**[ 10 ] 추상클래스**

목표: 추상클래스의 이해와 필요 이유, 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) 강요!

ex. 한가지 간단한 상속을 통한 예제를 살펴보도록 하겠습니다.

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

본사 : 김치찌개-0 부대찌개-0 비빔밥-0 순대국-0 공기밥-0

주택가에 매장1호점: 김치찌개-8,000 부대찌개-8,000 비빔밥-9,000 순대국-안팔아 공기밥-1,000원

대학가에 매장2호점: 김치찌개-8,000 부대찌개-8,000 비빔밥-8,000 순대국-8,000 공기밥-무료

증권가에 매장3호점: 김치찌개-9,000 부대찌개-10,000 비빔밥-10,000 얼큰순대국-9,000 공기밥-1,000원

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

**private** String storeName;

**public** HeadQuarterStore(String storeName) {**this**.storeName = storeName;}

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

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

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

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

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

**public** String getStoreName() {**return** storeName;}

}

**public** **class** Store1 **extends** HeadQuarterStore {

**public** Store1(String storeName) {**super**(storeName);}

@Override

**public** **void** bude() {System.***out***.println("부재찌개 8,000원");}

@Override

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

}

**public** **class** Store **extends** HeadQuarterStore {

**public** Store(String storeName) { **super**(storeName); }

@Override

**public** **void** bude() {System.***out***.println("부재찌개 8,000원");}

@Override

**public** **void** bibim() {System.***out***.println("비빔밥 8,000원");}

@Override

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

}

**public** **class** Store3 **extends** HeadQuarterStore {

**public** Store3(String storeName) { **super**(storeName); }

@Override

**public** **void** kimchi() {System.***out***.println("김치찌개 9,000원");}

@Override

**public** **void** bude() {System.***out***.println("부대찌개 10,000원");}

@Override

**public** **void** bibim() {System.***out***.println("비빔밥 10,000원");}

@Override

**public** **void** sunde() {System.***out***.println("얼큰순대국 9,000원");}

}

**public** **class** Main2 {

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

HeadQuarterStore[] store = {**new** HeadQuarterStore("=== 본사 ==="),

**new** StoreNum1("=== 1호점 ==="),

**new** StoreNum2("=== 2호점 ==="),

**new** StoreNum3("=== 3호점 ===") };

**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 {

**private** String storeName;

**public** HeadQuarterStore(String storeName) {**this**.storeName = storeName;}

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

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

**public** **abstract** **void** bibim();

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

**public** **abstract** **void** gongibab();

**public** String getStoreName() {**return** storeName;}

}

**public** **class** Store1 **extends** HeadQuarterStore {

**public** Store1(String storeName) {**super**(storeName);}

@Override

**public** **void** bude() {System.***out***.println("부재찌개 8,000원");}

@Override

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

@Override

**public** **void** kimchi() {System.***out***.println("김치찌개 8,000원");}

@Override

**public** **void** bibim() {System.***out***.println("비빔밥 9,000원");}

@Override

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

}

**public** **class** Store2 **extends** HeadQuarterStore {

**public** Store2(String storeName) { **super**(storeName); }

@Override

**public** **void** bude() {System.***out***.println("부재찌개 8,000원");}

@Override

**public** **void** bibim() {System.***out***.println("비빔밥 8,000원");}

@Override

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

@Override

**public** **void** kimchi() {System.***out***.println("김치찌개 8,000원");}

@Override

**public** **void** sunde() {System.***out***.println("순대국 8,000원");}

}

**public** **class** Store3 **extends** HeadQuarterStore {

**public** Store3(String storeName) { **super**(storeName); }

@Override

**public** **void** kimchi() {System.***out***.println("김치찌개 9,000원");}

@Override

**public** **void** bude() {System.***out***.println("부대찌개 10,000원");}

@Override

**public** **void** bibim() {System.***out***.println("비빔밥 10,000원");}

@Override

**public** **void** sunde() {System.***out***.println("얼큰순대국 9,000원");}

@Override

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

}

**public** **class** StoreMain {

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

HeadQuarterStore[] stores = {//new HeadQuarterStore("= = 본사 = ="),

**new** Store1("= = 주택가 1호점 = ="),

**new** Store2("= = 대학가 2호점 = ="),

**new** Store3("= = 증권가 3호점 = =")};

**for**(**int** idx=0 ; idx<stores.length ; idx++) {

System.***out***.println(stores[idx].getStoreName());

stores[idx].kimchi();

stores[idx].bude();

stores[idx].bibim();

stores[idx].sunde();

stores[idx].gongibab();

}//일반 for

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

System.***out***.println(store.getStoreName());

store.kimchi();

store.bude();

store.bibim();

store.sunde();

store.gongibab();

}// 확장 for

}

}

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. 추상메소드는 정의만 하고 구현은 하지 않습니다.

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** **class** ChildClass **extends** SuperClass {

@Override

**public** **void** method1() {

System.***out***.println("ChildClass의 method1()");

}

}

**public** **class** TestMain {

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

ChildClass childClass = **new** ChildClass();

childClass.method1();

childClass.method2();

}

}

<예제1-1>

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

System.***out***.println("도형을 그려요");

}

}

**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***.print("원");

**super**.draw();

}

}

**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***.print("삼각형");

**super**.draw();

}

}

**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***.print("사각형");

**super**.draw();

}

}

**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();

Shape[] shape = { **new** Circle(5),

**new** Triangle(10, 5),

**new** Rectangle(5, 10)

};// 배열

**for**(Shape s : shape) {

s.computeArea();

s.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** MainClass {

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

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

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

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

circle.draw();

System.***out***.println("삼각형넓이:"+

triangle.computeArea());

triangle.draw();

}

}

3. 추상클래스가 필요한 이유

※강제성을 느낄 때 사용 합니다

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

4. 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;

<quiz> 정규직 3명과 아르바이트생2명인 조그만 계열사의 월급명세서 출력 프로그램을 구현하여 [yisy0703@naver.com](mailto:yisy0703@naver.com) 에 파일첨부하여 제출하시오.

Employee : name, int computePay(),int computeIncentive(월급이 300이상인 사람만 월급의 5%를 incentive)

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

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

메인함수 :

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

Employee[] sawon = {

**new** SalaryEmployee("홍길동", 48000000),

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

**new** SalaryEmployee("김고은", 26000000),

**new** HourlyEmployee("홍길구", 100,9700),

**new** HourlyEmployee("김길자", 160,10000)};

**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);

}

}

}

실행결과 :

~ ~ ~ 월급명세서 ~ ~ ~

성 함 :홍길동

월 급 : 1714285

~ ~ ~ 월급명세서 ~ ~ ~

성 함 :강동원

월 급 : 5000000

상 여 :500000

~ ~ ~ 월급명세서 ~ ~ ~

성 함 :김고은

월 급 : 857142

~ ~ ~ 월급명세서 ~ ~ ~

성 함 :홍길구

월 급 : 700000

~ ~ ~ 월급명세서 ~ ~ ~

성 함 :김길자

월 급 : 960000

main() 함수에서 : new SalaryEmployee(“홍길동”, 2800) / new HourlyEmployee(“신길동”, 100, 8500)