**[ 11 ] 상속 II**

1. 오버라이드(override : 재정의)의 이해

(cf)생성자 오버로딩(overloading=polymorphism 중복정의) : 인자의 타입이 다르면 같은 이름의 메소드라도 다른 기능으로 중복정의가 가능한 것

(cf)오버라이드(override=재정의) : 부모 클래스의 메소드를 자식 클래스에서 재정의 함. 자식 클래스가 부모 클래스를 상속하여 자식한테 없는 메소드를 호출하면 부모클래스에 가서 해당 메소드를 찾게 됩니다. 만약 부모 클래스의 메소드를 자식클래스에서 동일한 이름으로 다시 재정의 하면 부모클래의 메소드를 찾지 않고 자식 클래스의 메소드를 호출하게 됩니다. 이것을 오버라이드(재정의)라고 합니다.

**package** com.ch.ex4override;

**public** **class** ParentClass {

**public** ParentClass() { }

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

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

}

**package** com.ch.ex4override;

**public** **class** ChildClass **extends** ParentClass {

**public** ChildClass() { }

@Override

**public** **void** method1() {System.***out***.println("ChildClass의 메소드1");}

**public** **void** method3() {System.***out***.println("ChildClass의 메소드3");}

}

**package** com.ch.ex4override;

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

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

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

c.method1(); //오버라이드(override)한 method1

c.method2();

c.method3();

}

}

/\*위의 경우 c.method1()을 호출하면 ChildClass의 method1()가 호출되며, ParentClass의 method1() 메소드는 호출되지 않는다\*/

**package** com.ch.ex5override;

**public** **class** ParentClass {

**private** String pStr = "부모클래스";

**public** ParentClass() { System.***out***.println("부모 생성자"); }

**public** **void** getPapaName(){ System.***out***.println("아빠 홍길동"); }

**public** **void** getMamiName(){ System.***out***.println("엄마 김길자"); }

**public** String getpStr() { **return** pStr; }

**public** **void** setpStr(String pStr) { **this**.pStr = pStr;}

}

**package** com.ch.ex5override;

**public** **class** ChildClass **extends** ParentClass {

**private** String cStr = "아들 클래스";

**public** ChildClass() { System.***out***.println("자식 생성자"); }

@Override

**public** **void** getMamiName() {

System.***out***.print("아주 이쁜 정말 이쁜 우리 ");

**super**.getMamiName();

}

**public** String getcStr() { **return** cStr; }

**public** **void** setcStr(String cStr) { **this**.cStr = cStr; }

}

**package** com.ch.ex5override;

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

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

ChildClass cc = **new** ChildClass();

System.***out***.println(cc.getpStr());

System.***out***.println(cc.getcStr());

cc.getPapaName();

cc.getMamiName();

ParentClass pp = **new** ParentClass();

pp.getMamiName();

}

}

2. 생성자와 관련된 상속 예제

⑴ 생성자함수 실행 순서도 위의 예제에서 추가하여 확인하자

**package** com.ch.ex3.copy;

**class** Parent {

**private** **int** p=0;

**public** Parent() { System.***out***.println("Parent형 객체 생성했지요"); }

**public** **int** getP() { **return** p; }

**public** **void** setP(**int** p) { **this**.p = p; }

}

**class** Child1 **extends** Parent {

**private** **int** c1=1;

**public** Child1() { System.***out***.println("Child1형 객체 생성했지요"); }

**public** **int** getC1() { **return** c1; }

**public** **void** setC1(**int** c1) { **this**.c1 = c1; }

}

**class** Child2 **extends** Parent {

**private** **int** c2=2;

**public** Child2() { System.***out***.println("Child2형 객체 생성했지요"); }

**public** **int** getC2() { **return** c2; }

**public** **void** setC2(**int** c2) { **this**.c2 = c2; }

}

**class** GrandChild **extends** Child1{

**private** **int** g=10;

**public** GrandChild() { System.***out***.println("GrandChild형 객체 생성했지요"); }

**public** **int** getG() { **return** g; }

**public** **void** setG(**int** g) { **this**.g = g; }

}

**public** **class** GrandChildTestMain {

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

Parent parent = **new** Parent();

Child1 child1 = **new** Child1();

Child2 child2 = **new** Child2();

GrandChild grandchild = **new** GrandChild();

System.***out***.println("parent의 p : "+ parent.getP()); //가능

System.***out***.println("child1.p : "+child1.getP()); //가능

System.***out***.println(child1.getC1()); //가능

System.***out***.println(child2.getP()); //가능

//System.out.println(child2.getC1()); //불가능

System.***out***.println(child2.getC2()); //가능

System.***out***.println(grandchild.getP());

System.***out***.println(grandchild.getC1());

System.***out***.println(grandchild.getG());

//System.out.println(grandchild.getC2()); //불가능

}

}

⑵ 상속 관계에서의 생성자 문제와 해결책

1. 디폴트 생성자는 JVM이 제공해주지만, 클래스 내의 매 개변수가 있는 생성자가 하나라도 존재하게 되면 JVM 은 더 이상 디폴트 생성자를 제공해 주지 않게 된다.
2. 만일 수퍼 클래스에 매개 변수가 있는 생성자를 정의하면서 매개 변수 없는 디폴트 생성자를 정의하지 않으면 수퍼 클래스에는 매개 변수 없는 생성자가 존재하지 않게 된다.
3. 이러한 상태에서 서브 클래스의 생성자는 수퍼 클래스 의 매개 변수 없는 디폴트 생성자를 여전히 호출하고 있 기에 존재하지 않는 생성자를 호출하는 셈이 되어 문제 가 발생하게 된다.

**public** **class** Person {

**private** String name;

**private** String character;

**public** Person() { System.***out***.println("인자없는 생성자 호출됨"); }

**public** Person(String name, String character) {

System.***out***.println("인자 2개짜리 생성자 호출됨");

**this**.name = name;

**this**.character = character;

}

**public** **void** intro(){ System.***out***.println(name+"는 "+character); }

**public** String getName() { **return** name; }

**public** **void** setName(String name) { **this**.name = name; }

**public** String getCharacter() { **return** character; }

**public** **void** setCharacter(String character) { **this**.character = character; }

}

**public** **class** Baby **extends** Person {

**public** Baby() {System.***out***.println("인자없는 자식 클래스 생성자 호출됨"); }

**public** Baby(String name, String character) {

System.***out***.println("인자 2개짜리 자식 클래스 생성자 호출됨");

setName(name);

setCharacter(character);

}

**public** **void** cry(){System.***out***.println("응애응애");}

@Override

**public** **void** intro() {System.***out***.println(getName()+"은(는) 아기라서 말 못해요");}

}

**public** **class** FamilyMain {

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

Person papa = **new** Person();

papa.setName("아빠곰");

papa.setCharacter("뚱뚱해");;

papa.intro();

Person mom = **new** Person("엄마곰","날씬해");

mom.intro();

Baby child1 = **new** Baby();

child1.setName("아기곰1");

child1.setCharacter("귀여워");

child1.cry();

child1.intro();

Baby child2 = **new** Baby("아기곰2","너무 귀여워");

child2.cry();

child2.intro();

}

}

실행결과 :

인자없는 부모 생성자 호출됨

아빠곰는 뚱뚱해

인자 2개짜리 생성자 호출됨

엄마곰는 날씬해

인자없는 부모 생성자 호출됨

인자없는 자식 클래스 생성자 호출됨

응애응애

아기곰1은(는) 아기라서 말 못해요

인자없는 부모 생성자 호출됨

인자 2개짜리 자식 클래스 생성자 호출됨

응애응애

아기곰2은(는) 아기라서 말 못해요

3. super키워드(오버라이드의 장점과 단점의 이해)

super란? super는 부모를 찾는 키워드

⑴ 하위 클래스에 의해 가려진 상위클래스의 멤버변수나 메소드에 접근할 때

super.객체변수

super.메소드이름(매개변수)

override의 장점은 잘 만들어진 클래스를 상속받아 일부 메소드를 수정(재정의) 할 수 있다는 것 입니다.

하지만, 자식클래스에서 부모클래스를 override하면 부모클래스의 메소드는 사용할 수가 없습니다.

이것이 override의 단점이라고 할 수 있습니다.

자식 클래스에서 부모클래스의 메소드를 사용하고자 할 때 super키워드를 사용 합니다.( cf. this )

**package** com.ch.ex9super;

**public** **class** ParentClass {

**protected int** i = 1;

**public** ParentClass() { System.***out***.println("ParentClass생성자"); }

**public** **void** method(){ System.***out***.println("부모 클래스의 method입니다"); }

}

**package** com.ch.ex9super;

**public** **class** ChildClass **extends** ParentClass {

**private int** i = 10

**public** ChildClass() { System.***out***.println("ChildClass생성자"); }

**public** **void** method(){

**super**.method();

System.***out***.println("부모클래스의 method()를 실행하고 자식클래스의 method 나머지 작업을 합니다.");

System.***out***.println(i+"/"+ **super**.i);

}

}

**package** com.ch.ex9super;

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

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

ChildClass child = **new** ChildClass();

child.method();

}

}

⑵ 상위클래스의 생성자를 호출할 때

super(매개변수) ; 이 super(매개변수)문장은 반드시 **첫 번째 라인**에 와야 한다.

super() ; 매개변수 없는 슈퍼클래스의 생성자 호출

**public** **class** Person {

**private** String name;

**private** String tel;

**public** Person(String name, String tel) {

**this**.name = name;

**this**.tel = tel;

}

**public** String infoString() {**return** "[이름]"+name+" [전화]"+tel;}

**public** **void** print() {System.***out***.print("[이름]"+name+" [전화]"+tel);}

**public** String getName() {**return** name;}

**public** **void** setName(String name) {**this**.name = name;}

**public** String getTel() {**return** tel;}

**public** **void** setTel(String tel) {**this**.tel = tel;}

}

**public** **class** Customer **extends** Person{

**private** String add;

**private** **int** sum; // 누적구매금액

**private** **int** point; // 포인트

**private** String date; // 생일 : 0703

**private** **boolean** vip; // true(vip고객) false(일반고객)

**public** Customer(String name, String tel, String add) {

**super**(name, tel); // super() 함수 호출은 반드시 첫번째라인에

sum = 0;

point = 1000;

vip = **false**;

}

**public** **void** buy(**int** price) {

sum += price;

**int** tempPoint = (**int**)(price\*0.05);

point += tempPoint;

System.***out***.println(getName()+"님 구매 감사합니다. 금번 포인트 : "+tempPoint+", 총 포인트 :"+point);

**if**(sum>1000000 && vip==**false**) {

vip = **true**;

}//vip여부 if

}//buy

@Override

**public** String infoString() {

**return** **super**.infoString() + " [포인트]"+point;

}

@Override

**public** **void** print() {

**super**.print();

System.***out***.println(" [포인트]"+point);

}

}

**public** **class** Staff **extends** Person {

**private** String hiredate; // 입사일

**private** String department; // 부서

**public** Staff(String name, String tel, String hiredate, String department) {

**super**(name, tel); // super() 함수 호출은 반드시 첫번째라인에

**this**.hiredate = hiredate;

**this**.department = department;

}

@Override

**public** String infoString() {

**return** **super**.infoString() + " [부서]"+department;

}

@Override

**public** **void** print() {

**super**.print();

System.***out***.println(" [부서]"+department);

}

}

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

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

Customer customer = **new** Customer("홍고객","010-9999-9999","종로구 수표로 105 803호");

customer.buy(10000);

Staff staff = **new** Staff("홍직원", "010-8888-8888", "19-01-01", "전산실");

Person[] person = {customer, staff};

**for**(Person p : person) {

System.***out***.println(p.infoString());

p.print();

}

**if**(person[0] **instanceof** Customer)

((Customer)person[0]).buy(1000);

}

}

4. 다음을 구현한 프로그램을 구현하여 보자

<예제> 체인점 사업을 통한 상속의 예

※본사의 메뉴 가격을 기본적으로 따르되, 지역 특색에 맞춰 가격을 변경 함.

본사 지침 : 김치찌개-5,000 부대찌개-6,000 비빔밥-6,000 순대국-5,000 공기밥-1,000원

주택가에 매장1호점: 김치찌개-5,000 부대찌개-**5,000** 비빔밥-6,000 순대국-**안팔아** 공기밥-1,000원

대학가에 매장2호점: 김치찌개-5,000 부대찌개-**5,000** 비빔밥-**5,000** 순대국-5,000 공기밥-**무료**

증권가에 매장3호점: 김치찌개-**6,000** 부대찌개-**7,000** 비빔밥-**7,000** 순대국-**6,000** 공기밥-1,000원

**package** com.ch.ex8store;

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

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

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

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

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

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

}

**package** com.ch.ex8store;

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

@Override

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

@Override

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

}

**package** com.ch.ex8store;

**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"); }

}

public class StoreNum3 extends HeadQuarterStore {

**package** com.ch.ex8store;

**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"); }

}

**package** com.ch.ex8store;

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

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

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

st.kimchi();

st.bude();

st.bibib();

st.sunde();

st.gonggibab();

System.***out***.println("========1호==========");

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

st1.kimchi();

st1.bude();st1.bibib();

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

System.***out***.println("========2호==========");

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

st2.kimchi();

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

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

System.***out***.println("========3호==========");

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

st3.kimchi();

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

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

}

}

Object 클래스

↑

S 클래스

↗ ↑ ↖ ☞ A클래스, B클래스, C클래스 모두 Object클래스로 타입이 같아질 수 있다

A클래스 B클래스 C클래스 ☞ A클래스, B클래스, C클래스 모두 S클래스로 타입이 같아질 수 있다.

반대는 안 됨

생물 일반화

↗ ↖ ↑

동물 식물

↗ ↗ ↖ ↖ ↓

조류 포유류 어류 양서류 특수화

※ Object클래스 ? 모든 클래스의 최고조상

- 조상이 없는 클래스는 자동적으로 Object클래스를 상속받게 된다.

- 상속계층도의 최상위에는 Object클래스가 위치한다.

- 모든 클래스는 Object클래스에 정의된 11개의 메서드를 상속받는다.

toString(), equals(Object obj), hashCode(), ...

[오늘의 총 실습예제 : 5개 파일을 압축하여 yisy0703@naver.com]

오늘의 내용을 종합하여 아카데미 프로그램을 구현해 보자.

클래스는 다음과 같다

1. Person – id, name / infoString(), getter와 setter
2. **Student** extends Person – ban / infoString() 재정의
3. **Gangsa** extends Person – subject / infoString() 재정의
4. **Staff** extends Person – department / infoString() 재정의
5. main함수 넣을 클래스

main(){

Person st1 = new Student(“A01”,”강유미”,”JAVA반”);

Person st2 = new Student(“A02”,”홍길동”,”C++반”);

Person sf1 = new Staff(“S01”,”유길동”,”운영지원팀”);

Person sf2 = new Staff(“S02”,”유만동”,”취업지원팀”);

Person ga1 = new Gangsa(“G01”,”이길동”,”객체지향”);

Person[] p = {st1, st2, sf1, sf2, ga1};

for(Person pp : p)

sysout(pp.infoString());

}

결과) (ID)A01 (이름):강유미 (반)JAVA반

(ID)A02 (이름):홍길동 (반)C++반

(ID)S01 (이름):유길동 (부서)운영지원팀

(ID)S01 (이름):김길동 (부서)취업지원팀

(ID)G01 (이름):이길동 (과목)객체지향

* 상속 시 final

- final class 클래스명 {} 상속불가

- final void 메서드명(){} 오버라이딩 금지

**class** Animal{

**protected** **int** speed;

**public** **void** running(){

speed += 5;

System.***out***.println("뛰고 있어요 현속도 : "+speed);

}

**public** **final** **void** stop(){

speed = 0;

System.***out***.println("멈춤!");

}

}

**final** **class** Dog **extends** Animal{

@Override

**public** **void** running() {

speed += 10;

System.***out***.println("뛰면서 꼬리를 흔들어요 현속도 : "+speed);

}

}

**class** Rabbit **extends** Animal{

@Override

**public** **void** running() {

speed += 30;

System.***out***.println("겁나서 마구 뛰어요 현속도 : "+speed);

}

}

// class JinDo extends Dog{ }

**public** **class** A05\_final {

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

Dog d = **new** Dog();

d.running();

d.stop();

Rabbit r = **new** Rabbit();

r.running();

r.stop();

}

}