Day12(8/16)

\*디자인 패턴 – 싱글톤

**package** com.ict.am;

**class** MySingleCls{

//생성자가 private이기 때문에 class내부에 instance생성을 위한 참조변수 준비

**private** **static** MySingleCls *instance*;

//생성자의 접근 지정자는 private

**private** MySingleCls() {}

//객체 생성

**public** **static** MySingleCls getInstance(){

**if**(*instance* == **null**) {

*instance* = **new** MySingleCls();

}

**return** *instance*;

}

**public** **void** sayHello() {

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

}

}

**public** **class** Ex01 {

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

//생성자를 이용한 싱글톤 패턴

//싱글톤 - 해당 클래스의 객체는 오직 하나다.

//싱글톤 패턴의 핵심은 생성자의 접근 지정자를 private으로 한다.

//생성자가 private이면 class외부에서 객체를 만들 수 없다.

//내부에 static 필드로 객체를 가지고 있어야 한다.

//외부에서는 getInstance 메소드를 이용해서 사용해야 한다.

//생성자가 private이기 때문에 외부에서 객체생성 불가능

//MySingleCls obj = new MySingleCls();

MySingleCls instance = MySingleCls.*getInstance*();

instance.sayHello();

}

}

상속

1. 상속은 코드의 재활용을 보다 체계적으로 하기 위한 하나의 방법으로 나온 개념이다.

2. 자식클래스가 부모클래스의 기능을 상속받는다. (생성자는 상속되지 않는다.)

3. 부모 클래스: super class, 자식 클래스: sub class

>>새로 만드는 클래스를 이미 정의된 클래스를 상속받는다면 많은 기능을 물려받아 사용할 수 있다.

\*오버로딩: 클래스 안에서 같은 이름의 메소드를 가지고 있는 것

\*오버라이딩: 상속관계에서 부모가 가지고 있는 메소드를 재정의하여 사용

\*protected

-상속이 적용되는 클래스에서는 자식클래스가 부모 클래스의 필드에 접근할 수 있도록 하고,

클래스 외부에서는 접근 가능하지 못하도록 하기 위한 protected접근 지정자를 사용한다.

-자신의 클래스, 같은 패키지, 하위클래스o, 다른 패키지x

**package** com.ict.am;

**public** **class** Point2D {

**protected** **int** x;

**protected** **int** y;

}

**package** com.ict.bm;

**import** com.ict.am.Point2D;

//import하면 사용 가능

**public** **class** Point3D **extends** Point2D {

**public** **int** z;

**public** **void** setX(**int** x) {

**super**.x = x;

}

**public** **void** setY(**int** y) {

**super**.y = y;

}

**public** **int** getX() {

**return** **super**.x;

}

**public** **int** getY() {

**return** **super**.y;

}

}

//public : 모두 접근 가능

//protected: 접근 지정자

//상속이 적용되는 클래스에서는 자식클래스가 부모클래스의 필드에 접근 할 수 있도록 하고,

//클래스 외부에서는 접근가능하지 못하도록하기 위한 protected접근 지정자를 사용한다.

//default: 같은 package에서만 접근 가능

//private: 같은 클래스내에서만 사용 가능

**package** com.ict.c;

**import** com.ict.bm.Point3D;

**public** **class** Main{

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

Point3D pt = **new** Point3D();

//다른 패키지에 있기 때문에 사용 불가

//pt.x = 10;

pt.setX(30);

pt.setY(20);

pt.z = 15;

System.***out***.println("pt.x=>" +pt.getX());

System.***out***.println("pt.y=>" +pt.getY());

System.***out***.println("pt.z=>" +pt.z);

}

}

**package** com.ict.dm;

**class** Grandfather{

**void** play() {

System.***out***.println("할아버지의 나팔 뿡~뿡뿡~~");

}

}

**class** Father **extends** Grandfather{

//어노테이션을 이용해서 Override가 되도록 체크한다.

@Override

**void** play() {

System.***out***.println("아버지의 북 둥둥둥~");

}

}

//상속을 받으면 Father의 타입은 Grandfather의 타입이라고 해도 틀린 말이 아니다.

**class** Manager{

**void** action(Grandfather p) {

p.play();

}

}

**public** **class** Ex01 {

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

Manager manager = **new** Manager();

Grandfather h = **new** Grandfather();

Father f = **new** Father();

manager.action(h);

manager.action(f);

//up-casting : Grandfather를 상속받은 자식 클래스도 사용가능

f.play();

h.play();

}

}

**package** com.ict.em;

//class의 관계

//has a 관계: 특정 객체 내에서 다른 객체를 가지고 있는 것을 의미한다.

//is a 관계: 특정 객체가 다른 객체에게 자신의 능력을 포함시켜주는 상속관계

//Ex)circle(원)은 shape(도형)이다.

**class** Person {

**void** speaking() {

System.***out***.println("사람이 말한다.");

}

}

**class** Professor **extends** Person {

@Override

**void** speaking() {

System.***out***.println("교수가 말한다.");

}

**void** lesson() {

System.***out***.println("교수가 강의한다.");

}

}

**public** **class** Ex01 {

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

Professor kim = **new** Professor();

kim.speaking(); //교수가 말한다.

kim.lesson(); //교수가 강의한다.

Person p = kim; // Person타입이 교수 객체를 참조할 수 있다.

p.speaking();

//p.lesson(); //Person class 내에 정의된 메소드나 변수만 사용할 수 있다.

//즉 Professor에만 정의되어있는 변수나 메소드는 사용할 수 없다.

((Professor)p).lesson();//자손으로 강제 형변환(캐스팅)후에 자식 클래스의 메소드를 호출할 수 있다

}

}

-생성자는 상속되지 않는 유일한 멤버함수이다.

-부모클래스의 생성자가 먼저 호출된 후 자식클래스의 생성자가 호출된다.

**package** com.ict.em;

**class** Point{

**int** x;

**int** y;

**public** Point() {

System.***out***.println("Point의 기본생성자");

}

//생성자 오버로딩

**public** Point(**int** x, **int** y) {

**this**.x = x;

**this**.y = y;

System.***out***.println("Point클래스의 생성자 오버로딩");

}

}

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

**int** z;

**public** Point3D() {

//Point클래스의 디폴트 생성자를 추가 또는

//super(5,10); 를 이용해서 Point 클래스의 생성자를 직접 호출한다.

**super**(5,10); //생성자 호출은 첫번째 줄에서 해야한다.

System.***out***.println("Point3D의 디폴트 생성자!");

}

}

**public** **class** Ex02 {

//생성자는 상속되지 않는 유일한 멤버이다.

//서브 클래스의 인스턴스가 생성될 때 자신의 생성자가 호출되면서

//슈퍼클래스의 생성자가 연속적으로 자동으로 호출된다

//(이때, 자동으로 호출되는 생성자는 매개변수가 없는 기본생성자이다.)

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

Point3D pt = **new** Point3D();

//Point생성자->Point3D의 디폴트 생성자

//업캐스팅(암시적 형변환)

Point up = **new** Point3D();

//우선순위가 Point, 하지만 오버라이딩된 변수와 메소드는 Point3D를 이용해야 한다.

//child클래스에서 부모클래스의 메소드를 오버라이딩한 것이 있다면

//부모 클래스의 기능이 은폐된다.

//부모 메소드와 똑같은 것을 자식이 가지게 되면 자식의 기능이 확장되면서

//더 우선권을 가지기 때문이다.

//다운캐스팅(명시적인 형변환)

Point3D down = (Point3D)up;

}

}

\*final

final 변수: 상수

final 메소드: 더 이상 재정의가 불가능한 메소드

>>자기클래스가 부모클래스의 메소드를 그대로 사용가능 하지만 오버라이딩 할 수 없다.

final 클래스: 더 이상 상속이 불가능한 클래스

\*instanceof연산자

:레퍼런스 변수가 어떤 인스턴스를 참조하고 있는지를 살펴서 특정 클래스로 캐스트 연산 가능한지를 검사하는 연산자이다.