**[ 7 ] 클래스 – 객체지향 프로그램의 기본 코딩방법**

목표(3일간) : 클래스와 객체의 개념을 알고, 구분할 수 있다.

클래스에서 속성, 생성자, 메소드를 활용할 수 있다.

this, static, final의 개념을 알고 구분할 수 있다.

접근 제어자를 통해서 필드, 메소드에 대한 접근범위를 알고 활용할 수 있다.

패키지 개념을 알고 사용할 수 있다.

1. 메소드 프로그램의 한계
   * 메소드를 이용하면 로직의 재사용이 가능하여 개발을 효율적으로 할 수 있었습니다.
   * 절차지향언어에서 중복된 로직을 메소드의 사용으로 인해 효율성을 높일 수 있다.
   * 하지만, 한 문서 내에 메소드의 수가 많아질 경우 추후 유지 보수에 많은 어려움이 발생됩니다.
2. 객체의 개념 및 클래스의 이해

* 객체는 **관련성 있는 데이터와 메소드가 함께 있는 구성체**입니다.
* 예를 들어 자동차 라는 객체가 있습니다.
* 자동차에는 색상, 배기량, 브렌드, 속도 등의 데이터(속성)가 있습니다. 드라이브. 주차. 레이싱이라는 메소드가 있습니다.

(cf) 객체 모델링 : 현실세계나 추상적인 내용의 속성과 동작을 추려내어 소프트웨어 객체의 필드와 메서드로 정의해 나가는 과정

(cf) 객체와 클래스

* 기계를 만들 때 설계를 하고 설계도면대로 기계를 제작하듯이, 객체의 설계도면인 클래스를 만들고 객체를 생성한다..
* 클래스는 객체를 만들기 위한 설계도라고 할 수 있다.

**public** **class** Car {

String color;

**int** cc;

**int** speed;

**void** drive(){

speed = 60;

System.***out***.println("운전한다. 지금 속도 : "+speed);

}

**void** park(){

speed=0;

System.***out***.println("주차한다. 지금 속도:"+speed);

}

**void** race(){

speed=120;

System.***out***.println("레이싱한다. 지금 속도 : "+speed);

}

}

* 객체는 클래스로부터 메모리에 생성(복사본)됩니다(클래스를 정의하고 클래스로부터 객체가 생성되어진다). Car myPorsche = new Car();
* 이때 myPorsche 자동차를 만들어 놓고, 설계대로(클래스정의대로) drive를 하기도 park를 하기도 race를 하기도 합니다.

myPorsche.drive();

myPorsche.park();

myPorsche.race();

* 즉 드라이브를 하거나 주차를 하거나 레이스를 하기 전에 드라이브나 주차나 레이스를 하도록 설계(정의)된 Car가 클래스이며, 설계한대로 생성되어 드라이브를 하거나 주차를 하거나 레이싱을 하는 완전한 자동차(객체)가 만들어 집니다.
* 프로그램에서의 객체 = 데이터+메소드

Car yourPorsche = new Car();

또는

Car yourPorsche;

yourPorsche = new Car();

* + - **myPorsche와 yourPorsche는 전혀 관계가 없는 독립된 객체입니다.**
    - **myPorsche의 color와 yourPorsche의 color는 전혀 별개이다.**
    - 클래스를 선언한 다음, main() 메서드가 있는 클래스에서 객체를 아래의 형식으로 생성할 수 있다.

**public** **class** CarMain {

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

Car myPorsche = **new** Car();

myPorsche.drive();

myPorsche.park();

myPorsche.race();

Car yourPorsche;

yourPorsche = **new** Car();

myPorsche.color = "red";

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

}

}

1. 객체 자료형과 변수와의 관계
   * 객체 데이터 변수는 값이 아닌 참조값을 저장한다.

(cf) int value = 10; // value 에는 값이 저장

(cf) ClasName classObject = new ClasName(); // classObject에는 값이 아닌 참조값이 저장

(변수명)

**(ex1)** <Car.java>

**public** **class** Car {

**private** String color;

**private** **int** cc;

**private** String brand;

**private** **int** speed;

**public** **void** drive() {

speed = 60;

System.***out***.println("운전한다. 지금 속도 : "+speed);

}

**public** **void** park() {

speed = 0;

System.***out***.println("주차한다. 지금 속도 : "+speed);

}

**public** **void** race() {

speed = 120;

System.***out***.println("레이싱 한다. 지금 속도 : "+speed);

}

**public** **void** setColor(String col) { color = col; }

**public** String getColor() { **return** color; }

**public** **void** setCc(**int** c){cc = c;}

**public** **void** setBrand(String br){brand = br;}

**public** **void** setSpeed(**int** s){speed = s;}

**public** **int** getCc() {**return** cc;}

**public** String getBrand(){**return** brand;}

**public** **int** getSpeed() {**return** speed;}

}

<CarMain.java>

**public** **class** CarMain {

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

Car myPorsche = **new** Car();

Car yourPorsche = **new** Car();

myPorsche.drive();

myPorsche.setSpeed(90);

System.***out***.println("myPorsche속도 : "+myPorsche.getSpeed());

yourPorsche.race();

yourPorsche.setCc(1000);

System.***out***.print("yourPorsche CC : "+yourPorsche.getCc()+"\t");

myPorsche.setCc(2400);

System.***out***.print("myPorsche CC : "+myPorsche.getCc()+"\t");

}

}

**(ex2)**

<Rect.java>

**public** **class** Rect {

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

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

**public** **int** area() {

**return** width\*height;

}

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

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

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

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

}

<RectMain.java>

**public** **class** RectMain {

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

Rect r1 = **new** Rect();

Rect r2 = **new** Rect();

r1.setWidth(10);

r1.setHeight(5);

System.***out***.println(r1.area());

System.***out***.println(r2.area());

}

}

1. 클래스 제작(ExClass.java)

(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

|  |
| --- |
| **package com.lec.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. 생성자함수 : 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**
7. 생성자의 이해 : 생성자는 매개변수 있는 생성자와 매개변수 없는 생성자 등 여러 종류의 생성자를 가질 수 있다. 생성자가 없을 때는 디폴트 생성자가 컴파일러 단계에서 자동 생성한다. 한 개 이상의 생성자가 있으면 디폴트 생성자는 자동 생성되지 않는다.

**(ex3) package** com.lec.ex2\_human;

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

**private** String name;

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

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

**private** **double** weight;

**public** Man() {

System.***out***.println("매개변수 없는 초기화 함수 호출");

}

// 파라미터값을 갖는 생성자 : 개발자가 생성자 함수를 만들면 컴파일러가 디폴트 생성자 안 만들어요

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

System.***out***.println("데이터 4개 초기화 생성자 함수 호출");

**this**.name = name;

**this**.age = age;

**this**.height = height;

**this**.weight = weight;

}

**public** Man(String name, **int** height, **double** weight) {

System.***out***.println("키, 몸무게 초기화 생성자 함수");

**this**.name = name;

**this**.height = height;

**this**.weight = weight;

}

**public** Man(String name) {

System.***out***.println("이름 초기화 생성자 함수 호출");

**this**.name = name;

}

//파라미터가 달랑 하나인 생성자

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

System.***out***.println("키 초기화 생성자 함수");

**this**.height = height;

}

// 파라미터가 실수형인 생성자

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

System.***out***.println("몸무게 초기화 생성자 함수");

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

}

**public** **double** calculateBMI() {

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

**return** result;

}

// setter

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

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

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

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

// getter

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

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

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

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

}

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

**public** Woman(){

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

}

}

**package** com.lec.ex2\_man.kid;

**public** **class** Kid {

**private** String name;

**public** Kid(String name) {

**this**.name = name;

System.***out***.println("매개변수 있는 Kid 생성자");

}

}

**import** com.lec.ex2\_human.\*;

**import** com.lec.ex2\_human.kid.Kid;

**public** **class** humanMain {

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

Woman woman1 = **new** Woman();

Woman woman2 = **new** Woman();

// ------------ ------------

// ①레퍼런스변수선언 ②인스턴스생성

System.***out***.println("woman1과 woman2가 같은가? " + (woman1==woman2));

System.***out***.println("woman1과 woman2가 같은가? " + woman1.equals(woman2));

// Kid kid = new Kid(); 매개변수 있는 생성자 함수가 있을 경우 디폴트 생성자 생성 안 함

Kid kid = **new** Kid("홍아기");

Man hong = **new** Man("홍길동");

Man kim = **new** Man("김길동", 180, 70.5);

Man kim2;

kim2 = kim;

kim.setAge(185);

System.***out***.println("hong과 kim이 같은 객체인지 : " + hong.equals(kim));

System.***out***.println("kim과 kim2이 같은 객체인지 : " + kim2.equals(kim));

**double** bmi = kim.calculateBMI();

System.***out***.println(kim.getName() + " : "+ kim.getHeight() +"," +kim.getWeight());

System.***out***.println("kim의 bmi지수 : " + bmi);

}

}

클래스 객체

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

설계도 제품

객체 ≒ 인스턴스

3. 객체지향 프로그래밍의 특징

(1) 캡슐화와 데이터은닉

(2) 다형성 : 같은 모습이지만 다른 기능 cf.오버로딩

(3) 상속

**(ex4)** 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 \*/

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

**private** String accountNo; // 계좌번호

**private** String name; // 예금주

**private** **long** 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**.accountNo = 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**.accountNo = accountNo;

**this**.name = name;

balance = 0; // 안해도 됨

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

System.***out***.print(balance+"원");

}

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

balance += money;

}

**public** **void** withdraw(**int** money) {// 잔액이 있을 경우만 인출

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

balance -= money;

}**else** {

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

}

}

**public** String getAccountNo() {**return** accountNo;}

**public** **void** setAccountNo(String accountNo) {**this**.accountNo = accountNo;}

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

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

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

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

}

**public** **class** AccountMain {

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

Account hong = **new** Account("111-111","홍길동",2200000000L);

//Account hong1 = new Account(20000);

Account hong1 = **new** Account();

System.***out***.println("hong 계좌정보 : " + hong.getAccountNo() + ", " + hong.getName());

System.***out***.println("hong1 계좌정보 : " + hong1.getAccountNo() + ", " + hong1.getName());

System.***out***.println("hong 잔액 : " + hong.getBalance());

System.***out***.println("hong1 잔액 : " + hong1.getBalance());

hong1.setAccountNo("222-222");

hong1.setName("신길동");

hong.withdraw(1000);

hong1.deposit(1000);

hong1.withdraw(2000);

}

}

**(ex5)** name, age, gender를 데이터로 갖고, 데이터 정보를 출력하는 print() 메소더를 갖는 PersonInfo 클래스를 구현하고 main함수를 이용하여 test 구현하시오(print() 메소드 실행 결과 : 이름=홍길동, 나이=20, 성별=m, main 메소드 안에는 PersonInfo형 객체 인스턴스를 배열로 구현해 봅니다)

**public** **class** PersonInfo {

**private** String name;

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

**private** **char** gender;

**public** PersonInfo() { }

**public** PersonInfo(String name, **int** age, **char** gender) {

**this**.name = name;

**this**.age = age;

**this**.gender = gender;

}

**public** **void** print() {//이름 = 홍길동 나이 = 20 성별 = m

System.***out***.println("이름 = "+name+"\t나이 = "+age+"\t성별"+gender);

}

}

**public** **class** PersonInfoTestMain {

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

PersonInfo[] person = {**new** PersonInfo("홍길동", 20, 'm'),

**new** PersonInfo("홍길순",19,'f')

};

**for**(PersonInfo p : person) {

p.print();

}

**for**(**int** idx=0 ; idx<person.length ; idx++) {

person[idx].print();

}

PersonInfo[] iperson = **new** PersonInfo[2];

iperson[0] = **new** PersonInfo("홍길동", 20, 'm');

iperson[1] = **new** PersonInfo("홍길순",19,'f');

**for**(PersonInfo p : iperson) {

p.print();

}

}

}

**(quiz) Member 클래스를 구현하고 테스트하시고**

데이터(속성) : id, pw, name, email, address, birth, gender

메소드 : “아이디 = aaa

이름 = 홍길동

이메일 = hong@company.com

주소 = 서울 강남구

생일 = 2000-01-01

성별 = 남 ”(객체 정보)을 출력하는 print() 메소드

main()함수 내용 :

public static void main(String[] args) {

Member member1 = new Member("aaa", "xxx", "홍길동", "hong@company.com",

"서울시 강남구", "2000-01-01", 'M');

Member member2 = new Member("bbb", "xxx", "신길동", "shin@company.com",

"서울시 서대문구 신촌로 141", "1999-01-02", 'F');

member1.print();

member2.print();

}