**[ 11 ] 상속 I**

목표 : 상속의 의미하는 바와 실제 코드를 작성할 수 있다.

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

다형성(polymorphism)의 개념을 이해하고, 활용할 수 있다.

(cf) 상속(Inheritance) : 상속이란 일반적으로 재산을 부모가 물려주는 개념이 있듯이, 자바에서는 부모가 가지고 있는 클래스의 속성과 메서드를 활용할 수 있는 개념으로 시작한다

1. 일상에서 알고 있는 상속

※ 객체지향의 핵심적인 요소 중의 하나.

객체지향 언어의 대표인 JAVA를 공부할 때 가장 중요하고 꼭 알고 넘어가야 하는 부분이 **상속과 추상화** 개념입니다.

상속은 우리가 일상에서 알고 있는 상속입니다. 아버지의 재산을 물려받는 아들의 경우 아버지로부터 아들은 재산을 상속받은 경우입니다.

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.

**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);

}

**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***.println("본 객체의 total 값은 "+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);

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

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

child.sum();

}

}

**public** **class** Point {

**private** **int** x;

**private** **int** y;

**public** **void** pointPrint() {

System.***out***.println("2차원 좌표 : "+x +", "+y);

}

**public** String pointInfoString() {

**return** "2차원 좌표 : "+x+", "+y;

}

**public** **int** getX() {**return** x;}

**public** **void** setX(**int** x) {**this**.x = x;}

**public** **int** getY() {**return** y;}

**public** **void** setY(**int** y) {**this**.y = y;}

}

**public** **class** Point3D **extends** Point{

**private** **int** z;

**public** **void** point3DPrint() {

System.***out***.println("3차원 좌표 : "+getX() +" , "+getY()+", "+z);

}

**public** String point3DInfoString() {

**return** "3차원 좌표 : "+getX()+" , "+getY()+", "+z;

}

**public** **int** getZ() {**return** z;}

**public** **void** setZ(**int** z) {**this**.z = z;}

}

**public** **class** PointMain {

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

Point point = **new** Point();

point.setX(5);point.setY(6);

System.***out***.println(point.pointInfoString());

point.pointPrint();

Point3D point3d = **new** Point3D();

point3d.setX(2); point3d.setY(3); point3d.setZ(4);

System.***out***.println(point3d.point3DInfoString());

point3d.point3Dprint();

System.***out***.println("2차원 좌표 : "+point.getX()+", "+point.getY());

System.***out***.println("3차원 좌표 : "+point3d.getX()+", "+point3d.getY()+", "+point3d.getZ());

}

}

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

**package** com.ch.ex2;

**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);

}

}

}