**[ 9 ] 상속**

목표 : 상속의 의미하는 바와 실제 코드를 작성할 수 있다.

오버로딩(overloading)과 오버라이딩(overriding)의 차이를 알고 구분해서 구현할 수 있다

(cf) 상속(Inheritance) : 상속이란 부모클래스가 가지고 있는 속성과 메서드를 활용할 수 있는 개념

1. 일상에서 알고 있는 상속과 비슷한 개념

※ 객체지향의 핵심적인 요소는 상속과 추상화.

프로그래밍에서도 어떤 클래스가 다른 클래스로부터 **데이터(속성)와 메소드(기능)를 상속받을 수 있습니다. 이럴 경우 상속받은 객체는 상속해 준 객체의 데이터 및 메소드를 이용할 수 있고 또는 변경할 수도 있습니다.**

상속을 통해서 많은 개발시간을 단축시킬 수 있습니다. 또 기존에 만들어진 클래스는 검증이 된 클래스이므로, 시행착오도 줄일 수 있습니다.

2. 상속 문법의 이해

|  |
| --- |
| 접근제어자 [final/abstract] class 클래스이름 **extends** 상위클래스(super class) {  추가할 멤버변수선언;  생성자;  추가할 메소드선언;  } |

⑴ 상속이란 기존의 클래스를 재사용해서 새로운 클래스를 작성하는 것

⑵ 두 클래스를 부모와 자식(조상과 자손)으로 관계를 맺어줄 수 있다.

⑶ 자손은 조상의 멤버를 상속받으나, private멤버는 직접 제어할 없다.

자손의 멤버 개수는 조상보다 작을 수 없다(같거나 많다)

⑷ 자바에서는 다중 상속이 지원되지 않는다.

3. 상속이 필요한 이유

상속이 필요한 이유는 크게 두 가지 이유입니다.

첫째로는, 모든 개발을 처음부터 하얀 도화지에 할 필요는 없습니다. 이미 훌륭한 선배님들께서 만들어 놓은 프로그램이 있다면 상속을 통해서 훌륭한 결과물을 빠른 시간 내에 만들 수 있습니다. 또한 기존의 훌륭한 프로그램은 대부분 검증이 잘 되어 있어 버그도 거의 없을 수 있습니다.

**(ex01)**

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

String pStr = "부모클래스";

**public** ParentClass() {System.***out***.println("부모클래스 객체 부분 생성"); }

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

**public** **void** getMamiName() {System.***out***.println("엄마이름 : 김길순");}

}

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

String cStr = "아들클래스";

**public** ChildClass() {System.***out***.println("자식클래스 객체 부분 생성");}

}

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

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

child.getPapaName();

child.getMamiName();

System.***out***.println(child.pStr);

System.***out***.println(child.cStr);

}

**(ex02)**

**public** **class** SuperIJ {

**private** **int** i;

**private** **int** j;

**public** **void** setIJ(){}

**public** **void** setIJ(**int** i, **int** j) {

**this**.i = i;

**this**.j = j;

}

**protected** **void** setI(**int** i) {**this**.i = i;}

**public** **void** setJ(**int** j) {**this**.j = j;}

**public int** getI() {**return** i; }

**protected** **int** getJ() {**return** j; }

}

**public** **class** Child **extends** SuperIJ{

**private** **int** total;

**public** **void** sum() {

total = getI() + getJ();

System.***out***.printf("본 객체의 i=%d, j=%d. ", getI(), getJ());

System.***out***.println("본 객체의 total값은 "+total);

}

}

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

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

Child child = **new** Child();

System.***out***.println("child의 i : "+child.getI());

System.***out***.println("child의 j : "+child.getJ());

child.setI(10);child.setJ(20);

child.sum();

}

}

둘째로는, **다양한 객체(타입)를 상속을 통해서 하나의 객체(타입)으로 묶을 수 있습니다**.

이것은 추상화 개념의 이해가 있어야 하는데요, 아직은 추상화에 대해서 살펴보지 않았으므로 우선은 다양한 객체를 하나로 통일시킬 수 있다고만 알고 있으면 될 거 같습니다.

생물 = 호흡해

동물 = 호흡하고 움직여

식물 = 호흡하고 안움직여 ☞동물과 식물을 생물로 통일시킬 수 있는 개념.

생물 일반화(추상화)

↗ ↖ ↑

동물 식물

↗ ↗ ↖ ↖ ↓

조류 포유류 어류 양서류 특수화

**(ex03)**

**public** **class** S {

**public** **int** s=0;

}

**package** com.ch.ex2;

**public** **class** A **extends** S{

**public** A() {

System.***out***.println("A 클래스 생성자");

s = 1;

}

}

**package** com.ch.ex2;

**public** **class** B **extends** S {

**public** B() {

System.***out***.println("B 클래스 생성자");

s = 2;

}

}

**package** com.ch.ex2;

**public** **class** C **extends** S {

**public** C() {

System.***out***.println("C 클래스 생성자");

s = 3;

}

}

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

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

// A a = new A();

S a = **new** A(); // 타입을 통일화. 이것이 다형성

// B b = new B();

S b = **new** B();

// C c = new C();

S c = **new** C();

//A s = new S(); 거꾸로는 불가능

S[] objArr = {a, b, c};

**for**(S obj : objArr) {

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

}

}

}

4. 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(), ...

5. 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.

**(ex04)**

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

**public** ParentClass() { // overloading = 중복정의

System.***out***.println("매개변수 없는 ParentClass 생성자");

}

**public** ParentClass(**int** i) {

System.***out***.println("매개변수 있는 ParentClass 생성자");

}

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

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

}

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

**public** ChildClass() { // 오버로딩(함수의 중복정의)

System.***out***.println("매개변수 없는 ChildClass 생성자");

}

**public** ChildClass(**int** i) {

System.***out***.println("매개변수 있는 ChildClass 생성자");

}

@Override

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

**public** **void** method3() {System.***out***.println("ChildClass의 emthod3()");}

@Override

**public** **boolean** equals(Object obj) {**return** **true**;}

}

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

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

ParentClass parentObj1 = **new** ParentClass();

ParentClass parentObj2 = **new** ParentClass(2);

parentObj1.method1();

parentObj1.method2();

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

ChildClass childObj1 = **new** ChildClass();

ChildClass childObj2 = **new** ChildClass(3);

childObj1.method1(); // 오버라이드된 method1() 호출

childObj1.method2();

childObj1.method3();

**int** i=10, j=11;

**if**(i==j) System.***out***.println("둘이 똑같아");

**if**(childObj1.equals(childObj2)) {

System.***out***.println("둘이 똑같아");

}

}

}

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

3. 생성자와 관련된 상속 예제

⑴ 생성자함수 실행 순서도 위의 예제에서 추가하여 확인하자

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.

**(ex05)**

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

**private** String name;

**private** String character;

**public** Person() {

System.***out***.println("매개변수=파라미터=인자가 없는 Person 생성자");

}

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

**this**.name = name;

**this**.character = character;

System.***out***.println("매개변수 2개짜리 Person 생성자");

}

**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("매개변수 없는 Baby");

}

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

**super**(name, character);

// super() ; 내 클래스의 슈퍼단의 생성자 함수

// setName(name);

// setCharacter(character);

// System.out.println("매개변수 2개 Baby");

}

@Override

**public** **void** intro() {

System.***out***.println("＠ 응애 응애 난 아기야");

**super**.intro();

}

}

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

**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.intro();

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

Baby child2 = **new** Baby("아기곰2","너무너무귀여워");

child2.intro();

}

}

6. 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 )

⑵ 상위클래스의 생성자를 호출할 때

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

super() ; 매개변수 없는 슈퍼클래스의 생성자 호출

quiz 1)

com.lec.ex2\_protected 패키지를 com.lec.ex2\_protected.copysuper 패키지로 복사한 후, Child클래스에서 super키워드를 이용해 보자

**(ex06)**

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

**private** String name;

**private** String tel; // Person p = new Person("홍길동","9999-9999")

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

**this**.name = name;

**this**.tel = tel;

}

**public** String infoString() {// sysout(p.infoString())

**return** "[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;}

}

**import** com.lec.constant.Constant;

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

**private** String address;

**private** **int** sum; //구매누적금액

**private** **int** point; // 포인트(구매할 때마다 5%적립)

**private** String date; // 기념일

**private** **boolean** vip; // true면 vip 고객, false면 일반고객

// Customer c = new Customer("홍","9999-9999","서울","0811");

**public** Customer(String name, String tel, String address, String date) {

**super**(name, tel);

**this**.address = address;

**this**.date = date;

sum = 0; // 새로운 고객은 구매누적금액 0원으로

point = 1000; // 회원가입한 첫고객은 1000점 포인트 자동 부여

vip = **false**; // 새로운 고객은 일반고객으로 자동 분류

System.***out***.println(name+"님 감사감사 어서오세요 포인트 1000점을 선물로");

}

**public** **void** buy(**int** price) {// c.buy(10000);

sum += price;

**int** tempPoint = (**int**)(price\*Constant.***RATE***);

point += tempPoint;

System.***out***.println(getName() +"님 감사합니다. \n금번 포인트 : " + tempPoint);

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

vip = **true**; //구매누적금액이 100만원 초과시 vip고객으로

System.***out***.println(getName()+"님 VIP 고객으로 업그레이드 되셨습니다. 자주 뵙겠습니다");

}// if

}//buy

@Override

**public** String infoString() {

**return** **super**.infoString()+" [주소]"+address +" [포인트]"+point +" [구매누적]"+sum;

}

}

**public** **class** Constant {

**public** **static** **final** **double** ***RATE*** = 0.05;

}

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

**private** String hiredate; // // 입사일(2022-03-21) 다음주부터는 Date형

**private** String department; // 부서

// Staff s = new Staff("홍길동","9999-9999","2020-05-01","전산실");

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

**super**(name, tel);

**this**.hiredate = hiredate;

**this**.department = department;

}

// sysout(s.infoString()) -> [이름]홍길동 [전화]9999-9999 [부서]전산실

@Override

**public** String infoString() {

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

}

}

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

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

Customer customer = **new** Customer("홍길동", "010-9999-9999", "서울", "03-21");

customer.buy(100000);

customer.buy(900000);

//System.out.println(customer.infoString());

Staff staff = **new** Staff("홍일꾼", "010-8888-8888", "2022-12-01", "전산실");

Person[] person = {customer, staff};

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

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

}

Person customer1 = **new** Staff("홍", "99", "19-12-1", "전산실");

//new Customer("김길순","7777-7777", "서울시 마포구","11-04");

System.***out***.println(customer1.infoString());

**if**(customer1 **instanceof** Customer) {

((Customer)customer1).buy(10000);

}**else** {

System.***out***.println("buy 할수 없는 타입");

}

}

}

7. 다음을 구현한 프로그램을 구현하여 보자

**(ex07)** 체인점 사업을 통한 상속의 예

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

본사 지침 : 김치찌개-8,000 부대찌개-9,000 비빔밥-9,000 순대국-8,000 공기밥-1,000원

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 8,000원");}

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

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

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

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

**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** Main {

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

// HeadQuarterStore head = new HeadQuarterStore("= = 본사 = =");

// Store1 st1 = new Store1("= = 주택가 1호점 = =");

// Store2 st2 = new Store2("= = 대학가 2호점 = =");

// Store3 st3 = new Store3("= = 증권가 3호점 = =");

// HeadQuarterStore[] stores = {head, st1, st2, st3};

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

}

}

* 상속 시 final

- final class 클래스명 {} 상속불가

- final void 메서드명(){} 오버라이딩 금지

**(ex08)**

**public 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!");

}

}

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

@Override

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

speed += 10;

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

}

}

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

@Override

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

speed += 30;

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

}

}

// class JinDo extends Dog{ }

**public** **class** Ex\_final {

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

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

d.running();

d.stop();

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

r.running();

r.stop();

**final** **int** CONSTANT = 10;

//CONSTANT=11;

}

}

**package** com.lec.cons;

**public** **class** Constant {

**public** **static** **final** **double** ***RATE*** = 0.05;

}

* 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();

}

}

<quiz>

오늘의 내용을 종합하여 아카데미 프로그램을 구현해 보자.

클래스는 다음과 같다

1. Person – no, id, name / print() 또는 infoString(), getter와 setter
2. **Student** extends Person – ban / print() 또는 infoString() 재정의
3. **Gangsa** extends Person – subject / print() 또는 infoString() 재정의
4. **Staff** extends Person – department / print() 또는 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(“S01”,”유길동”,”취업지원팀”);

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

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

for(Person pp : p)

pp.print();

// System.out.println(pp.infoString());

}

결과) (번호)student1 (ID)A01 (이름):강유미 (반)JAVA반

(번호)student2 (ID)A02 (이름):홍길동 (반)C++반

(번호)staff1 (ID)S01 (이름):유길동 (과목)운영지원팀

(번호)staff2 (ID)S01 (이름):김길동 (과목)취업지원팀

(번호) lecturer1 (ID)G01 (이름):이길동 (부서)객체지향