Day10(8/13)

\*접근제한자

public< protected< default< private

-public: 공용, 누구나 접근 가능

-protected: 같은 패키지, 상속관계에서만 접근 가능

-default: 같은 패키지에서만 접근 가능 (접근제한자 생략하면 default)

-private: 같은 클래스에서만 접근가능 (외부에서 접근 불가)

private - 같은 클래스 내에서만 호출 가능 (데이터 은닉)

🡪private 필드에 접근하기 위해서 public 메소드를 정의한다.

🡪같은 class내의 getter, setter 메소드를 이용

-getter: private 변수들에게 접근할 때 사용

-setter: private 변수들의 데이터 값을 변경할 때 사용

**package** com.ict.bm;

//private 같은 class에서만 호출할 수 있다.

//public 으로 선언된 getter, setter를 사용해서 데이터에 접근해야 한다.

//private을 사용해서 class외부에서 바로 접근하지 못하게 하는 것을 정보 은닉이라 한다.

**class** Point{

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

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

//setter

**public** **void** setX(**int** \_x) {

x = \_x;

}

**public** **void** setY(**int** \_y) {

y = \_y;

}

//getter

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

**return** x;

}

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

**return** y;

}

}

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

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

Point pt = **new** Point();

pt.setX(100);

pt.setY(200);

System.***out***.println("pt.getX()=>" +pt.getX()); //100

System.***out***.println("pt.getY()=>" +pt.getY()); //200

}

}

**package** com.ict.bm;

**import** java.util.Scanner;

**class** People{

**private** String name;

**private** **int** age;

//setter

**public** **void** setName(String \_name) {

name = \_name;

}

**public** **void** setAge(**int** \_age) {

age = \_age;

}

//getter

**public** String getName() {

**return** name;

}

**public** **int** getAge() {

**return** age;

}

}

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

**static** Scanner *scan* = **new** Scanner(System.***in***);

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

//성명과 나이를 입력하는 프로그램을 만드시오.

//선언

People p1 = **new** People(); //객체화

String name = "";

**int** age = 0;

//입력 - 선언된 지역변수에 입력 받는다.

System.***out***.print("성명입력>>>");

name = *scan*.next();

System.***out***.print("나이입력>>>");

age = *scan*.nextInt();

//지역변수에 저장된 데이터를 setter를 이용해 대입

p1.setName(name);

p1.setAge(age);

//출력-객체에 저장된 데이터를 getter를 이용해서 출력

System.***out***.println("이름: "+p1.getName());

System.***out***.println("나이: "+p1.getAge());

}

}

-오버로딩(중복정의): 하나의 클래스 안에서 같은 이름을 가진 메소드가 여러 개 존재하는 것

단, 메소드 이름은 같으나 반드시 인자의 자료형이나 개수가 다르다.

-메소드를 구분하기 위한 시그니처

1. 메소드의 이름

2. 매개 변수의 자료형

3. 매개 변수의 개수

메소드의 오버로딩은 메소드의 이름을 동일하게 주되 시그니처를 다르게 하는 것

접근지정자와 리턴 값은 메소드의 오버로딩에 포함되지 않는 메소드의 구성요소이다.

\*다형성(Polymorphism):

메소드를 호출할 때에 동일한 접근 방식으로 호출하지만 다양한 결과를 얻을 수 있기 때문에 다형성을 제공한다.>>오버로딩, 오버라이딩, 인터페이스, upcasting, downcasting을 통해서 다형성 구현이 가능하다.

\*오버라이딩: 부모class의 메소드를 자식class에서 재정의 하는 것을 말한다.

**package** com.ict.bm;

**public** **class** Ex03\_Overloarding {

//오버로딩

**static** **void** printstr(String the\_string) {

System.***out***.println(the\_string);

}

**static** **void** printstr(**char** the\_char, **int** repeat\_cnt) {

**for** (**int** i = 0; i < repeat\_cnt; i++) {

System.***out***.print(the\_char);

}

}

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

*printstr*("슬프도록 아름다운");

*printstr*('A', 10);

}

}

**package** com.ict.bm;

**public** **class** Ex04 {

// 동일한 기능을 갖는 메소드가 모두 다른 이름으로 정의 한다면

// 프로그램을 작성할 때 마다 메소드 이름을 개별적으로 외워야할 것이다.

// 이를 피하기 위해서 오버로딩 기능을 제공한다.

**static** **void** prn(**int** x) {

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

}

**static** **void** prn(**int** x, **int** y) {

System.***out***.println(x + " " + y);

}

**static** **void** prn(**int** x, **int** y, **int** z) {

System.***out***.println(x + " " + y + " " + z);

}

// 메소드의 기능이 같더라도 전달 인자의 개수가 다르면

// 전달 인자의 개수가 다른 메소드를 여러개 정의해야한다.

// Varargs를 사용하면 메소드를 한개만 정의해 두면 된다.

**static** **void** prn(**int**... num) {

**for** (**int** i = 0; i < num.length; i++) {

System.***out***.print(num[i] + "\t");

}

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

}

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

*prn*(10, 20, 30);

*prn*(40, 50);

*prn*(60);

}

}

\*생성자의 특징

-클래스를 객체로 만들 때 1번 호출되어 사용된다.

-객체를 만들 때 초기값을 지정해주는 역할🡪멤버 필드의 초기화를 의미한다.

-반환형을 지정하지 않는다.

-생성자가 없으면 JVM이 자동으로 기본 생성자를 만들어 준다.

-하지만 개발자가 생성자를 하나라도 만들면 자동적으로 기본생성자가 생성되지 않는다

🡪따라서 생성자를 만들 때, 기본 생성자도 같이 만들어 준다.

-모든 클래스는 생성자를 가지고 있다. (Interface는 생성자가 없음🡪객체로 만들지 못함)

\*클래스 이름과 동일해야 하는 것>>파일명, 생성자

\*객체를 생성할 때 필요한 연산자>>new

\*클래스를 객체로 만드는 방법

-클래스 이름 참조변수 = new 생성자;

Ex) Ex01 test = new Ex01();//기본생성자

Scanner scan = new Scanner(System.in);//생성자, 기본 생성자 아님

-생성자도 오버로딩이 가능하다. 즉, 한 클래스에 여러 개의 생성자를 만들 수 있다.

**package** com.ict.bm;

**public** **class** Ex06{

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

Pointer pt01 = **new** Pointer();

Pointer pt02 = **new** Pointer(200,300);

pt01.showPoint();

pt02.showPoint();

}

}

**class** Pointer {

**int** x;

**int** y;

**public** Pointer() {

System.***out***.println("매개변수가 없는 디폴트 생성자");

}

// 생성자 오버로딩을 하면 JVM이 자동으로 디폴트 생성자를 만들어주지 않는다

// -->컴파일 오류가 발생한다.

// 생성자 오버로딩을 할때에는 디폴트 생성자도 같이 정의해야 한다.

**public** Pointer(**int** new\_x, **int** new\_y) {

System.***out***.println("매개변수가 있는 생성자");

x = new\_x;

y = new\_y;

}

**public** **void** showPoint() {

System.***out***.println(x + ", " + y);

}

}

**package** com.ict.bm;

**class** Music {

**private** String singer = "";

**private** String song = "";

// 생성자 오버로딩

**public** Music() {

}

**public** Music(String singer, String song) {

**this**.singer = singer;

**this**.song = song;

}

// getter

**public** String getSinger() {

**return** singer;

}

**public** String getSong() {

**return** song;

}

**public** String getData() {

**return** singer + "가 " + song + "을 부른다.";

}

// setter

**public** **void** setSinger(String singer) {

**this**.singer = singer;

}

**public** **void** setSong(String song) {

**this**.song = song;

}

}

**class** MusicPlayer {

**public** **void** play(Music m) {

System.***out***.println(m.getData());

}

//play() 메소드 오버로딩

**public** **void** play(Music[] mArr) {

//개선된 for문

**for**(Music k : mArr) {

System.***out***.println(k.getData());

}

/\*for (int i = 0; i < mArr.length; i++) {

System.out.println(mArr[i].getData());

}\*/

}

}

**public** **class** Ex08 {

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

Music m1 = **new** Music();

m1.setSinger("이선희");

m1.setSong("나는 괜찮아요");

Music m2 = **new** Music("A-pink", "1도 없어");

Music[] mArr = { m1, m2, **new** Music("조용필", "단발머리") };

MusicPlayer mp = **new** MusicPlayer();

mp.play(m1); // 이선희가 나는 괜찮아요를 부른다.

mp.play(m2);

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

mp.play(mArr);

}

}

\*전역변수, 지역변수의 이름이 같다

-우선 순위: 1. 지역변수 2. 전역변수

-전역 변수: 멤버 필드에 this를 붙인다.

-this란, 객체 안에서 자신을 지칭하는 예약어

-this는 자신의 주소를 가지고 있다.

-지역 변수는 메소드가 실행되야 만들어 진다.

\*this(): 생성자에서 다른 생성자를 호출할 때 사용

-생성자 이름을 사용하는 것이 아니다.

-반드시 첫번째 줄에서만 사용 가능

-참고) 메소드에서 다른 메소드 호출 가능 => 메소드 이름();//위치 상관없음

-참고) 생성자에서 메소드 호출 가능 => 메소드 이름();//위치 상관없음

\*static예약어: class변수, class 메소드

-해당 객체가 생성될 때 객체가 생성된 메모리 공간에 같이 존재하게 되지만

static으로 선언된 메소드나 변수들은 static 영역이라는 곳에 유일하게 만들어 진다.

공유 개념: 모든 객체(Object)들이 사용할 수 있다.

-객체생성과 상관없이 먼저 생성된다.

-누구나 접근 가능 -> 공용

-지역 변수나 클래스에는 정의할 수 없음

-예외적으로 내부클래스에서는 사용 가능

-static 메모리에 유일하게 만들어 진다. (이름 중복 안됨)

\*static 메소드

-인스턴스 변수(멤버 변수)를 참조할 수 없다.

-static 변수를 참조할 수 있다.

heap: 객체가 생성되어 저장되는 공간

instance: 객체가 생성될 때 같이 만들어 지는 변수, 상수, 메소드

static: 객체 실행 전에 자바가 미리 만들어 놓는 것

**package** com.ict.bm;

//this란, 객체 안에서 객체 자신을 지칭하는 예약어

//this는 객체 자신의 주소를 가지고 있다.

//지역 변수는 메소드가 실행되야 만들어진다.

//static메소드에서는 인스턴스 멤버를 사용할 수 없다.

//static 변수를 참조 할 수 있다.

//지역 변수로 static을 사용할 수 없다.

**class** Account{

**int** idx = 0;

**static** **int** *cnt* = 0;

Account(){

*cnt*++;

**this**.idx = *cnt*;

}

**void** showInfo() {

System.***out***.println("cnt=>"+*cnt*+" idx=>"+idx);

}

}

**public** **class** Ex09 {

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

Account ac01 = **new** Account();

ac01.showInfo();

Account ac02 = **new** Account();

Account ac03 = **new** Account();

Account ac04 = **new** Account();

//객체 생성 후에는 인스턴스 변수와 static변수 모두 사용가능

ac01.showInfo();

ac02.showInfo();

Account.*cnt* =10;

ac03.showInfo();

ac04.showInfo();

}

}