[ 15 ] 인터페이스 II

1. 인터페이스와 다중상속

Java는 다중상속이 불가 합니다. 예전의 C++에서는 다중상속이 가능하였으나, java에서는 다중상속이 불가능 합니다. 이유는 C++에서 다중상속을 가능하게 하여 좋은 점도 있었으나, 다중상속으로 인한 문제점도 많아서 C++에서 발전된 java 에서는 다중상속을 막았습니다.

하지만 java에서도 interface를 통해서는 다중구현이 가능하여, 여러 가지 타입으로 객체를 선언할 수 있습니다. 정확하게 말하면 java는 다중상속이 아닌 ‘다형성(type이 다양하게 존재할 수 있다)’이 존재한다고 이해하셔야 합니다.

인터페이스를 구현하는 것은 클래스를 상속받는 것과 비슷합니다

다만, ‘extends’ 대신 ‘implements’를 사용합니다

⑴ 단일상속(O)

public class ChildClass extends ParentClass1 {

}

⑵ 다중상속(X)

public class ChildClass extends ParentClass1, ParentClass2, ParentClass3 {

}

⑶ 다중구현(O)

public class ChildClass implements IFunction1, IFunction2, IFunction3 {

}

⑷ 인터페이스의 다중상속(O)

|  |  |  |
| --- | --- | --- |
|  | 단일 | 다중 |
| class  ▲  class | 단일상속 | 다중상속(X) |
| interface  ↑  class | 단일구현 | 다중구현 |
| interface  ▲  interface | 단일상속 | 다중상속(O) |
| Interface class  ↖ ▲  class | class className extends C implements I, I2, I3{  } | |

public interface I1 {

public int *i1* = 1;

public void m1();

}

public interface I2 {

public static final int *i2* = 2;

public void m2();

}

public interface I3 extends I1, I2 {

public int *i3* = 3;

public void m3();

}

public class TestClass implements I3 {

@Override

public void m1() {System.*out*.println("상수i1:"+*i1*);}

@Override

public void m2() {System.*out*.println("상수i2:"+*i2*);}

@Override

public void m3() {System.*out*.println("상수i3:"+*i3*);}

}

public interface I11 {

public int *i11* = 11;

public void m11();

}

public class TestChildClass extends TestClass implements I11 {

@Override

public void m11() {System.*out*.println("상수i11:"+*i11*);}

}

public class TestMain {

public static void main(String[] args) {

TestChildClass test = new TestChildClass();

//test.i3 = 7; 인터페이스안의 멤버변수는 static final 변수(상수)이므로 값 변경 불가

test.m1();

test.m2();

test.m3();

test.m11();

}

}

2. 인터페이스를 통한 예제만들기

<예제1> 로봇 장난감 만들기 프로그래밍

마트에 장난감 코너에 가보면 로봇 장난감이 많습니다. 장난감에는 해당 장난감을 가지고 놀 수 있는 어린아이의 연령대가 표시되어 있습니다.

예를 들어 미사일이 나가는 로봇은 위험성이 존재하여 4세 이상의 어린이만 구매 가능하고, 곰돌이 로봇의 경우에는 귀엽고, 위험성이 없어 0세 이상의 모든 어린아이들이 가지고 놀 수 있습니다. 이러한 상황을 java 프로그램으로 구현해 보도록 하겠습니다.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 미사일발사 | 불빛발사 | 팔다리움직임 |
| 곰돌이 | X | X | O |
| 마징가 | O | X | O |
| 비행기 | O | O | X |

public interface IToy { }

public interface IMoveArmLeg extends IToy {

void canMoveArmLeg();

}

public interface ILight {

void canLight();

}

public interface IMissile extends IToy {

void canMisile();

}

public class PoohToy implements IMoveArmLeg {

public PoohToy() {

System.*out*.println("곰돌이입니다");

canMoveArmLeg();

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

}

@Override

public void canMoveArmLeg() {System.*out*.println("팔다리를 움직일 수 있습니다");}

}

public class MazingerToy implements IMissile, IMoveArmLeg{

public MazingerToy() {

System.*out*.println("마징가입니다");

canMisile();

canMoveArmLeg();

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

}

@Override

public void canMoveArmLeg() {System.*out*.println("팔다리를 움직일 수 있습니다");}

@Override

public void canMisile() {System.*out*.println("미사일을 발사할 수 있습니다");}

}

public class AirPlaneToy implements IMissile, ILight{

public AirPlaneToy() {

System.*out*.println("비행기입니다");

canLight();

canMisile();

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

}

@Override

public void canLight() {System.*out*.println("불빛발사 가능합니다");}

@Override

public void canMisile() {System.*out*.println("미사일을 발사할 수 있습니다");}

}

public class TestMain {

public static void main(String[] args) {

/\*PoohToy pooh = new PoohToy();

MazingerToy mazinger = new MazingerToy();

AirPlaneToy airPlanToy = new AirPlaneToy();\*/

IToy pooh = new PoohToy();

IToy mazinger = new MazingerToy();

IToy airPlanToy = new AirPlaneToy();

IToy[] toys = {pooh, mazinger, airPlanToy};

for(IToy t : toys)

System.*out*.println(t.getClass().getName());

}

}

<예제2> 볼륨을 올리거나 내리는 인터페이스를 만들어 그 인터페이스를 활용한 클래스(Radio, TV, Speaker\_를 만들어봅시다.

**public** **interface** IVolume {

**void** volumeUp();

**void** volumeUp(**int** level);

**void** volumeDown();

**void** volumeDown(**int** level);

}

**public** **class** Speaker **implements** IVolume {

**private** **final** **int** SPEAKER\_MAX\_VOLUME=100;

**private** **final** **int** SPEAKER\_MIN\_VOLUME=0;

**private** **int** volumeLevel;

@Override

**public** **void** volumeUp() {

**if**(volumeLevel<SPEAKER\_MAX\_VOLUME) {

volumeLevel++;

System.***out***.println("스피커 볼륨을 1만큼 올려 "+volumeLevel);

}**else** {

System.***out***.println("스피커 볼륨이 최대치입니다");

}

}

@Override

**public** **void** volumeUp(**int** level) {

// 98 - > 10만큼

volumeLevel += level;

**if**(volumeLevel>SPEAKER\_MAX\_VOLUME) {

volumeLevel = SPEAKER\_MAX\_VOLUME;

System.***out***.println("스피커 볼륨이 최대치입니다");

}//if

System.***out***.println("스피커 볼륨을 "+level+"만큼 올려 "+volumeLevel);

}

@Override

**public** **void** volumeDown() {

**if**(volumeLevel>SPEAKER\_MIN\_VOLUME) {

volumeLevel--;

System.***out***.println("스피커 볼륨을 1만큼 내려 "+volumeLevel);

}**else** {

System.***out***.println("스피커 볼륨이 최소치입니다");

}

}

@Override

**public** **void** volumeDown(**int** level) {

volumeLevel -= level;

**if**(volumeLevel<SPEAKER\_MIN\_VOLUME) {

volumeLevel = SPEAKER\_MIN\_VOLUME;

System.***out***.println("스피커 볼륨이 최소치입니다");

}

System.***out***.println("스피커 볼륨을 "+level+"만큼 내려 "+volumeLevel);

}

}

**public** **class** TV **implements** IVolume {

**private** **int** volumeLevel;

**private** **final** **int** TV\_MAX\_VOLUME = 20;

**private** **final** **int** TV\_MIN\_VOLUME = 0;

@Override

**public** **void** volumeUp() {

**if**(volumeLevel<TV\_MAX\_VOLUME) {

volumeLevel++;

System.***out***.println("TV 볼륨을 1만큼 올려 "+volumeLevel);

}**else** {

System.***out***.println("TV 볼륨이 최대치여서 올리지 못 했습니다");

}

}

@Override

**public** **void** volumeUp(**int** level) {

**if**(volumeLevel+level<=TV\_MAX\_VOLUME) {

volumeLevel += level;

System.***out***.println("TV볼륨을 "+level+"만큼 올려서 "+volumeLevel);

}**else** {// 현재 볼륨이 18일 때 볼륨을 10만큼 올린다고 하면 2

//int tempLevel = level -((volumeLevel+level)- TV\_MAX\_VOLUME);

**int** tempLevel = TV\_MAX\_VOLUME - volumeLevel;

volumeLevel = TV\_MAX\_VOLUME;

System.***out***.println("TV볼륨을 "+level+"만큼 못 올리고 "+

tempLevel+"만큼 올려 현재 최대치가 되었습니다");

}

}

@Override

**public** **void** volumeDown() {

**if**(volumeLevel>TV\_MIN\_VOLUME) {

volumeLevel--;

System.***out***.println("TV 볼륨을 1만큼 내려 "+volumeLevel);

}**else** {

System.***out***.println("TV 볼륨이 최소치여서 못 내렸어요");

}

}

@Override

**public** **void** volumeDown(**int** level) {

**if**(volumeLevel-level>=TV\_MIN\_VOLUME) {

volumeLevel -= level;

System.***out***.println("TV볼륨을 "+level+"만큼 내려서 "+volumeLevel);

}**else** {// 현재 볼륨이 2일 때 볼륨을 10만큼 내린다고 하면 2

**int** tempLevel = volumeLevel-TV\_MIN\_VOLUME;

volumeLevel = TV\_MAX\_VOLUME;

System.***out***.println("TV볼륨을 "+level+"만큼 못 내리고 "+

tempLevel+"만큼 내려 현재 최소치가 되었습니다");

}

}

}

public class TV implements Volume {

private int volumeLevel;

public TV() {volumeLevel=0;}

public TV(int volumeLevel) {this.volumeLevel=volumeLevel;}

@Override

public void volumeUp() {

volumeLevel++;

if(volumeLevel>50) {

volumeLevel=50;

System.*out*.println("TV 최대 볼륨은 현재 볼륨 "+volumeLevel);

}else {

System.*out*.println("TV 볼륨을 1만큼 올려서 "+volumeLevel);

}

}

@Override

public void volumeUp(int level) {

volumeLevel += level;

if(volumeLevel>50) {

volumeLevel=50;

System.*out*.println("TV 최대 볼륨은 현재 볼륨 "+volumeLevel);

}else {

System.*out*.println("TV 볼륨을 1만큼 올려서 "+volumeLevel);

}

}

@Override

public void volumeDown() {

volumeLevel--;

if(volumeLevel<0) {

volumeLevel = 0;

System.*out*.println("TV 최저 볼륨은 현재 볼륨 "+volumeLevel);

}else {

System.*out*.println("TV 볼륨을 1만큼 내려서 "+volumeLevel);

}

}

@Override

public void volumeDown(int level) {

volumeLevel -= level;

if(volumeLevel<0) {

volumeLevel = 0;

System.*out*.println("TV 최저 볼륨은 현재 볼륨 "+volumeLevel);

}else {

System.*out*.println("TV 볼륨을 1만큼 내려서 "+volumeLevel);

}

}

}

public class VolTestMain {

public static void main(String[] args) {

Speaker speaker = new Speaker();

Radio radio = new Radio(3);

TV tv = new TV();

Volume vol[] = new Volume[3];

vol[0] = speaker;

vol[1] = radio;

vol[2] = tv;

speaker.volumeUp(10);

radio.volumeUp(10);

tv.volumeUp(10);

System.*out*.println("이제부터는 인터페이스를 이용한 호출입니다");

for(int i=0 ; i<vol.length ; i++)

vol[i].volumeUp(10);

for(Volume v:vol)

v.volumeUp();

}

}

<예제 3> 대출관련 인터페이스와 단행본 클래스



↑



public interface ILendable {

byte *STATE\_BORROWED* = 1; // 대출중

byte *STATE\_NORMAL* = 0; // 대출되지 않은 상태

void checkOut(String borrower, String checkOutDate); // 대출

void checkIn(); // 반납

void printState();//대출상태 출력

}

public class Book implements ILendable {

private String requestNo; // 청구번호

private String bookTitle; // 책제목

private String writer;

private String borrower;

private String checkOutDate;

private byte state;

public Book(String requestNo, String bookTitle, String writer) {

this.requestNo = requestNo;

this.bookTitle = bookTitle;

this.writer = writer;

state = *STATE\_NORMAL*;

}

@Override

public void checkOut(String borrower, String checkOutDate) {

if(state!=*STATE\_NORMAL*) {

System.*out*.println("대출불가");

return;

}

this.borrower = borrower;

this.checkOutDate = checkOutDate;

state = *STATE\_BORROWED*;

System.*out*.println("\""+bookTitle+"\"이(가) 대출되었습니다.");

System.*out*.println("[대출인] "+borrower);

System.*out*.println("[대출일] "+checkOutDate);

}

@Override

public void checkIn() {

borrower = null;

checkOutDate = null;

state = *STATE\_NORMAL*;

System.*out*.println("\""+bookTitle+"\"이(가) 반납되었습니다.");

}

public void printState() {

if(state==*STATE\_BORROWED*) {

System.*out*.println("\""+bookTitle+"\"은(는) 대출 중입니다");

}else if(state==*STATE\_NORMAL*) {

System.*out*.println("\""+bookTitle+"\"은(는) 대출 가능합니다");

}else {

System.*out*.println("\""+bookTitle+"\"은(는) 유령상태입니다");

}

}

}

public class BookMain {

public static void main(String[] args) {

Book obj = new Book("409101a","이것이 자바다", "신용권");

obj.printState();

obj.checkOut("홍길동", "20171010"); // 홍길동이 대출함

obj.printState();

obj.checkIn();

obj.printState(); // 반납함

}

}

※ 인터페이스의 장점

1. 개발시간을 단축시킬 수 있다.

일단 인터페이스가 작성되면, 이를 사용해서 프로그램을 작성하는 것이 가능하다. 메서드를 호출하는 쪽에서는 메서드의 내용에 관계없이 선언부만 알면 되기 때문이다. 그리고 동시에 다른 한 쪽에서는 인터페이스를 구현하는 클래스를 작성하도록 하여, 인터페이스를 구현하는 클래스가 작성될 때까지 기다리지 않고도 양쪽에서 동시에 개발을 진행할 수 있다.

1. 표준화가 가능하다

프로젝트에 사용되는 기본 틀을 인터페이스로 작성한 다음, 개발자들에게 인터페이스를 구현하여 프로그램을 작성하도록 함으로써 보다 일관되고 정형화된 프로그램의 개발이 가능하다

1. 서로 관계없는 클래스들에게 관계를 맺어 줄 수 있다.

서로 상속관계에 있지도 않고, 같은 조상클래스를 가지고 있지 않은 서로 아무런 관계도 없는 클래스들에게 하나의 인터페이스를 공통적으로 구현하도록 함으로써 관계를 맺어 줄 수 있다

1. 독립적인 프로그래밍이 가능하다

인터페이스를 이용하면 클래스의 선언과 구현을 분리시킬 수 있기 때문에 실제구현에 독립적인 프로그램을 작성하는 것이 가능하다. 클래스와 클래스간의 직접적인 관계를 인터페이스를 이용해서 간접적인 관계로 변경하면, 한 클래스의 변경이 관련된 다른 클래스에 영향을 미치지 않는 독립적인 프로그래밍이 가능하다.

public interface I {

public void method();

}

public class A implements I {

@Override

public void method() { System.*out*.println("A method()"); }

}

public class B implements I {

@Override

public void method() { System.*out*.println("B method()"); }

}

public class User {

public void aorbUse(I i) {

i.method();

}

}

public class TestMain {

public static void main(String[] args) {

User user = new User();

A a = new A();

B b = new B();

user.aorbUse(a);

user.aorbUse(b);

//user.aorbUse(new A());

//user.aorbUse(new B());

}

}

3. 인터페이스와 추상클래스

(공통점)

1. 추상메소드를 가지고 있습니다 - 추상메소드를 가지고 있어 하위 클래스에서 구현해야 합니다.
2. 변수 타입이 목적 입니다 - 객체생성이 목적이 아닌 변수 타입을 정의하는 것이 목적입니다.
3. 객체 생성은 anonymose(익명클래스)를 이용해야 합니다.

public class BookMain {

public static void main(String[] args) {

Book obj = new Book("409101a","이것이 자바다", "신용권");

obj.printState();

obj.checkOut("홍길동", "20171010"); // 홍길동이 대출함

obj.printState();

obj.checkIn();

obj.printState(); // 반납함

ILendable ob = new ILendable() {

@Override

public void checkOut(String borrower, String checkOutDate) {

System.*out*.println("test : "+borrower+" : "+checkOutDate);

}

@Override

public void checkIn() {

System.*out*.println("반납");

}

};

ob.checkIn();

}

}

(차이점)

상속, 구현 - 추상메소드는 상속을 통한 사용이고, 인터페이스는 구현을 통한 사용 입니다.

구성요소 차이 - 추상클래스는 일반 클래스와 동일하게 변수, 메소드의 모든 기능을 사용할 수 있지만, 인터페이스는 상수와 추상메소드만이 존재 합니다.

단일상속, 다중구현 - 추상클래스는 상속이므로 단일 상속만 지원하고, 인터페이스는 다중구현이 가능 합니다.

4. 이상 상속, 추상클래스, 인터페이스에서 변수타입과 객체타입은 다를 수 있다는 것을 보았습니다.

아래는 객체의 형변환(캐스트)와 캐스트 가능성을 검사하는 instanceof 연산자를 사용해 본다.

다음과 같은 실행결과를 얻도록 코드를 완성한다.

메서드명 : void action(Robot r)

기능 : 주어진 객체의 메서드를 호출한다.

DanceRobot인 경우, dance()를 호출하고,

SingRobot인 경우, sing()을 호출하고,

DrawRobot인 경우, draw()를 호출한다.

public interface Robot { }

public class DanceRobot implements Robot {

public void dance(){System.*out*.println("춤을 춥니다");}

}

public class SingRobot implements Robot {

public void sing(){System.*out*.println("노래를 합니다");}

}

public class DrawRobot implements Robot {

public void draw(){System.*out*.println("그림을 그립니다");}

}

public class RobotOrder {

public void action(Robot r) {

if(r instanceof DanceRobot) {

DanceRobot dr = (DanceRobot)r;

dr.dance();

}else if(r instanceof SingRobot){

SingRobot sr = (SingRobot)r;

sr.sing();

} else if(r instanceof DrawRobot){

DrawRobot dr = (DrawRobot)r;

dr.draw();

}else {

System.*out*.println("아무것도 할 수 없는 로봇이여요");

}//if

}//action()

}//class

public class RobotMain {

public static void main(String[] args) {

Robot danceRobot = new DanceRobot();

RobotOrder order = new RobotOrder();

if(danceRobot instanceof DanceRobot) {

//((DanceRobot)danceRobot).dance();

DanceRobot d = (DanceRobot)danceRobot;

d.dance();

}

DanceRobot danceR = new DanceRobot();

SingRobot singR = new SingRobot();

DrawRobot drawR = new DrawRobot();

Robot[] robots = {danceR, singR, drawR};

for(Robot r:robots) {

order.action(r);

}

}

}

5. default 메소드와 인터페이스의 확장

기존 인터페이스의 이름과 추상 메소드의 변경 없이 디폴트 메소드만 추가할 수 있기 때문에 이전에 개발한 구현 클래스를 그대로 사용할 수 있으면서 새롭게 개발하는 클래스는 디폴트 메소드를 활용할 수 있다.

// 상수, 추상메소드, default메소드, static메소드

public interface RemoteControl {

public int *MAX\_VOLUME* = 10; // 상수(final변수)

public int *MIN\_VOLUME* = 10;

public /\*abstract\*/ void turnOn(); //추상메소드

public /\*abstract\*/ void turnOff();

public /\*abstract\*/ void setVolume(int volume);

public default void setMute(boolean mute) {//default 메소드

if(mute) {

System.*out*.println("무음 처리합니다");

}else {

System.*out*.println("무음 해제합니다");

}

}

public static void changeBattery() { // static 메소드

System.*out*.println("건전지를 교환합니다");

}

}

public class Audio implements RemoteControl {

private int volume;

@Override

public void turnOn() {

System.*out*.println("오디오를 켭니다");

}

@Override

public void turnOff() {

System.*out*.println("오디오를 끕니다");

}

@Override

public void setVolume(int volume) {

if(*MAX\_VOLUME*<volume) {

this.volume = *MAX\_VOLUME*;

}else if(*MIN\_VOLUME*>volume) {

this.volume = *MIN\_VOLUME*;

}else {

this.volume = volume;

}

System.*out*.println("현재 오디오 볼륨 : "+volume);

}

}

public class TV implements RemoteControl {

private int volume;

@Override

public void turnOff() {

System.*out*.println("TV를 끕니다");

}

@Override

public void turnOn() {

System.*out*.println("TV를 켭니다");

}

@Override

public void setVolume(int volume) {

if(*MAX\_VOLUME*<volume) {

this.volume = *MAX\_VOLUME*;

}else if(*MIN\_VOLUME*>volume){

this.volume = *MIN\_VOLUME*;

}else {

this.volume = volume;

}

System.*out*.println("현재 TV 볼륨 : "+this.volume);

}

}

public class TestMain {

public static void main(String[] args) {

TV tv = new TV();

Audio audio = new Audio();

tv.turnOn();

tv.setVolume(20);

tv.setMute(true);//default 메소드

// tv.changeBattery();

RemoteControl.*changeBattery*();//static 메소드

System.*out*.println("최대볼륨:"+RemoteControl.*MAX\_VOLUME*);//상수

System.*out*.println("최대볼륨:"+TV.*MAX\_VOLUME*);//상수

System.*out*.println("최대볼륨:"+Audio.*MAX\_VOLUME*);//상수

audio.setMute(false); // default 메소드

System.*out*.println("최소볼륨:"+Audio.*MIN\_VOLUME*);

}

}