JAVA 프로그래밍

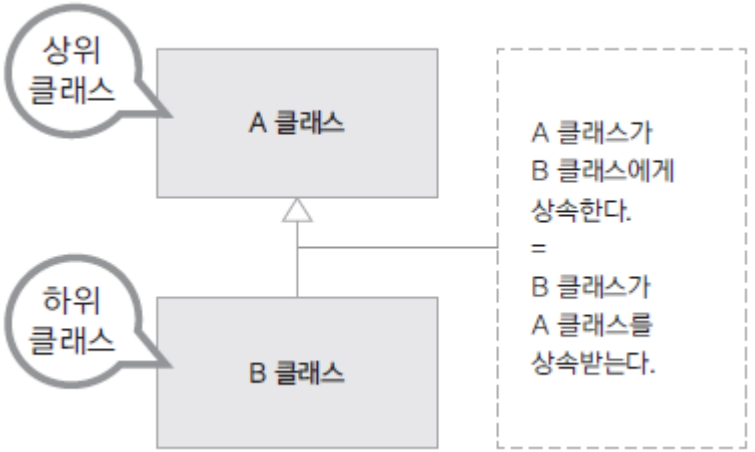
상속

상속이란?

- 클래스를 정의 할 때 이미 구현된 클래스를 상속(inheritance) 받아 속성이나 기능이 확장되는 클래스를 구현함
- 상속하는 클래스 : 상위 클래스, parent class, base class, super class
- 상속받는 클래스 : 하위 클래스, child class, derived class, sub class
- 클래스 상속 문법

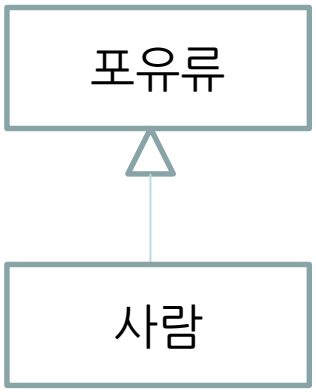
```
class B extends A {


}
```



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 inheritance**만을 지원함**

상속의 장점

- 상속을 통하여 기존 클래스의 필드와 메서드를 재사용
- 기존 클래스의 일부 변경도 가능
- 상속을 이용하게 되면 복잡한 GUI 프로그램을 순식간에 작성
- 상속은 이미 작성된 검증된 소프트웨어를 재사용
- 신뢰성 있는 소프트웨어를 손쉽게 개발, 유지 보수
- 코드의 중복을 줄일 수 있다.

상속의 정의

- 기존의 클래스를 재사용해서 새로운 클래스를 작성하는 것
- 두 클래스를 부모와 자식으로 관계를 맺어주는 것
- **자식은 부모의 모든 멤버를 상속받는다.**
- 자식의 멤버 개수는 부모보다 작을 수 없다. (같거나 많아야 한다.)

```
class Point {
    int x;
    int y;
}
```

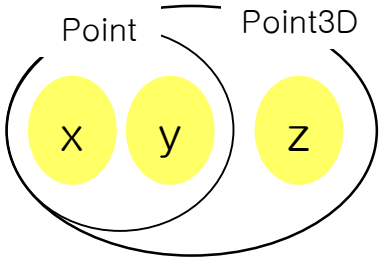
class 자식 클래스 extends 부모 클래스 {

}

```
class Point3D {
    int x;
    int y;
    int z;
}
```

→

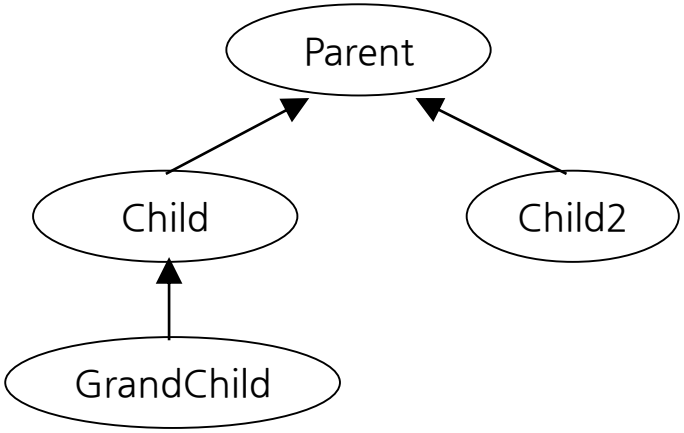
```
class Point3D extends Point {
    int z;
}
```

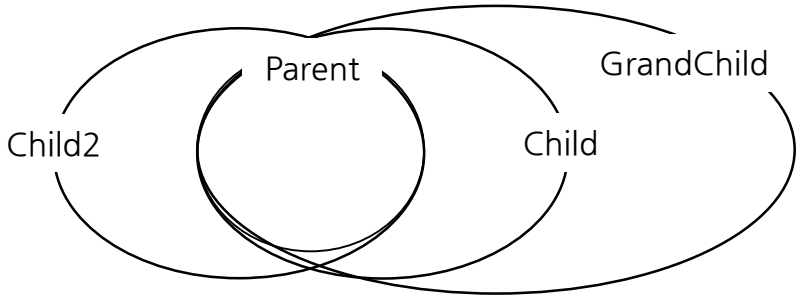


클래스 간의 관계 - 상속 관계

- 공통 부분은 부모 클래스에서 관리하고, 개별 부분은 자식 클래스에서 관리한다.
- 부모 클래스의 변경은 자식 클래스에는 영향을 미치지만, 자식 클래스의 변경은 부모 클래스에 영향을 미치지 않는다.

```
class Parent {}
class Child extends Parent {}
class Child2 extends Parent {}
class GrandChild extends Child {}
```

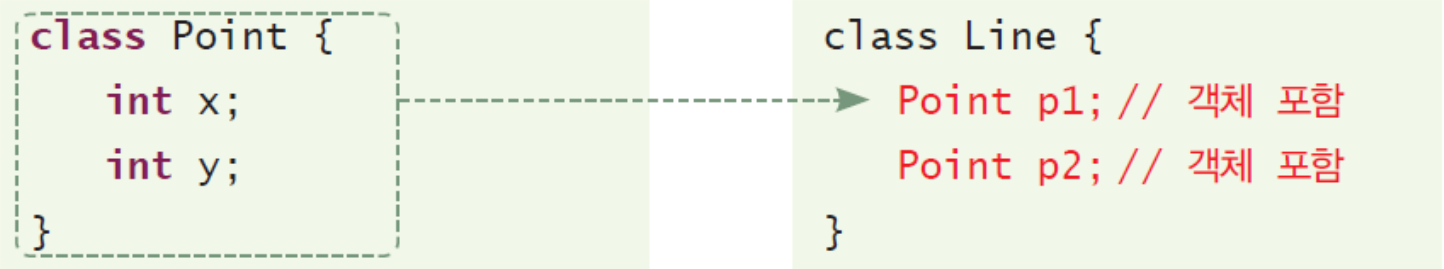




상속 is-a 관계

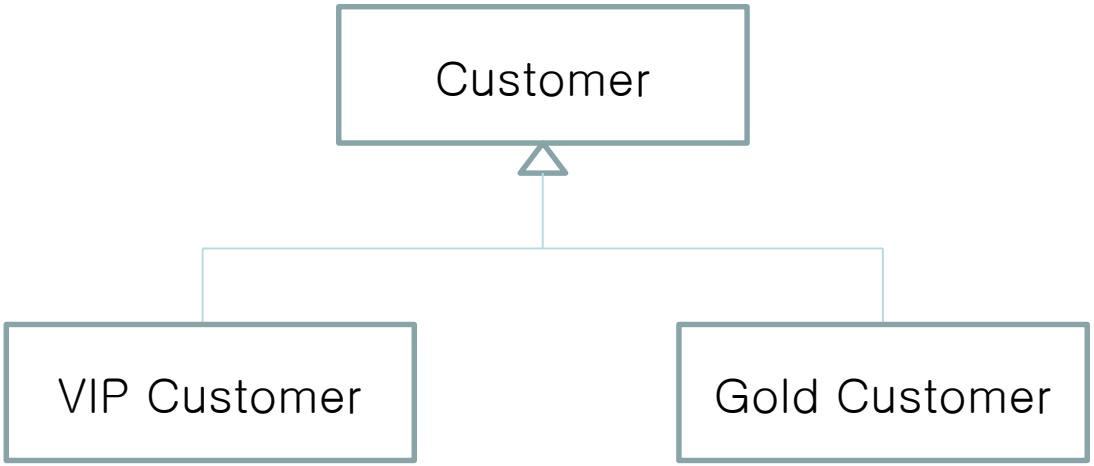
- **상속 관계 : is-a 관계**
  - 자동차는 탈것이다. (Car **is a** Vehicle)
  - 사자, 개, 고양이는 동물이다.

- **포함 관계 : has-a 관계**
  - 도서관은 책을 가지고 있다. (Library **has a** book)
  - 거실은 소파를 가지고 있다.



상속을 활용한 고객관리 프로그램

- ✓ 고객의 정보를 활용하여 고객 맞춤 서비스를 구현
- ✓ 고객의 등급에 따라 차별화 된 할인율과 포인트를 지급
- ✓ 등급에 따른 클래스를 따로 구현하는 것이 아니라,
  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 CustomerGrade는 SILVER이고, bonusRatio는 1%로 지정함
calcPrice(int price)	제품에 대해 지불해야 하는 금액을 계산하여 반환. 할인되지 않는 경우 가격을 그대로 반환. 가격에 대해 보너스 포인트 비율을 적용하여 보너스 포인트를 적립
showCustomerInfo()	고객정보를 출력 고객이름, 등급, 현재 적립된 포인트 정보를 보여줌

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 필요함
- 우수 고객을 관리 하기 위해 다음과 같은 혜택을 줌
- 고객 등급 : VIP
- 제품 구매 할인율 : 10%
- 보너스 포인트 : 5%
- 담당 전문 상담원 배정

VIP 클래스 정의하기

- 이미 구현되어 있는 Customer 클래스를 상속 받고,
  Customer 클래스에 구현되지 않은 속성과 메서드를 추가적으로 구현함

```
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)의 private으로 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 private과 동일한 기능

- 단, 동일 패키지 내에서는 가시성이 허용됨

접근 제어자 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 protected로 선언함

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

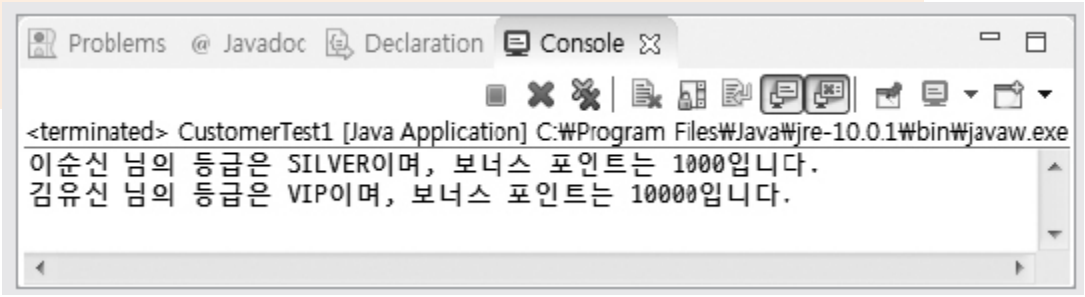
상속 클래스 테스트

```
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



Quiz 상속

〈InheritanceTest.java〉

Person 클래스에서 상속받은 Man 클래스와 Woman 클래스를 만들고, 출력 결과와 같이 구현하시오.

[Person 클래스]

```
private int no;    // 일련번호
private int age; // 나이
```

[출력 결과]

```
남자의 일련번호 : 871023
남자의 나이 : 27
남자의 성별 : M
여자의 일련번호 : 891225
여자의 나이 : 29
여자의 키 : 165
```

Quiz single inheritance

〈SingleInheritance.java〉

[클래스 정의]

- Super(부모 클래스)
- Sub1(Super의 자식 클래스)
- Sub2(Sub1의 자식 클래스)

[main() 정의]

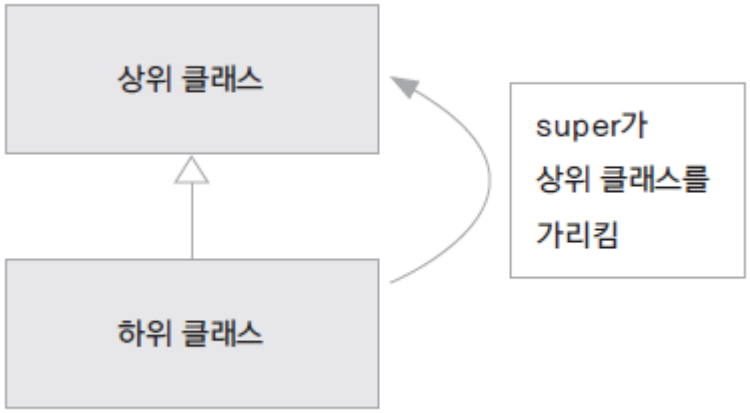
Sub2 클래스로 객체 생성하여 모든 인스턴스 변수와 메서드에 접근

[출력 결과]

```
Super 클래스 생성자()
Sub1 클래스 생성자()
Sub2 클래스 생성자()
Sub1 클래스의 count값 : 10
Super 클래스의 num값 : 20
Sub2 클래스의 id값 : 30
```

super 예약어

- this가 자기 자신의 인스턴스의 주소를 가지는 것처럼
  super는 하위 클래스가 상위 클래스에 대한 주소를 가지게 됨
- 하위 클래스가 상위 클래스에 접근 할 때 사용할 수 있음



✓ 메서드를 재정의할 때, 부모 클래스의 메서드를 완전히 다른 것으로 바꾸는 경우 보다 내용을 추가하는 경우가 많다. 이런 경우에 super 키워드를 사용하여 부모 클래스의 메서드를 호출한 뒤 필요한 부분을 추가하여 사용하면 된다.

super() : 상위 클래스의 생성자

```
class Point   {
   int x;
   int y;
   Point(int x, int y)  {
      this.x = x;
      this.y = y;
   }
}

class Point3D extends Point {
   int z;
   Point3D() {
      this(100, 200, 300);
     // Point3D(int x, int y, int z)를 호출
   }
   Point3D(int x, int y, int z) {
      super(x, y);
      // Point(int x, int y)를 호출
      this.z = z;
   }
}


public class  ConstructorSuper {
   public static void main(String args[]) {
      Point3D p3 = new Point3D();
      System.out.println("p3.x = " + p3.x);
      System.out.println("p3.y = " + p3.y);
      System.out.println("p3.z = " + p3.z);
   }
}
```

상속에서 클래스 생성 과정

- 하위 클래스가 생성될 때 상위 클래스가 먼저 생성됨
- 상위 클래스의 생성자가 호출되고 하위 클래스의 생성자가 호출됨
- 하위 클래스의 생성자에서는 무조건 상위 클래스의 생성자가 호출되어야 함
- 아무것도 없는 경우, 컴파일러는 상위 클래스의 기본 생성자를 호출하기 위한 super()를 코드에 넣어줌
- super() 키워드로 호출되는 생성자는 상위 클래스의 기본 생성자!
- 만약 상위 클래스의 기본 생성자가 없는 경우(매개변수가 있는 생성자만 존재하는 경우) 하위 클래스는 명시적으로 상위 클래스를 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

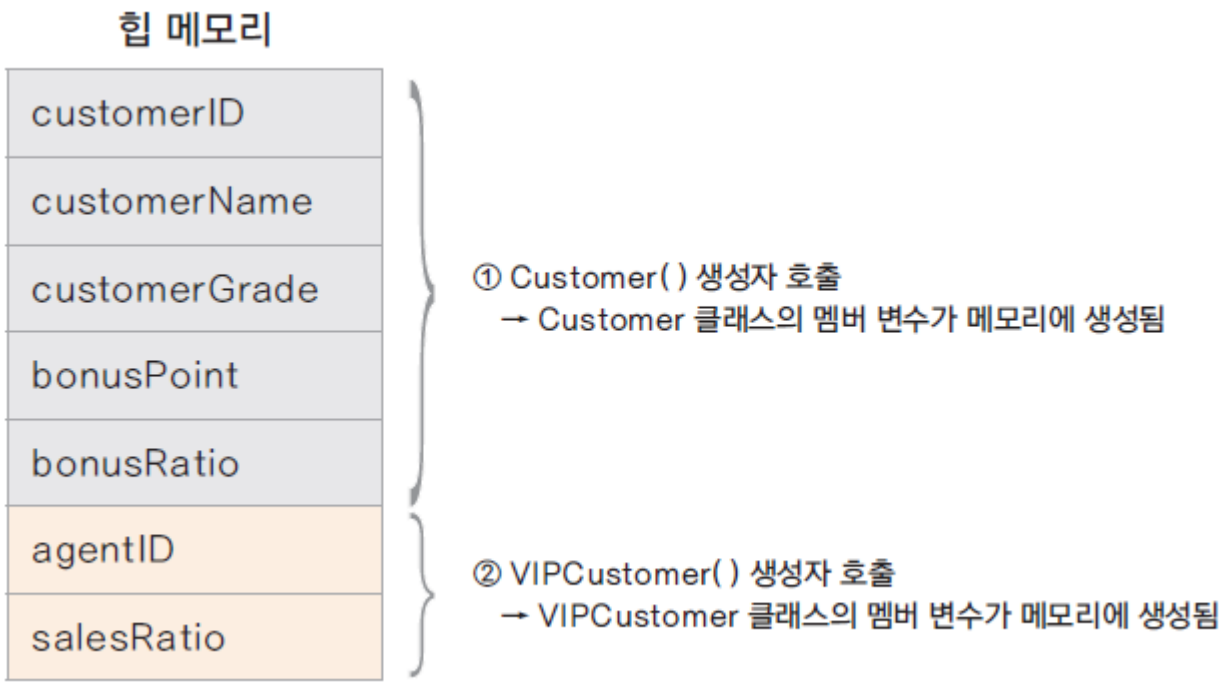
- 명시적으로 상위 클래스 생성자를 호출하는 경우

```
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 생성됨



Quiz super

〈SuperTest.java〉

[부모 클래스] Mother

- 인스턴스 변수 : name
- 매개변수 생성자
- 리턴값 없는 메서드 : display() { name 출력 }

[자식 클래스] Son

- 인스턴스 변수 : name
- 매개변수 생성자
- display() 메서드 오버라이딩 : mother name과 son name 출력, super 사용

[main() 메서드] 자식 클래스의 객체 생성하여 display() 호출

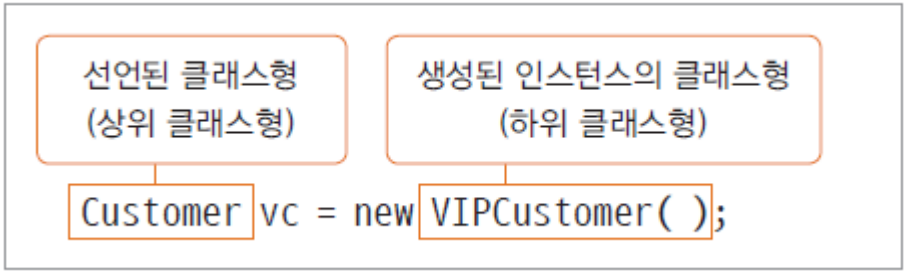
[실행 결과]

mother name : 어머니 신사임당

son name : 아들 율곡이이

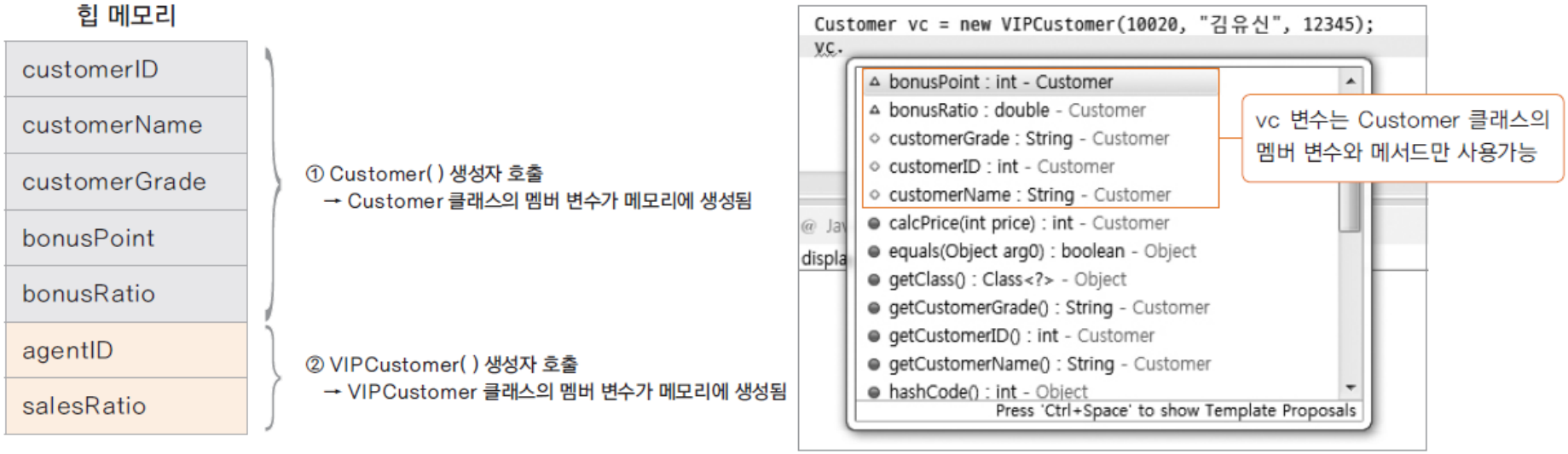
상위 클래스로의 묵시적 형 변환 (업캐스팅)

- 상위 클래스 형으로 변수를 선언하고 하위 클래스 인스턴스를 생성할 수 있음
- 하위 클래스는 상위 클래스의 타입을 내포하고 있으므로 상위 클래스로 묵시적 형 변환이 가능



형 변환에서의 메모리

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



- ✓ VIPCustomer() 생성자의 호출로 인스턴스는 모두 생성 되었지만
- ✓ 타입이 Customer 이므로 접근할 수 있는 변수나 메서드는 Customer의 변수와 메서드임

메서드 재정의(overriding)

- 자식 클래스가 필요에 따라 상속을 이용하여
  부모 클래스의 메서드를 변경하여 사용하는 것

- **주의할 점**
  - 메서드의 헤더(머리 부분)은 그대로 두고, 바디(몸체 부분)만 교체하는 것
  - 메서드의 헤더 부분은 부모 클래스의 헤더와 동일해야 함
    (메서드 이름, 반환 타입, 매개 변수의 개수와 데이터 타입이 일치.)
  - 메서드가 **public**으로 선언되어 있는 경우에만 **재정의가 가능**
    private 메서드는 오버라이딩 불가능!!
  - 부모 클래스의 메서드 보다 더 좁은 범위의 접근 제어자로 변경할 수 없음
    - *예를 들어, 부모 클래스의 메서드가 protected로 선언되어 있는 경우,*
      자식 클래스의 재정의 메서드는 protected나 public으로만 선언 가능하다.

메서드 재정의(overriding)

```
class Animal {                          // 부모 클래스
   public void sound() {                // 부모 클래스에서 정의된 메서드
      System.out.println(" 으르렁!");
   }
}

class Dog extends Animal {              // 자식 클래스
   public void sound() {                // 메서드 오버라이딩
      System.out.println("멍멍!");
   }
}

public class MethodOverriding {
   public static void main(String[] args) {
         Dog d = new Dog();
         d.sound();
   }
}
```

메서드 오버라이딩(overriding)

- 상위 클래스에 정의된 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 할인율을 적용한 가격을 반환

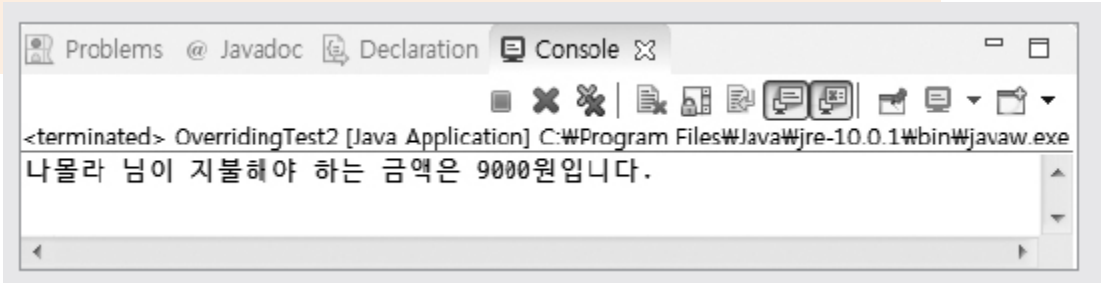
묵시적 형 변환과 재정의 된 메서드 호출

```
Customer vc = new VIPCustomer();
vc.calcPrice(10000);
```

- 위 코드에서 calcPrice() 메서드는 어느 메서드가 호출될 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 호출되는 원리

```
Customer vc = new VIPCustomer();
vc.calcPrice(10000);
```

- vc의 타입은 Customer 이지만, 실제 생성된 인스턴스인 VIPCustomer 클래스의 calcPrice() 메서드가 호출됨

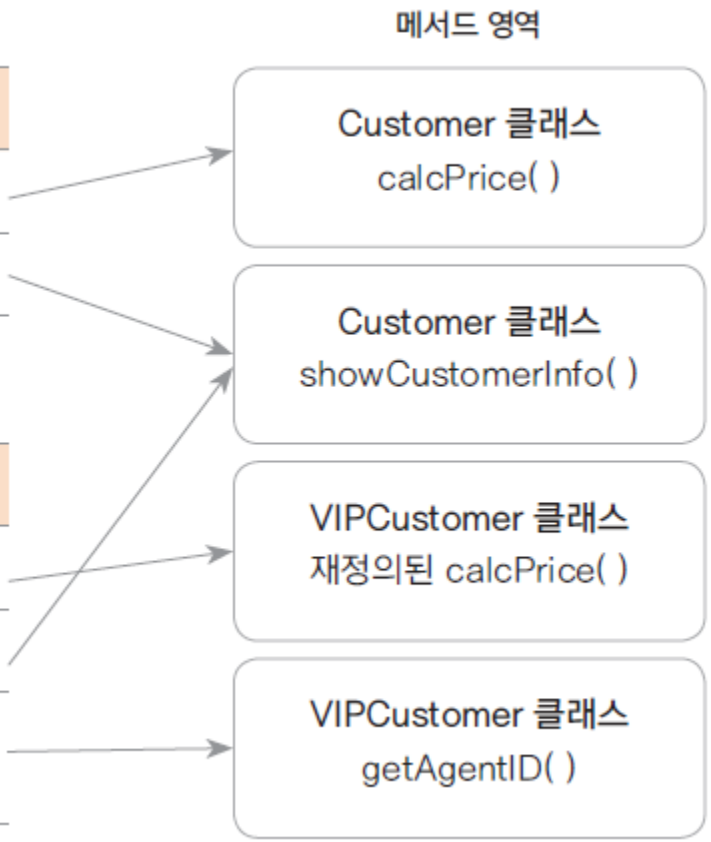
가상 메서드(virtual method)

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



Quiz 오버라이딩(Overriding)

〈OverrideTest.java〉

[클래스 정의]

- 부모 클래스 : Employee
  - name과 age 멤버 변수
- 자식 클래스 : Manager
  - job 멤버 변수

[출력 결과]

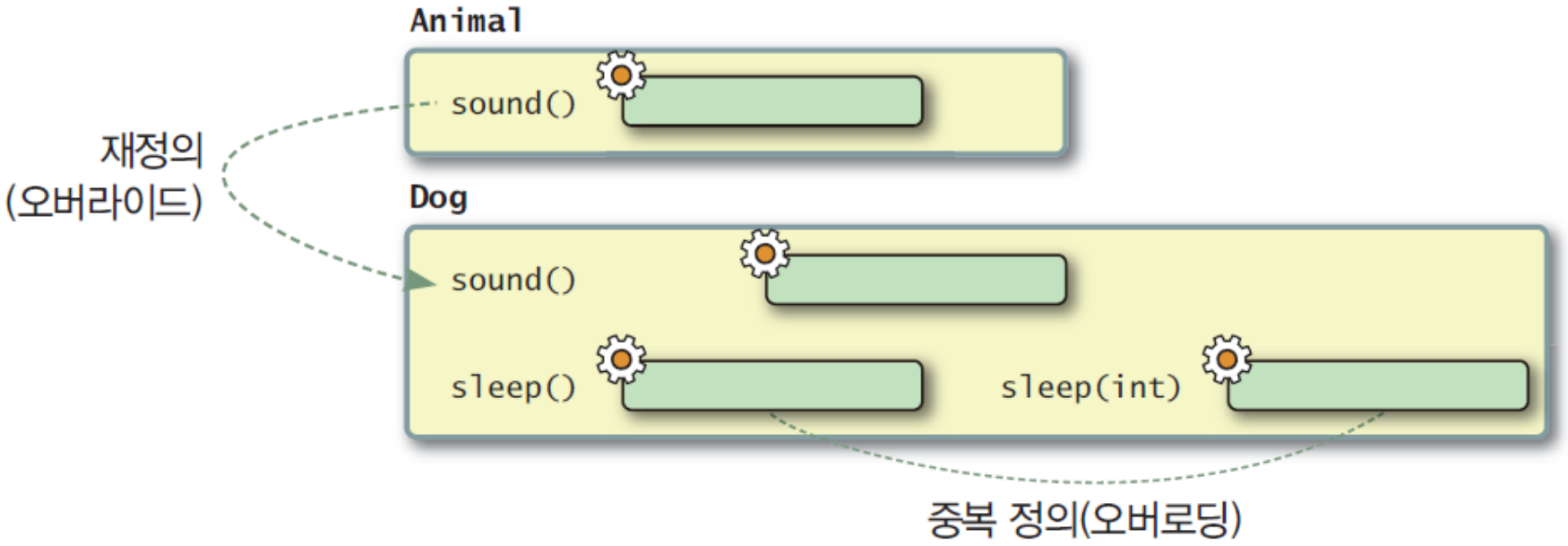
사원의 이름은 James Kim이고, 나이는 30입니다.
James Kim 담당은 프로젝트 매니저입니다.

사원의 이름은 Tomas Lee이고, 나이는 15입니다.
Tomas Lee 담당은 싱어송 라이터입니다.

재정의(overriding) vs. 중복정의(overloading)

- **메서드 오버라이딩**(overriding) : 부모 클래스를 상속받은 자식 클래스에서 부모의 메서드를 변경하여 사용하는 것 **[재정의]**

- **메서드 오버로딩**(overloading) : 같은 클래스 안에서 이미 정의된 메서드를 변경하여 사용하는 것 **[중복정의]**



메서드 오버로딩(Method Overloading)

```
class MethodOverload{
  // test() 메서드
  void test() {
    System.out.println("매개변수 없음"); }

  // test() 메서드의 오버로딩
  void test(int a, int b) {
    System.out.println("매개변수 : "+ a
    + ", " + b);
  }

   // test() 메서드의 오버로딩
  void test(double d) {
    System.out.println("매개변수 : " + d);
  }
}
```

```
public class Overloading {
  public static void main(String [] args) {
  // 객체 생성
   MethodOverload  obj
          = new MethodOverload();

  // test() 호출
  obj.test();

  // test(int a, int b) 호출
  obj.test(10, 20);

  // test(double d) 호출
  obj.test(123.4);
  }
}
```

Quiz Method Overloading

〈OverloadingTest.java〉

[Over 클래스 정의]

- void method() { System.out.println("첫 번째 메서드 호출"); }
- 메서드 오버로딩1 : method()의 오버로딩, 매개변수(int a, int b)
- 메서드 오버로딩2 : method()의 오버로딩, 매개변수(String a, int b)
- 메서드 오버로딩3 : method()의 오버로딩, 매개변수(int b, String a)
- 원의 넓이 메서드 : void area(int r), 원의 넓이=반지름*반지름*원주율
- 사각형의 넓이 메서드 : area() 메서드의 오버로딩, 사각형의 넓이=가로*세로

[main() 메서드 정의] 객체 생성하여 6개의 메서드 호출

[실행 결과]

첫 번째 메서드 호출
메서드 오버로딩1 호출
메서드 오버로딩2 호출
메서드 오버로딩3 호출
원의 넓이 : 78.5
사각형의 넓이 : 200

toString()

➢ Object 클래스의 toString() 메서드 : **객체가 가지는 값을 문자열 형태로 반환,**
객체가 생성되면 자동 호출. (→ overriding 하여 사용)

```
Class Car {
  private String model;
  public Car(String model) { this.model= model; }
  public String toString() { return "Car 모델명 : " + model; } // 오버라이딩
}

public class ToString {
  public static void main(String[] args) {
    Car obj = new Car(“HYBRID CAR");
    System.out.println(obj.toString());
    System.out.println(obj);   // toString() 메서드는 자동 호출되므로 생략 가능
  }
}
```

Quiz

〈HumanTest.java〉

[부모 클래스] Human

- private 접근 제어로 name과 age 변수 선언
- 디폴트 생성자 정의, 매개변수 생성자(name, age) 정의
- toString() 메서드 정의(이름, 나이 반환)

[자식 클래스] Woman

- private 접근 제어로 job 변수 선언
- 매개변수 생성자(name, age, job) 정의
- toString() 메서드 오버라이딩(이름, 나이, 직업 반환)

[main() 메서드]

- 부모 클래스의 객체 3개 생성 : (춘향, 18) (몽룡, 21) (사또, 50)
- 부모 클래스의 toString() 메서드 호출하여 출력
- 자식 클래스의 객체 3개 생성 :

  (지윤, 30, 판사) (시우, 27, 디자이너) (혜정, 25, 프로그래머)
- 자식 클래스의 toString() 메서드 호출하여 출력

[실행 결과]

이름 : 춘향, 나이 : 18세
이름 : 몽룡, 나이 : 21세
이름 : 사또, 나이 : 50세

이름 : 지윤, 나이 : 30세, 직업 : 판사
이름 : 시우, 나이 : 27세, 직업 : 디자이너
이름 : 혜정, 나이 : 25세, 직업 : 프로그래머

Quiz

[부모 클래스] Animal

- name 변수 선언 : protected 접근 제어자
- 디폴트 생성자 정의 : 내용 없음
- 매개변수 생성자(name) 정의 { name 변수 초기화, 이름 출력 }
- sound() 메서드 { 출력문("동물 울음 소리") }
- eat() 메서드 { 출력문("동물 먹이") }

[Animal의 자식 클래스] Tiger

- 디폴트 생성자 정의 { 출력문("고양이과 동물입니다.") }
- 매개변수 생성자(name) 정의 { 부모의 생성자 호출(super 사용), 출력문 }
- sound() 메서드의 오버로딩 : sound(String cry) { name의 울음소리 출력 }
- eat() 메서드의 오버로딩 : eat(String food) {
  "호랑이는 OO을 잡아먹는다" 출력 }

[Animal의 자식 클래스] Cat

- 디폴트 생성자 정의 { }
- CatName(String name) 메서드 생성 :
  { "name은 고양이입니다." 출력 }
- sound() 메서드의 오버로딩 :
  sound(String name, String cry) { name의 울음소리 출력 }
- eat() 메서드의 오버로딩 :
  eat(String food) { "고양이는 OO을 잡아먹는다" 출력 }

[main() 메서드]

- Tiger 클래스의 객체 2개 생성(디폴트, 매개변수)
- Tiger 클래스의 sound()와 eat() 메서드 호출
- Cat 클래스의 객체 생성
- Cat 클래스의 CatName(), sound(), eat() 메서드 호출

[실행 결과]

고양이과 동물입니다.

이름 : 백두

호랑이입니다.

백두 울음소리 : 으르렁

호랑이는 사슴을 잡아먹는다.

루나는 고양이입니다.

루나 울음소리 : 야옹

고양이는 생쥐를 잡아먹는다.