**[ 8 ] 객체지향 프로그램의 기초적인 코딩방법**

(cf) 캡슐화(Encapsulation) : 객체가 포함한 속성과 메서드는 객체간의 관계에 있어서 감추거나 권한에 따라 접근이 가능하게 처리하는 것을 말한다. 여기에 사용되는 keyword로 접근제어자(access modifier)가 있다

1. 클래스 제작

Package com.ch.ex;

public class ExClass {

private 자료형 인스턴스변수( = 속성 = 필드)명;

public ExClass(){ }

public method(){

. . .

}

}

1. 패키지명
2. 클래스명
3. **데이터(인스턴스 변수=멤버변수, 필드) :** 이 데이터는생성자나 setter를 이용해서 초기화하지 않으면 객체는 null, 숫자는 0, boolean은 false로 초기화되어 들어간다
4. 생성자 : 클래스명과 똑같이 리턴타입이 없는 메소드를 생성자라 하며 처음 클래스형 객체를 만들때 호출된다. 모든 클래스는 반드시 하나 이상의 생성자가 있어야 한다. 만약 하나도 없으면 JVM이 디폴트 생성자를 만들어 준다(new 연산자로 호출되는 메서드)
5. **메소드**
6. **Getter & setter**

**package** com.tj.square;

**public** **class** Square {

**private** **int** side;

**public** Square() {System.***out***.println("매개변수가 없는 생성자 호출했음");}

**public** Square(**int** side) {**this**.side = side;

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

}

**public** **int** area() {**return** side\*side;}

**public** **int** getSide() { **return** side; }

**public** **void** setSide(**int** side) {

**this**.side = side;

}

}**package** com.tj.square;

**public** **class** SquareMain {

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

Square s1 = **new** Square(5);

Square s2 = **new** Square(25);

//s1.setSide(5);

//s2.setSide(10);

System.***out***.println("s1의 넓이는 "+ s1.area());

System.***out***.println("s2의 넓이는 "+ s2.area());

}

}

2. 생성자의 이해 : 생성자는 매개변수 있는 생성자와 매개변수 없는 생성자 등 여러 종류의 생성자를 가질 수 있다.

**package** com.tj.human;

**public** **class** Man {

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

**public** Man() { System.***out***.println("파라미터가 없는 생성자 호출"); }

**public** Man(**int** age) {

**this**.age = age;

System.***out***.println("파라미터가 있는 생성자 호출");

}

}

**public** **class** Woman {

**public** Woman() {

System.***out***.println("Woman 클래스의 생성자 함수 왔어요");

}

}

**package** com.tj.human;

**import** com.tj.human.woman.Woman;

**public** **class** ManMain {

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

Man man1 = **new** Man();

Man man2 = **new** Man(20);

Woman woman = **new** Woman();

System.***out***.println(man2.equals(man1));

}

}

(ex)

**public** **class** Man {

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

**private** **int** height;

**private** **int** weight;

**private** String phoneNum;

// 파라미터값을 갖는 생성자

**public** Man(**int** age, **int** height, **int** weight, String phoneNum) {

**this**.age = age;

**this**.height = height;

**this**.weight = weight;

**this**.phoneNum = phoneNum;

}

**public** Man(**int** height, **int** weight) {

**this**.height = height;

**this**.weight = weight;

System.***out***.println("실행한 파라미터가 있는 생성자");

}

//파라미터가 없는 생성자

**public** Man(){

System.***out***.println("실행한 파라미터가 없는 생성자");

}

//파라미터가 달랑 하나인 생성자

**public** Man(**int** height){

**this**.height = height;

}

// 파라미터가 실수형인 생성자

**public** Man(**double** weight){

**this**.weight = (**int**)weight;

}

**double** calculateBMI(){

**double** result = weight / ((height/100.0)\*(height/100.0));

**return** result;

}

**public** **int** getAge() {**return** age; }

**public** **void** setAge(**int** age) {**this**.age = age;}

**public** **int** getHeight() {**return** height;}

**public** **void** setHeight(**int** height) {**this**.height = height;}

**public** **int** getWeight() {**return** weight;}

**public** **void** setWeight(**int** weight) {**this**.weight = weight;}

**public** String getPhoneNum() {**return** phoneNum;}

**public** **void** setPhoneNum(String phoneNum) {**this**.phoneNum = phoneNum;}

}

**public** **class** Woman {

**public** Woman(){

System.***out***.println("Woman 클래스의 생성자 함수 왔어요");

}

}

**public** **class** MainClass {

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

Woman hee1 = **new** Woman();

------------ --------------

①레퍼런스변수선언 ②인스턴스생성

Woman hee2 = **new** Woman();

System.***out***.println("hee1과 hee2는 같은가"+hee1.equals(hee2));

Man kang = **new** Man(185,75);

Man kim = **new** Man();

Man kang1;

kang1 = kang;

kim.setHeight(185);

kim.setWeight(75);

System.***out***.println("Kim과 Kang이 같은지 " +kim.equals(kang));

System.***out***.println("kang과 kang1이 같은지 " +kang.equals(kang1));

**double** biman = kang.calculateBMI();

**if**(biman>24)

System.***out***.println("Kang님 비만이니 다이어트하세요");

**else**

System.***out***.println("Kang님 비만 아니니 다이어트하지 마세요");

biman = kim.calculateBMI();

**if**(biman>24)

System.***out***.println("Kim님 비만이니 다이어트 하세요");

**else**

System.***out***.println("Kim님 비만아니니 다이어트하지 마세요");

}

}

클래스 객체

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

설계도 제품

객체 ≒ 인스턴스

3. this키워드의 이해

this란 객체자신

~~~~~~~~~~~~~~~~~~~~~~~~~~~~

객체지향 프로그래밍의 특징

(1) 캡슐화와 데이터은닉

(2) 다형성 : 같은 모습이지만 다른 기능 cf.오버로딩

(3) 상속

**(오늘의 문제1)**

은행계좌(Account) 클래스 설계

데이터(속성) : 계좌번호(accountNo:String),

예금주(ownerName:String),

잔액(balance:int)

기능(메소드) : 예금하다(void deposit(int)).

인출하다(int withdraw(int)),

잔액조회(int getBalance())

<Account.java>

/\* 은행계좌(Account) 클래스 설계

데이터(속성) : 계좌번호, 예금주, 잔액

기능(메소드) : 예금하다. 인출하다 \*/

**package** com.oop.acc;

**public** **class** Account {

**private** String accoutNo; // 계좌번호

**private** String name; // 예금주

**private** **int** balance; // 잔액

**public** Account() {

System.***out***.print("계좌오픈 감사");

System.***out***.println("계좌번호, 이름이 입력 안 되있음");

}

**public** Account(String accountNo, String name, **int** balance){

**this**.accoutNo = accountNo;

**this**.name = name;

**this**.balance = balance;

System.***out***.print("계좌 오픈 감사합니다 잔액: ");

System.***out***.print(balance+"원");

}

**public** Account(String accountNo, String name){

**this**.accoutNo = accountNo;

**this**.name = name;

balance = 0;

System.***out***.print("계좌 오픈 뭐 억지로 감사합니다 잔액: ");

System.***out***.print(balance+"원");

}

**public** **void** deposit(**int** money){ // 무조건 예금

balance += money;

}

**public** **int** withdraw(**int** money){ // 인출금 return

**if**(balance>=money){

balance = balance - money;

**return** money;

}**else**{

System.***out***.println("잔액이 부족합니다");

**return** 0;

}

}

**public** String getAccoutNo() {**return** accoutNo;}

**public** **void** setAccoutNo(String accoutNo) {**this**.accoutNo = accoutNo;}

**public** String getName() {**return** name;}

**public** **void** setName(String name) {**this**.name = name;}

**public** **int** getBalance() {**return** balance;}

**public** **void** setBalance(**int** balance) {**this**.balance = balance;}

}

<Main.java>

package oop.acc;

public class Main {

public static void main(String[] args){

Account hong = new Account("111-111","홍길동",2100000000);

Account hong1 = new Account(20000);

Account hong2 = new Account();

System.out.println(hong.getBalance()+"홍");

System.out.println(hong1.getBalance()+"홍1");

System.out.println(hong2.getBalance()+"홍2");

long money = hong.withdraw(1000000);

if(money>300000)

System.out.println("뭐할겨요?");

}

}

**(오늘의 문제2)** 직육면체의 가로, 세로, 높이, 부피나 직사각형의 가로, 세로, 넓이를 속성으로 갖는 클래스를 구현하라. 부피나 넓이를 구하는 메소드 calVolume()도 구현한다.

**package** com.oop.boxorrect;

**public** **class** BoxOrRect {

**private** **int** width;

**private** **int** height;

**private** **int** depth;

**private** **int** volume; // 부피나 넓이를 갖음

**public** BoxOrRect() { } // 디폴트 생성자

**public** BoxOrRect(**int** width, **int** height){ // 직사각형

**this**.width = width;

**this**.height = height;

depth = 0;

}

**public** BoxOrRect(**int** width, **int** height, **int** depth){ // 직육면체

**this**.width = width;

**this**.height = height;

**this**.depth = depth;

}

**public** **void** calV(){

**if**(depth==0)

volume = width\*height;

**else**

volume = width\*height\*depth;

// volume = (depth==0)? (width\*height) : (width\*height\*depth);

}

**public** **int** getWidth() {**return** width;}

**public** **void** setWidth(**int** width) {**this**.width = width;}

**public** **int** getHeight() {**return** height;}

**public** **void** setHeight(**int** height) {**this**.height = height;}

**public** **int** getDepth() {**return** depth;}

**public** **void** setDepth(**int** depth) {**this**.depth = depth;}

**public** **int** getVolume() {**return** volume;}

**public** **void** setVolume(**int** volume) {**this**.volume = volume;}

}

**package** com.oop.boxorrect;

**public** **class** MainClass {

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

BoxOrRect box = **new** BoxOrRect(5,6,10);

BoxOrRect rect = **new** BoxOrRect(10,5);

box.calV();

rect.calV();

System.***out***.println("박스 부피 : "+box.getVolume());

System.***out***.println("rect 넓이 : "+rect.getVolume());

}//main

}

프로그래밍이란 결국 **데이터를 효율적으로 관리**하기 위한 TOOL

프로그래밍을 처음 접하는 친구들 또는 개발을 조금해 본 친구들과 같이 일을 하다 보면, 프로젝트 시작과 동시에 이클립스를 열고 열심히 코딩을 시작 합니다. 물론 틀린 행동은 아닙니다. 하지만, 아주 긍정적인 행동도 아닙니다.

프로그래밍이란 버그 없이 돌아가야 합니다. 이것은 당연한 말 입니다.

여기에 우리가 꼭 생각해야 되는 것이 있습니다. 바로 프로그래밍이란 작업은 데이터를 누군가가 효율적으로 활용할 수 있게 시스템을 만드는 행동입니다.

코딩 전 반드시 프로젝트 설계 단계를 꼭 거쳐 효율적으로 프로그램이 돌아가도록

버그 없이 돌아는 가는데, 데이터를 다루는 기술이 부족하여 시스템 속도가 너무 느리다면, 효율적인 시스템이라고 할 수 없을 것 입니다.

우리 개발자들은 코딩 보다 프로그래밍 구조에 대해서 더욱 깊이 생각해야 됩니다.

시간을 내어 **리펙토링** 관련 지식을 접해 보는 것도 많은 도움이 될 것 입니다.