day13(8/17)

**package** com.ict.em;

//추상화: 구체적인 개념으로부터 공통된 부분만 추려내어 일반화할 수 있도록 하는 것을 의미

//즉, 일반적으로 사용할 수 있는 단계가 아닌 아직 미완성적 개념이다.

//추상 클래스이면 추상 메소드를 가지고 있다.

//추상 메소드는 public abstract void 이름()

//추상 메소드를 하나라도 가지게 되면 추상 클래스가 된다.

//그리고 이런 추상클래스 또한 다음과 같이 추상 클래스임을 명시해야 한다.

//abstract class AbsEx{}

//추상클래스는 인스턴스화 할 수 없다.

//abstract, final을 같이 사용할 수 없다.

//-->상속받아서 구현해야 하기 때문에 final을 사용할 수 없다.

추상클래스 예제 1)

**abstract** **class** AbsEx01{

**int** su1 = 10;

**static** **int** *su2* = 20;

**final** **int** su3 = 30; //상수

**static** **final** **int** ***SU4*** = 40; //상수

**public** **void** prn() {

su1++;

*su2*++;

//su3, SU4는 상수 이기 때문에 값을 변화시킬 수 없다.

}

//static 메소드는 static 변수만 사용할 수 있다.

**public** **static** **void** prn2() {

//System.out.println(su1); //static 변수가 아니므로 사용할 수 없다.

System.***out***.println(*su2*);

//System.out.println(su3);

System.***out***.println(***SU4***);

}

**public** **abstract** **void** prn3(); //body가 없는 메소드-->추상 메소드

}

//상속

//추상 클래스를 상속 받으면 자식 class는 두가지로 나눠진다.

//추상클래스(부모)-->상속-->일반 클래스(자식): 부모가 가진 추상 메소드를 반드시 오버라이딩 해서 반드시 구체화해야 함

//추상클래스(부모)-->상속-->추상 클래스(자식): 부모가 가진 추상 메소드를 오버라이딩하지 않으면 자식도 추상클래스가 되어야 함.

**public** **class** AbsEx02 **extends** AbsEx01 {

//추상 클래스인 부모 클래스를 상속받아서 일반 클래스가 되기 위해서는

//반드시 추상메소드를 오버라이딩해야 한다.

@Override

**public** **void** prn3() {

}

}

**abstract** **class** AbsEx03 **extends** AbsEx01{

//추상클래스인 부모클래스를 상속받아서 추상메소드를 오버라이딩하지 않으려면

//자식클래스도 추상클래스가 되어야 한다.

}

**abstract** **class** AbsEx04 **extends** AbsEx03{

**abstract** **void** result(); //추상 메소드

}

//추상메소드를 모두 오버라이딩해야 한다.

**class** AbsEx05 **extends** AbsEx04{

@Override

**void** result() {

}

@Override

**public** **void** prn3() {

}

}

추상 클래스 예제 2)

//같은 package

**public** **abstract** **class** Shape {

**public** **abstract** **void** draw();

}

**public** **class** Circle **extends** Shape {

@Override

**public** **void** draw() {

System.***out***.println("원을 그리다.");

}

}

**public** **class** Rect **extends** Shape {

// 추상 클래스를 상속받아서 일반 클래스로 만들때에는

// 추상메소드를 모두 오버라이딩해야 한다.

@Override

**public** **void** draw() {

System.***out***.println("사각형을 그리다.");

}

}

**public** **class** Triangle **extends** Shape {

@Override

**public** **void** draw() {

System.***out***.println("삼각형 그리기");

}

}

//추상 메소드와 일반 메소드를 가질 수 있다.

//모든 추상 메소드는 구현해야 사용할 수 있다.

//단일 상속만 가능하다(오직 하나의 클래스에 대해서만 extends해야 한다.)

//업캐스팅(부모 타입으로 자식 클래스의 인스턴스를 생성하는 것)이 가능하다.

**public** **class** Ex01\_Main {

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

Shape[] ref = {**new** Rect(), **new** Triangle(), **new** Circle()};

//ref는 Shape타입 이기 때문에 모든 자식클래스의 타입을 업캐스팅할 수 있다.

**for**(Shape k : ref) {

k.draw();

}

}

}

\*인터페이스 예제 1)

**package** com.ict.em;

//인터페이스:

//음식점의 메뉴판과 같은 것이다. 메뉴판을 보고 고객이 원하는 음식을 요청하게 되는데 메뉴판 자체가 음식을 주지 않는다.

//실제 음식은 주방이라는 곳에서 나오므로 메뉴판은 고객이 호출할 수 있는 서비스의 목록이다.

//-상수, 추상 메소드(바디가 없는 메소드)

//-생성자가 없다. 객체로 만들 수 없다.

//인터페이스가 같다는 뜻은 대체가 가능, 호환이 가능하다는 뜻이다.

//유일하게 다중상속이 가능하다.

//인터페이스의 상속

//1.일반 클래스: 인터페이스가 가지고 있는 추상 메소드들을 반드시 오버라이딩해야한다.

//2.인터페이스: 오버라이딩할 필요 없다.(extends)

//3.추상클래스: 오버라이딩할 필요 없다.

//[접근 제한자] interface [인터페이스명]{

//상수;

//추상 메소드;

//}

//\*\*인터페이스 내에는 상수 또는 메소드들만 정의가 가능하다.\*\*(

//static final, abstract를 생략해도 인식한다.

//또한 인터페이스에 있는 모든 메소드는 내부적으로 모두 public이다.

//접근 지정자를 명시하지 않은 인터페이스 내부의 모든 메소드를 오버라이딩 할 경우

//자식 클래스에서 접근 지정자를 축소시켜서 지정할 수 없으므로 public으로 지정해야 한다.

**public** **interface** InterfaceEx01 {

//상수, 추상메소드만 갖는다.

//Interface내에서 변수를 선언하면 무조건 static final로 선언된다.-->상수

**int** ***su1*** = 10;

**final** **int** ***su2*** = 20;

**static** **int** ***su3*** = 30;

**static** **final** **int** ***su4*** = 40;

//static final로 선언하지 않아도 static final로 선언된다.

//추상메소드(인터페이스에서는 abstract 생략 가능)

**public** **void** prn1();

**public** **abstract** **void** prn2();

//일반메소드는 오류

/\*public void prn3() {

}\*/

}

**package** com.ict.em;

//\*인터페이스끼리는 extends

//\*인터페이스를 class로 구현할때는 implements 사용

//상속

//1.일반 클래스: 반드시 오버라이딩해야 한다.

//implements(구현) 예약어 사용

**class** InterfaceEx02 **implements** InterfaceEx01 {

@Override

**public** **void** prn1() {

}

@Override

**public** **void** prn2() {

}

}

//2.추상 클래스: 오버라이딩

//일반 멤버들을 가질 수 있다.

**abstract** **class** InterfaceEx03 **implements** InterfaceEx01{

**int** k = 10;

}

//3.인터페이스: 오버라이딩

//상수, 추상메소드만 가질 수 있다. 일반 멤버들을 가질 수 없다.

**interface** InterfaceEx04 **extends** InterfaceEx01{

**final** **static** **int** ***su5*** = 9;

**public** **void** sum();

//일반 클래스로 구현할 때는 모두 오버라이딩 해야한다.

}

**class** InterfaceEx05 **implements** InterfaceEx04{

@Override

**public** **void** prn1() {

}

@Override

**public** **void** prn2() {

}

@Override

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

}

}

\*인터페이스 예제 2)

**package** com.ict.bm;

**public** **interface** RemoteControl {

// 인터페이스: 상수, 추상메소드

**int** ***MAX\_VOLUME*** = 10;

**int** ***MIN\_VOLUME*** = 0;

// public static final이 생략

// 컴파일러가 자동적으로 추가한다.

**void** turnOn(); // public abstract 생략

**void** turnOff();

**void** setVolume(**int** volume);

// 인터페이스에 들어가는 것이 꼭 상수나 추상메소드만이 아니다.

// 디폴트 메소드 : default라고 선언하면 여러개의 함수를 만들 수 있다.

// 디폴트 메소드 : 오버라이드 가능

**default** **void** setMute(**boolean** mute) {

**if** (mute) {

System.***out***.println("무음 처리합니다.");

} **else** {

System.***out***.println("무음 해제합니다.");

}

}

// static 메소드를 추가할 수 있다.

// static 메소드는 오버라이딩 불가

**static** **void** testStaticMethod(**int** mute) {

}

}

**package** com.ict.bm;

**public** **class** Tv **implements** RemoteControl {

**private** **int** volume = 0;

@Override

**public** **void** turnOn() {

System.***out***.println("TV의 전원을 켠다.");

}

@Override

**public** **void** turnOff() {

System.***out***.println("TV의 전원을 끈다.");

}

@Override

**public** **void** setVolume(**int** volume) {

**this**.volume = volume;

}

// getter

**public** **int** getVolume() {

**return** volume;

}

**int** aaa = 0;

@Override

**public** **void** setMute(**boolean** mute) {

**if** (mute) {

aaa++; // 자식 클래스의 인스턴스 멤버 접근 가능

System.***out***.println("오버라이드된 setMute 켜기");

} **else** {

System.***out***.println("오버라이드된 setMute 끄기");

}

}

//@Override - RemoteControl의 testStaticMethod를 override한 것이 아니다.

//static은 오버라이드가 불가능하다.

**public** **static** **void** testStaticMethod(**int** mute) {

//aaa++; static 메소드에에서 인스턴스 멤버는 호출 불가

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

}

}

**package** com.ict.bm;

**public** **class** TvUser {

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

//RemoteControl의 상수값 접근

System.***out***.println("최대 볼륨=>"+RemoteControl.***MAX\_VOLUME***);

System.***out***.println("최소 볼륨=>"+RemoteControl.***MIN\_VOLUME***);

Tv tv = **new** Tv();

tv.turnOn();

tv.turnOff();

tv.setVolume(20);

System.***out***.println("현재의 볼륨=>" +tv.getVolume());

tv.*testStaticMethod*(4);

}

}

\*인터페이스 예제3)

**package** com.ict.cm;

**public** **interface** Runnable {

**int** ***MAX\_SPEED*** = 100;

**int** ***MIN\_SPEED*** = 0;

**void** run();

/\* static void start() {

run(); static 메소드는 인스턴스 메소드 사용 불가

\*/

//start-->run()을 호출하는 것

**default** **void** start() {

System.***out***.println("최대"+***MAX\_SPEED***+"까지 속도를 낼 수 있다.");

run();

}

}

**public** **class** Car **implements** Runnable{

@Override

**public** **void** run() {

System.***out***.println("자동차가 달린다.");

}

}

**package** com.ict.cm;

**public** **class** CarUser {

**public** **void** action(Runnable car) {

car.start();

}

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

Car car = **new** Car();

car.start();// interface에 선언된 default를 이용해서 run()을 호출

car.run();

// 핵심적인 기능은 run()에서 구현

// run()에서 공통적인 부분은 start()에서 만들어 놓는다.

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

Runnable runner = **new** Car(); // 업캐스팅

runner.start();

// Runnable autobike = new Runnable();

// 인터페이스는 직접 객체 생성 불가능

// 익명 객체 생성 가능

// 인터페이스, 추상클래스를 선언하자마자 구현하는 것을 의미한다.

// 클래스의 이름이 없기 때문에 익명객체라고 한다.

Runnable autobike = **new** Runnable() {

@Override

**public** **void** run() {

System.***out***.println("오토바이가 달린다.");

}

};

//autobike.start();

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

CarUser user = **new** CarUser();

user.action(car);

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

//익명객체를 인자값으로 사용할 수 있다.-많이 사용하는 형태

//인터페이스만 있고 그를 구현한 클래스가 존재하지 않을 때 바로 구현하면서 익명 객체를 생성한다.

user.action(**new** Runnable() {

@Override

**public** **void** run() {

System.***out***.println("람보르기니가 달린다.");

}

});

}

}

인터페이스 예제4)

**package** com.ict.em;

**public** **interface** Soldier{

**public** **void** action();

}

**public** **interface** Weapon {

**public** **void** use();

}

**public** **class** Gun **implements** Weapon{

@Override

**public** **void** use() {

System.***out***.println("권총을 사용한다. 탕!탕!탕!");

}

}

**public** **class** Sword **implements** Weapon{

@Override

**public** **void** use() {

System.***out***.println("검을 사용한다. 휙!휙!휙!");

}

}

**public** **class** GameClient {

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

Weapon gun = **new** Gun();

Weapon sword = **new** Sword();

Soldier rambo = **new** Rambo();

rambo.setWeapon(gun);

rambo.action();

rambo.setWeapon(sword);

rambo.action();

rambo.setWeapon(**new** Weapon() {

@Override

**public** **void** use() {

System.***out***.println("활을 쏜다 슉! 슉!");

}

});

rambo.action();

}

}

인터페이스 예제5)

**public** **interface** Soundable {

String sound();

}

**public** **class** Cat **implements** Soundable{

@Override

**public** String sound() {

**return** "야옹";

}

}

**public** **class** Dog **implements** Soundable{

@Override

**public** String sound() {

**return** "멍멍";

}

}

**public** **class** SoundableExample {

**private** **static** **void** printSound(Soundable soundable) {

System.***out***.println(soundable.sound());

}

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

Soundable cat = **new** Cat();

Soundable dog = **new** Dog();

*printSound*(cat);

*printSound*(dog);

}

}

인터페이스 예제5)

**public** **interface** DataAccessObject {

**public** **void** select();

**public** **void** insert();

**public** **void** update();

**public** **void** delete();

}

**public** **class** MysqlDao **implements** DataAccessObject {

@Override

**public** **void** select() {

System.***out***.println("MySql DB의 Select Work 입니다.");

}

@Override

**public** **void** insert() {

System.***out***.println("MySql DB의 insert Work 입니다.");

}

@Override

**public** **void** update() {

System.***out***.println("MySql DB의 update Work 입니다.");

}

@Override

**public** **void** delete() {

System.***out***.println("MySql DB의 delete Work 입니다.");

}

}

**public** **class** OracleDao **implements** DataAccessObject{

@Override

**public** **void** select() {

System.***out***.println("Orcle DB의 Select Work 입니다.");

}

@Override

**public** **void** insert() {

System.***out***.println("Orcle DB의 insert Work 입니다.");

}

@Override

**public** **void** update() {

System.***out***.println("Orcle DB의 update Work 입니다.");

}

@Override

**public** **void** delete() {

System.***out***.println("Orcle DB의 delete Work 입니다.");

}

}

**public** **class** DaoExample {

**public** **static** **void** dbWork(DataAccessObject dao) {

dao.select();

dao.insert();

dao.update();

dao.delete();

}

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

*dbWork*(**new** OracleDao());

*dbWork*(**new** MysqlDao());

}

}

인터페이스 예제 6)

**public** **interface** Action {

**void** work();

}

**public** **class** ActionExample {

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

//인터페이스 Action을 구현한 익명의 내부 클래스

//익명의 내부 클래스: 인터페이스 Action을 따로 구현하지 않고

//바로 main에서 구현하여 사용하는 것을 말한다.

Action action = **new** Action() {

@Override

**public** **void** work() {

System.***out***.println("익명의 내부 클래스 action");

}

};

action.work();

}

}

\*다중상속을 위한 인터페이스

-클래스: 다중상속 X,

-인터페이스: 다중상속 가능

>>차이점:

추상 클래스

-abstract class로 선언

-일반 메소드와 추상 메소드를 모두 가질 수 있다.

-단일 상속만 가능하다

인터페이스

-클래스가 아니다(interface)

-public abstract 추상 메소드만 가질 수 있다. (일반 메소드는 가질 수 없다)

-일반 메소드: default메소드, static메소드 사용 가능

-구현을 하며 implements로 서브 클래스 정의

-다중 상속도 가능하다.

-static final 형태의 상수만 가질 수 있다.

>>공통점:

-모든 추상 메소드는 인스턴스 생성을 위한 서브 클래스에서 반드시 구현되어야 한다.

-업캐스팅이 가능하다.