**[ 12 ] 추상클래스**

목표: 추상클래스의 이해와 필요이유, 문법. 추상클래스 상속과 일반 클래스의 상속의 차이를 설명할 수 있다.

(cf) 다형성(Polymorphism) : 같은 형태이지만 기능적으로 여러 다양한 객체를 이용할 수 있는 성질을 말한다. 프로그래밍 입장에서 상속받는 여러 객체를 대입함으로 다양한 기능을 이용할 수 있다.

1. 추상클래스의 이해

※추상적으로 정의할 테니(선언해 놓을테니), 객체를 사용할 사용자가 꼭 재정의(overriding) 하세요!

ex. 한가지 간단한 상속을 통한 예제를 살펴보도록 하겠습니다.

시나리오는 어제 살펴본 음식점 체인 사업관련 예제입니다. 우선 본사에서 메뉴에 대한 가격 메소드를 정해 주고, 매장에서는 주변 환경에 맞게 가격을 책정(overriding) 합니다. 단. 지난번과 달리 본사에서는 메뉴만 정해 주고, 가격을 매장에 전부 위임 합니다.

본사 :김치찌개-0 부대찌개-0 비빔밥-0 순대국-0 공기밥-0

1호점:김치찌개-4,500 부대찌개-5,000 비빔밥-6,000 순대국-안팔아 공기밥-1,000원

2호점:김치찌개-5,000 부대찌개-5,000 비빔밥-5,000 순대국-5,000 공기밥-무료

3호점:김치찌개-6,000 부대찌개-7,000 비빔밥-7,000 순대국-6,000 공기밥-1,000원

**public** **class** HeadQuarterStore {

**private** String str = "본사지침";

**public** **void** kimchi() { System.***out***.println("김치찌개 0원. 주변환경에 맞게 책정하세요");}

**public** **void** bude() {System.***out***.println("부대찌개 0원. 주변환경에 맞게 책정하세요");}

**public** **void** bibib() {System.***out***.println("비빔밥 0원. 주변환경에 맞게 책정하세요");}

**public** **void** sunde() {System.***out***.println("순대국 0원. 주변환경에 맞게 책정하세요");}

**public** **void** gonggibab() {System.***out***.println("공기밥 0원. 주변환경에 맞게 책정하세요");}

**public** String getStr() { **return** str; }

**public** **void** setStr(String str) { **this**.str = str; }

}

**public** **class** StoreNum1 **extends** HeadQuarterStore{

**private** String str = "\* \* 1호점 \* \*";

@Override

**public** **void** kimchi() {System.***out***.println("김치찌개 4,500원");}

@Override

**public** **void** bude() {System.***out***.println("부대찌개 5,000원");}

@Override

**public** **void** sunde() {System.***out***.println("순대국 안팔아");}

**public** String getStr() {**return** str;}

**public** **void** setStr(String str) {**this**.str = str;}

}

**public** **class** StoreNum2 **extends** HeadQuarterStore{

**private** String str = "\* \* 2호점 \* \*";

@Override

**public** **void** bude() {System.***out***.println("부대찌개 5,000원");}

@Override

**public** **void** bibib() {System.***out***.println("비빔밥 5,000원");}

@Override

**public** **void** gonggibab() {System.***out***.println("공기밥 무료");}

**public** String getStr() {**return** str;}

**public** **void** setStr(String str) {**this**.str = str;}

}

**public** **class** StoreNum3 **extends** HeadQuarterStore{

**private** String str = "\* \* 3호점 \* \*";

@Override

**public** **void** kimchi() {System.***out***.println("김치찌개 6,000원");}

@Override

**public** **void** bude() {System.***out***.println("부대찌개 7,000원");}

@Override

**public** **void** bibib() {System.***out***.println("비빔밥 7,000원");}

@Override

**public** **void** sunde() {System.***out***.println("순대국 6,000원");}

**public** String getStr() {**return** str;}

**public** **void** setStr(String str) {**this**.str = str;}

}

**public** **class** A08TestMain {

**public** **static** **void** main(String[] args) {

HeadQuarterStore st = **new** HeadQuarterStore();

StoreNum1 st1 = **new** StoreNum1();

StoreNum2 st2 = **new** StoreNum2();

StoreNum3 st3 = **new** StoreNum3();

HeadQuarterStore[] store = {st, st1, st2, st3};

**for**(HeadQuarterStore s : store) {

System.***out***.println(s.getStr());

s.kimchi();

s.bude();

s.bibib();

s.sunde();

s.gonggibab();

}//for

}//main

}//class

살펴본 예제와 같이 본사에서 가격을 정하지 않았고, 매장에서 가격을 정(재정의)하지 않았더니, 문제가 발생 했습니다.

이런 문제를 사전에 예방하려면 ★매장이 오픈 예정일 때, 본사에서는 ‘모든 메뉴의 가격을 정하세요.’ 라고 강요 하면 됩니다.

**JAVA프로그램에서도 강제로 부모클래스에서 자식클래스에게 메소드를 강제로 재정의(override)하게 할 수 있습니다. 그리고 이러한 방법으로 만들어진 클래스를 ‘추상클래스’라고 합니다**.★

예제를 통해 살펴보도록 하겠습니다.

**public** **abstract** **class** HeadQuarterStore {

**public** **abstract** **void** kimchi();

**public** **abstract** **void** bude();

**public** **abstract** **void** bibib();

**public** **abstract** **void** sunde();

**public** **abstract** **void** gonggibab();

}

**public** **class** StoreNum1 **extends** HeadQuarterStore {

**public** **void** kimchi() { System.***out***.println("김치찌개 4,500원"); }

**public** **void** bude() { System.***out***.println("부대찌개5,000원"); }

**public** **void** sunde() { System.***out***.println("순대국 안팔아"); }

**public** **void** bibib() { System.***out***.println("비빔밥 6,000원");}

**public** **void** gonggibab() {System.***out***.println("공기밥 1,000원");}

}

**package** com.ch.ex0store;

**public** **class** StoreNum2 **extends** HeadQuarterStore {

@Override

**public** **void** bude() {System.***out***.println("부대찌개 5,000원");}

@Override

**public** **void** bibib() { System.***out***.println("비빔밥 5,000원");}

@Override

**public** **void** gonggibab() { System.***out***.println("공기밥 무료");}

@Override

**public** **void** kimchi() {System.***out***.println("김치찌개 5,000원");}

@Override

**public** **void** sunde() {System.***out***.println("김치찌개 5,000원");}

}

**package** com.ch.ex0store;

**public** **class** StoreNum3 **extends** HeadQuarterStore {

@Override

**public** **void** kimchi(){System.***out***.println("김치찌개 6,000원");}

@Override

**public** **void** bude(){System.***out***.println("부대찌개 7,000원");}

@Override

**public** **void** bibib(){System.***out***.println("비빔밥 7,000원");}

@Override

**public** **void** sunde(){System.***out***.println("순대국 6,000원");}

@Override

**public** **void** gonggibab() {System.***out***.println("공기밥 1,000원");

}

}

**package** com.ch.ex0store;

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

System.***out***.println("★ ★ 본사");

HeadQuarterStore st = **new** HeadQuarterStore() {

@Override

**public** **void** sunde() { }

@Override

**public** **void** kimchi() {System.***out***.println("50원");}

@Override

**public** **void** gonggibab() { }

@Override

**public** **void** bude() { }

@Override

**public** **void** bibib() { }

};

st.kimchi(); st.bude(); st.bibib();

st.sunde(); st.gonggibab();

System.***out***.println("★ ★ 주택가 1호점");

HeadQuarterStore st1 = **new** StoreNum1();

st1.kimchi(); st1.bude(); st.bibib();

st1.sunde(); st1.gonggibab();

System.***out***.println("★ ★ 대학가 2호점");

HeadQuarterStore st2 = **new** StoreNum2();

st2.kimchi(); st2.bude(); st2.bibib();

st2.sunde(); st2.gonggibab();

System.***out***.println("★ ★ 증권가 3호점");

HeadQuarterStore st3 = **new** StoreNum3();

st3.kimchi(); st3.bude(); st3.bibib();

st3.sunde(); st3.gonggibab();

}

}

2. 추상클래스의 문법(※추상메소드에서는 정의만 합니다. 구현하지 않습니다)

① abstract(추상클래스 및 추상 메소드를 선언하는 예약어)이용

public abstract class ClassName {

…

}

② 추상클래스에는 하나 이상의 추상 메소드가 포함.

③ 추상메소드에는 메소드의 선언부만 있고 실행부(구현부, Body)는 없습니다

abstract 리턴타입 methodName([매개변수])

④ 추상 클래스에서는 메소드 선언만 하고 실제로 구현은 상속받는 클래스에서 한다.

기능는 자식 클래스에게 위임 - 추상클래스에서 정의된 추상적인 기능은 하위 클래스에서 상세 구현

⑤ 클래스의 프레임만 구성. 직접 객체 생성 불가능(abstract는 인스턴스화를 금지하는 키워드)

<예제1>

**public** **abstract** **class** SuperClass {

**public** SuperClass() {}

**public** **abstract** **void** method1();

**public** **void** method2() {System.***out***.println("Super의 method2()");

}

**public** **abstract** **class** ChildClass **extends** SuperClass {

//method1() - 추상메소드

@Override

**public** **void** method2() {System.***out***.println("ChildClass의 method2()");}

**public** **void** method3() {System.***out***.println("ChildClass의 method3()");}

}

**public** **class** GrandChildClass **extends** ChildClass{

@Override

**public** **void** method1() {System.***out***.println("GrandChildClass의 method1()");}

// method2() - ChildClass에서 상속

// method3() - ChildClass에서 상속

}

**public** **class** Ex01TestMain {

**public** **static** **void** main(String[] args) {

// SuperClass superClass = new SuperClass();

// ChildClass childClass = new ChildClass();

GrandChildClass g = **new** GrandChildClass();

g.method1();

g.method2();

g.method3();

ChildClass c = **new** GrandChildClass();

c.method1();

c.method2();

c.method3();

SuperClass s = **new** GrandChildClass();

s.method1();

s.method2();

//s.method3();

Object o = **new** GrandChildClass();

((ChildClass)o).method1();

//o.method2();

//o.method3();

}

}

<예제2>

**public** **abstract** **class** Shape {

**public** **abstract** **void** computeArea();

**public** **abstract** **void** draw();

}

**public** **class** Circle **extends** Shape {

**private** **int** r;

**public** Circle() { }

**public** Circle(**int** r) {**this**.r = r;}

@Override

**public** **void** computeArea() {

System.***out***.println("원 넓이 : "+(3.14\*r\*r));

}

@Override

**public** **void** draw() {

System.***out***.println("원을 그려요");

}

}

**public** **class** Triangle **extends** Shape {

**private** **int** w, h;

**public** Triangle() { }

**public** Triangle(**int** w, **int** h){

**this**.w = w;

**this**.h = h;

}

@Override

**public** **void** computeArea() {

System.***out***.println("삼각형 넓이 : "+((w\*h)/2.0));

}

@Override

**public** **void** draw() {

System.***out***.println("삼각형 그려요");

}

}

**public** **class** Rectangle **extends** Shape {

**private** **int** w, h;

**public** Rectangle() { }

**public** Rectangle(**int** w, **int** h){

**this**.w = w;

**this**.h = h;

}

@Override

**public** **void** computeArea() {

System.***out***.println("사각형 넓이 : "+(w\*h));

}

@Override

**public** **void** draw() {

System.***out***.println("사각형을 그려요");

}

}

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

Shape circle = **new** Circle(5);

Shape triangle = **new** Triangle(10,5);

Shape rectangle = **new** Rectangle(10,5);

circle.computeArea();

circle.draw();

triangle.computeArea();

triangle.draw();

rectangle.computeArea();

rectangle.draw();

}

}

※ 부모클래스인 Shape 클래스에 생성자가 없는데도 문제가 발생하지 않는 이유는?

<위의 예제 수정>

**public** **abstract** **class** Shape {

**public** **abstract** **double** computeArea();

**public** **abstract** **void** draw();

}

**public** **class** Circle **extends** Shape {

**private** **int** r;

**public** Circle() { }

**public** Circle(**int** r) {**this**.r = r;}

@Override

**public** **double** computeArea() {

**return** r\*r\*3.14;

}

@Override

**public** **void** draw() {

System.***out***.println("원을 그려요");

}

}

**public** **class** Triangle **extends** Shape {

**private** **int** w, h;

**public** Triangle() { }

**public** Triangle(**int** w, **int** h){

**this**.w = w;

**this**.h = h;

}

@Override

**public** **void** draw() {

System.***out***.println("삼각형 그려요");

}

@Override

**public** **double** computeArea() {

**return** w\*h/2.0;

}

}

**public** **class** Rectangle **extends** Shape {

**private** **int** w, h;

**public** Rectangle() { }

**public** Rectangle(**int** w, **int** h){

**this**.w = w;

**this**.h = h;

}

@Override

**public** **double** computeArea() {

**return** w\*h;

}

@Override

**public** **void** draw() {

System.***out***.println("사각형을 그려요");

}

}

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

Shape circle = **new** Circle(5);

Shape triangle = **new** Triangle(10,5);

Shape rectangle = **new** Rectangle(10,5);

System.***out***.println("원넓이:"+circle.computeArea());

circle.draw();

System.***out***.println("삼각형넓이:"+

triangle.computeArea());

triangle.draw();

System.***out***.println("사각형넓이:"+

rectangle.computeArea());

rectangle.draw();

}

}

3. 추상클래스가 필요한 이유

※강제성을 느낄 때 사용 합니다

객체지향 언어를 하다 보면 추상화 또는 추상개념 등의 ‘추상’이라는 말을 많이 접하게 될 것입니다.

프로그래밍에서 ‘추상＇이라는 것은 일반생활에서의 ‘추상’과 동일 합니다.

어떤 행위(메소드)를 추상적으로 정의만 해놓고, 실제로 필요한 곳에서 필요에 따라서 재정의 하는 방식입니다.

상속을 사용하다가, 자식 클래스에서 강제로 무엇인가 꼭 재정의 해서 사용해야 될 경우가 발생되면, 추상클래스 개념을 이용하면 됩니다

추상 개념은 객체지향 언어에서는 아주 중요한 의미가 있습니다. 그리고 추상적으로 프로그래밍을 한다는 것은 결코 쉬운 일은 아닐 것입니다.

하지만, 예제를 많이 접해보고, 추후에 알아볼 패턴에 대해서 공부를 하고 나면, 추상적으로 프로그래밍 한다는 것이 얼마나 중요한 건지 쉽게 아실 수 있을 것입니다.

4. 추상클래스 상속과 일반적인 상속의 차이

※차이는 있지만, 어느 쪽이 더 좋다는 정답은 없습니다

현장에 있는 개발자들도 항상 고민하는 부분 중 하나가 추상 클래스로 할 것인지 일반 클래스로 할 것인지 고민을 하게 됩니다.

결론부터 말씀드리면, 정답은 없습니다. 그리고 처음부터 100% 확정하지도 않습니다.

초기 설계 단계에서 어느 정도 추상과 일반 클래스를 분리 하기는 하지만, 프로젝트 개발 도중 언제든지 변경하기도 합니다.(리펙토링)

부모 클래스의 메소드를 자식 클래스에서 변경할 일이 없다면, 일반 클래스를 이용한 상속이 좋습니다.

하지만, 이전 예제처럼 자식클래스에서 꼭 재정의 해야 하는 부분이 있다면, 추상클래스를 이용하여 강제로 재정의 하게 해야 됩니다.

추후에 패턴을 살펴볼 때 일반클래스와 추상클래스를 이용한 상속의 차이를 확실하게 느끼실 수 있습니다.

<오늘의 총 실습 예제1>

어린이집 아이들의 식대 계산 프로그램 구현

※시나리오 분석 후 추상클래스와 일반클래스를 생각해 본다

어린이집에 어린이들의 점심 식대 관련하여 프로그래밍을 구현합니다.

어린이들은 점심으로 쌀밥, 불고기, 국이 배식 됩니다. 추가로, 바나나, 우유, 아몬드가 간식으로 배식 됩니다.

기본적인 쌀밥, 불고기, 나물, 미역국은 모든 아이들이 먹습니다. (일반메소드 역할)

하지만, 개인 체질에 따라서 간식은 선택할 수가 있습니다. 특히 알레르기가 있는 아이들은 간식을 전부 선택하지 않을 수도 있습니다.

클래스 설계를 해 보도록 합니다.

LunchMenu : rice, bulgogi, soup(기본주식), banana, milk, almond(간식)

Child1 : rice, bulgogi, banana, almond ☞ Child1형 아이는 간식으로 바나나와 아몬드를 먹습니다

Child2 : rice, bulgogi, milk ☞ Child2형 아이는 간식으로 우유만 먹습니다

**public** **abstract** **class** LunchMenu {

**private** **int** rice;

**private** **int** bulgogi;

**private** **int** soup;//기본 메뉴

**private** **int** banana;

**private** **int** milk;

**private** **int** almond;

**public** LunchMenu() { }

**public** LunchMenu(**int** rice, **int** bulgogi, **int** soup, **int** banana, **int** milk, **int** almond) {

**this**.rice = rice;

**this**.bulgogi = bulgogi;

**this**.soup = soup;

**this**.banana = banana;

**this**.milk = milk;

**this**.almond = almond;

}

**public** **abstract** String getStr();

**public** **abstract** **int** calculate();

**public** **int** getRice() {**return** rice;}

**public** **void** setRice(**int** rice) {**this**.rice = rice;}

**public** **int** getBulgogi() {**return** bulgogi;}

**public** **void** setBulgogi(**int** bulgogi) {**this**.bulgogi = bulgogi;}

**public** **int** getSoup() {**return** soup;}

**public** **void** setSoup(**int** soup) {**this**.soup = soup;}

**public** **int** getBanana() {**return** banana;}

**public** **void** setBanana(**int** banana) {**this**.banana = banana;}

**public** **int** getMilk() {**return** milk;}

**public** **void** setMilk(**int** milk) {**this**.milk = milk;}

**public** **int** getAlmond() {**return** almond;}

**public** **void** setAlmond(**int** almond) {**this**.almond = almond;}

}

**public** **class** Child1 **extends** LunchMenu {

**private** String str = "Child1형 식대:";

**public** Child1() { }

**public** Child1(**int** rice, **int** bulgogi,**int** soup, **int** banana, **int** milk, **int** almond) {

**super**(rice, bulgogi, soup, banana, milk, almond);

}

@Override

**public** **int** calculate() {

**return** getRice()+getBulgogi()+getSoup()+getBanana()+getAlmond();

}

@Override

**public** String getStr() {**return** str;}

}

**public** **class** Child2 **extends** LunchMenu {

**private** String str = "Child1형 식대:";

**public** Child2() { }

**public** Child2(**int** rice, **int** bulgogi,**int** soup, **int** banana, **int** milk, **int** almond) {

**super**(rice, bulgogi, soup, banana, milk, almond);

}

@Override

**public** **int** calculate() {

**return** getRice()+getBulgogi()+getSoup()+getMilk();

}

@Override

**public** String getStr() {**return** str;}

}

**public** **class** PriceTable {

**public** **static** **int** ***RICE*** = 500;

**public** **static** **int** ***BULGOGI*** = 1000;

**public** **static** **int** ***SOUP*** = 500;

**public** **static** **int** ***MILK*** = 750;

**public** **static** **int** ***BANANA*** = 500;

**public** **static** **int** ***ALMOND*** = 200;

}

**public** **class** Ex3TestMain {

**public** **static** **void** main(String[] args) {

Child1 c1 = **new** Child1(PriceTable.***RICE***, PriceTable.***BULGOGI***,

PriceTable.***SOUP***, PriceTable.***BANANA***, PriceTable.***MILK***,

PriceTable.***ALMOND***);

Child2 c2 = **new** Child2(PriceTable.***RICE***, PriceTable.***BULGOGI***,

PriceTable.***SOUP***, PriceTable.***BANANA***, PriceTable.***MILK***,

PriceTable.***ALMOND***);

LunchMenu[] child = {c1, c2};

**for**(LunchMenu c:child)

System.***out***.println(c.getStr() + c.calculate());

}

}

<오늘의 총 실습 예제2>

자동차 옵션에 따라 변경되는 세금 계산 프로그래밍 구현

※시나리오 부석 후 상속을 통해 자동차 객체를 만들어본다.

자동차의 경우 동일한 자동차 모델에서 옵션사항에 따라 세금차이가 있습니다. 그 중 배기량이 높은 자동차의 경우 세금이 높습니다.

Car : color, tire, displacement, handle, getSpec()

LowGradeCar : tax, color(블루), tire(일반타이어), displacement(2000), handle(파워핸들), getSpec()

HighGradeCar : tax, color(레드), tire(광폭타이어), displacement(2200), handle(파워핸들), getSpec()

☞ getSpec()을 통해서 자동차의 옵션을 보고, 세금을 파악합니다

**public** **abstract** **class** Car {

**private** String color;

**private** String tire;

**private** **int** displacement;

**private** String handle;

**public** Car(String color, String tire, **int** displacement, String handle) {

**this**.color = color;

**this**.tire = tire;

**this**.displacement = displacement;

**this**.handle = handle;

}

**public** **abstract** **void** getSpec();

**setter & getter**

}

**public** **class** LowGradeCar **extends** Car {

**private** **int** tax = 50000;

**public** LowGradeCar(String color, String tire, **int** displacement, String handle) {

**super**(color, tire, displacement, handle);

}

@Override

**public** **void** getSpec() {

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.***out***.println("색 상 : "+getColor());

System.***out***.println("타이어 : "+getTire());

System.***out***.println("배기량 : "+getDisplacement());

System.***out***.println("핸 들 : "+getHandle());

**if**(getDisplacement()>1600) tax += 15000;

System.***out***.println("세 금 : " +tax);

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

}

**public** **class** HighGradeCar **extends** Car {

**private** **int** tax = 100000;

**public** HighGradeCar(String color, String tire, **int** displacement, String handle) {

**super**(color, tire, displacement, handle);

}

@Override

**public** **void** getSpec() {

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.***out***.println("색 상 : "+getColor());

System.***out***.println("타이어 : "+getTire());

System.***out***.println("배기량 : "+getDisplacement());

System.***out***.println("핸 들 : "+getHandle());

**if**(getDisplacement()>2500) tax += 30000;

System.***out***.println("세 금 : " +tax);

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

}

**public** **class** CarSpecs {

**public** **static** **final** String ***COLOR\_RED***="레드";

**public** **static** **final** String ***COLOR\_BLUE*** = "블루";

**public** **static** **final** String ***COLOR\_GRAY***= "회색";

**public** **static** **final** String ***TIRE\_NORMAL*** = "일반타이어";

**public** **static** **final** String ***TIRE\_WIDE*** = "일반타이어";

**public** **static** **final** **int** ***DISPLACEMENT\_2000*** = 2000;

**public** **static** **final** **int** ***DISPLACEMENT\_2200*** = 2200;

**public** **static** **final** String ***HANDLE\_POWER*** = "파워핸들";

}

**public** **class** MainTest {

**public** **static** **void** main(String[] args) {

Car lCar = **new** LowGradeCar(CarSpecs.***COLOR\_BLUE***, CarSpecs.***TIRE\_NORMAL***, CarSpecs.***DISPLACEMENT\_2000***, CarSpecs.***HANDLE\_POWER***);

Car hCar = **new** HighGradeCar(CarSpecs.***COLOR\_RED***, CarSpecs.***TIRE\_WIDE***, CarSpecs.***DISPLACEMENT\_2200***, CarSpecs.***HANDLE\_POWER***);

lCar.getSpec();

hCar.getSpec();

}

}

5. final(제한자)

※ 추상클래스란 "추상적으로 정의할 테니(선언해 놓을테니), 사용자가 꼭 메소드를 재정의(overriding) 하세요"란 의미입니다.

그와는 정반대되는 개념이 있어 소개합니다.

⑴ 클래스 앞에 붙일 경우 : 상속 금지

public final class Test {

}

⑵ 멤버 메소드 앞에 붙일 경우 : 오버라이딩 금지

public final void print(){

}

⑶ 멤버변수 앞에 붙일 경우 : 상수화된다(변경금지).

public final int PORT\_NUMBER = 80;

<예제1>

public class Parent {

public final int PORT\_NUMBER = 80;

public void parent1(){

System.out.println("난 일반 메소드 parent1 실행했다");

System.out.println("PORT\_NUMBER : "+PORT\_NUMBER);

}

public final void parent2(){

System.out.println("난 final 메쏘드(재정의 금지)");

}

}

public class Child extends Parent {

public void parent1() {

//super.parent1();

System.out.println("parent1() 재정의");

}

/\*public void parent2() {

//final 메쏘드는 오버라이드 될 수 없다

}\*/

}

<예제2> 정규직 3명과 아르바이트생2명인 조그만 계열사의 월급명세서 출력 프로그램을 구현하시오.

Employee : name, computePay(),computeIncentive()

SalaryEmployee : name, annalSalary, computePay(),computeIncentive()

HourlyEmployee : name, hoursWorked, moneyPerHour, computePay(),computeIncentive()

**public** **class** MainClass {

**public** **static** **void** main(String[] args) {

Employee[] sawon = {

**new** SalaryEmployee("홍길동", 24000000),

**new** SalaryEmployee("강동원", 70000000),

**new** SalaryEmployee("김고은", 12000000),

**new** HourlyEmployee("홍길구", 100,7000),

**new** HourlyEmployee("김길자", 120,8000)};

**for**(Employee temp : sawon){

System.***out***.println("~ ~ ~ 월급명세서 ~ ~ ~");

System.***out***.println("성 함 :"+temp.getName());

System.***out***.println("월 급 : "+temp.computePay());

//System.out.println("상 여 : "+ temp.computeIncentive());

**int** tempIncen = temp.computeIncentive();

**if**(tempIncen!=0){

System.***out***.println("상 여 :"+ tempIncen);

}

System.***out***.println(" 수 고 하 셨 습 니 다 ");

}

}

}