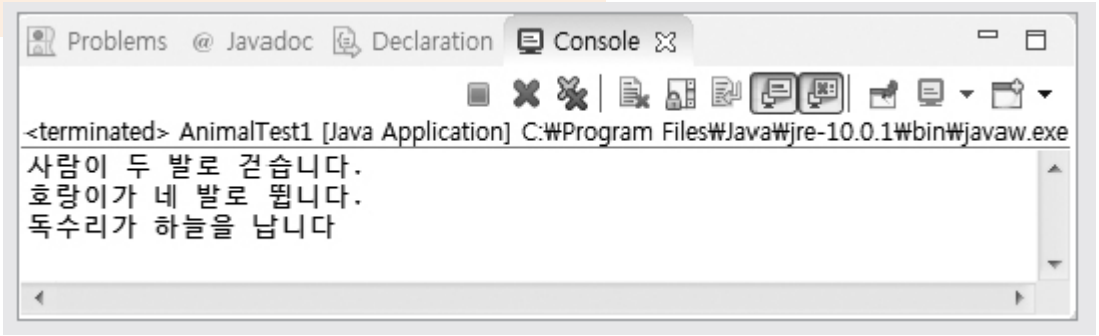
다형성 구현 하기

```
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 됨

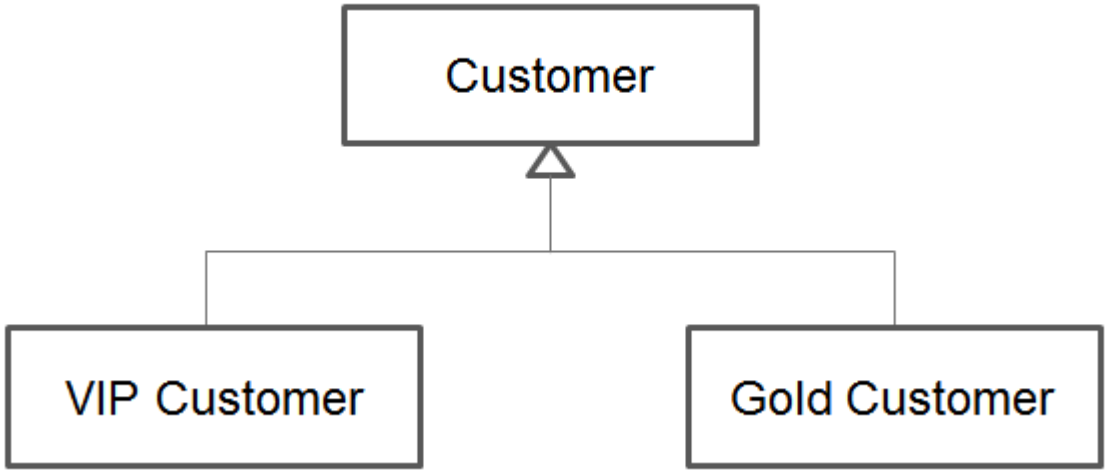


다형성 활용하기

일반 고객과 VIP 고객의 중간 등급의 고객을 생성

5명의 고객을 ArrayList 에 생성하여 저장한 다음

각 고객이 물건을 샀을 때의 가격과 보너스 포인트를 계산 함



예제는 책을 참고 하세요

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 if 문이 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 **Subject**를

포함한 관계

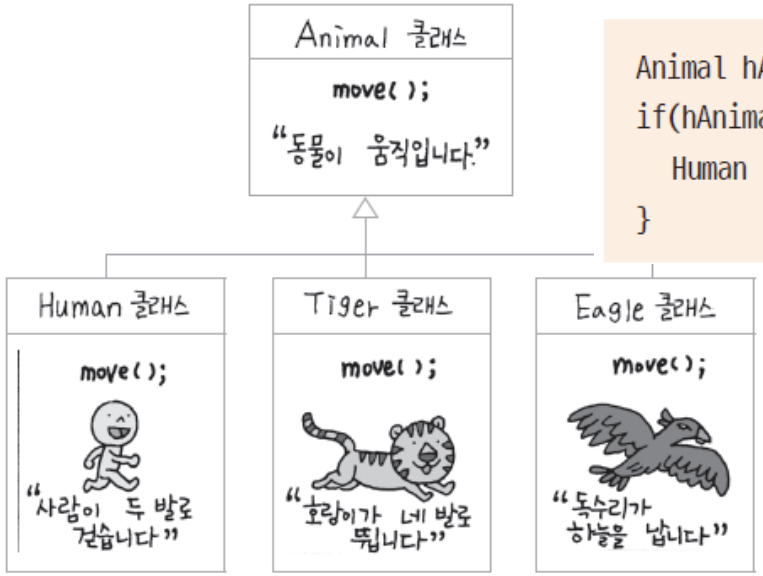
```
class Student {
  Subject majorSubject;
}
```

다운 캐스팅 - instanceof

하위 클래스가 상위 클래스로 형 변환 되는 것은 묵시적으로 이루어짐

다시 원래 자료 형인 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
}
```