**14회차(1)**

package staticc;

public class Student {

// 지역변수 vs 객체변수 : 변수를 어디에서 정의하느냐

// 기본적으로 자바는 클래스 외부에서는 변수 선언 혹은 메소드 정의가 불가능

// 무조건 클래스 내부에서만 가능!

// 메소드가 아닌 곳에서 변수가 정의되면 "객체변수" 가 된다

// 객체변수 : 속성

int kor;

int eng;

int math;

String name;

int tot;

double avg;

// 생성자(특별한 메소드)

// return 리턴을 하지 않기 때문에, 리턴 타입을 명시할 필요가 없다

// 직접 호출하지 않는다 (객체가 생성될 때 자동으로 호출)

// 생성자의 이름 = 클래스의 이름 과 동일

// 생성자도 매개변수를 가질 수 있다

Student(int kor, int eng, int math, String name) {

// 국영수 점수와 이름을 전달 받아 객체의 속성을 초기화할 수 있도록

// 메소드 내의 변수들도 지역변수!

// 지역변수와 객체변수의 이름이 동일한 경우에는 지역변수가 우선!

// => 객체변수를 참조할 수 없다

// => this.객체변수 = 지역변수;

this.kor = kor;

this.eng = eng;

this.math = math;

this.name = name;

this.tot = kor + eng + math;

this.avg = this.tot / 3.0;

}

// 해당 객체의 정보를 출력하는 메소드

void info() {

System.out.println("======" + this.name + "======");

System.out.println("국어 점수 : " + this.kor);

System.out.println("영어 점수 : " + this.eng);

System.out.println("수학 점수 : " + this.math);

System.out.println("총점 : " + this.tot);

System.out.println("평균 : " + this.avg);

}

public static void main(String[] args) {

// 정의된 생성자에 맞춰서 매개변수를 입력

Student stu1 = new Student(60, 90, 40, "홍길동");

stu1.info();

// 학생은 얼마든지 더 추가될 수 있다

Student stu2 = new Student(90, 80, 20, "원빈");

stu2.info();

}

}

**14회차(2)**

package staticc;

public class Student {

// static 앞에 붙이기 -> 클래스변수

static int num\_of\_student;

// 클래스 내에서 메소드를 호출할 수는 없다!

// System.out.println("여기서는 불가능");

// 객체변수 : 객체별로 고유한 속성값을 가지게 된다!

int kor;

int eng;

int math;

String name;

int tot;

double avg;

// 생성자(특별한 메소드)

Student(int kor, int eng, int math, String name) {

this.kor = kor;

this.eng = eng;

this.math = math;

this.name = name;

this.tot = kor + eng + math;

this.avg = this.tot / 3.0;

// 생성자가 호출될 때마다 num\_of\_student의 값을 1씩 증가

// 아래의 변수는 클래스 변수인가? 지역 변수인가? 객체 변수인가?

// 객체변수의 표현은 앞에 this를 붙여 this.객체변수 인 것처럼

// 클래스변수는 앞에 클래스 이름을 붙여서 명확하게 표현

// Student.num\_of\_student;

num\_of\_student++;

}

// 해당 객체의 정보를 출력하는 메소드

void info() {

System.out.println("======" + this.name + "======");

System.out.println("국어 점수 : " + this.kor);

System.out.println("영어 점수 : " + this.eng);

System.out.println("수학 점수 : " + this.math);

System.out.println("총점 : " + this.tot);

System.out.println("평균 : " + this.avg);

}

public static void main(String[] args) {

// 정의된 생성자에 맞춰서 매개변수를 입력

Student stu1 = new Student(60, 90, 40, "홍길동");

stu1.info();

Student stu2 = new Student(90, 80, 20, "원빈");

stu2.info();

// 객체를 생성한 이후에 학생 수를 확인해보면

System.out.println("현재 학생 수는 "+Student.num\_of\_student + "명 입니다");

}

}

**14회차(3)**

package staticc;

// static 앞에 붙이기 -> 클래스변수

static int num\_of\_student;

// 클래스변수 num\_of\_student를 참조하는 메소드 정의

void get\_num\_of\_student() {

System.out.println("현재 학생 수는 "+Student.num\_of\_student + "명 입니다");

}

int kor;

int eng;

int math;

String name;

int tot;

double avg;

Student(int kor, int eng, int math, String name) {

this.kor = kor;

this.eng = eng;

this.math = math;

this.name = name;

this.tot = kor + eng + math;

this.avg = this.tot / 3.0;

// Student.num\_of\_student;

num\_of\_student++;

}

// 해당 객체의 정보를 출력하는 메소드

void info() {

System.out.println("======" + this.name + "======");

System.out.println("국어 점수 : " + this.kor);

System.out.println("영어 점수 : " + this.eng);

System.out.println("수학 점수 : " + this.math);

System.out.println("총점 : " + this.tot);

System.out.println("평균 : " + this.avg);

}

public static void main(String[] args) {

// 객체를 생성하기 전에 학생 수를 확인하고 싶다면? -> 확인할 방법이 없다..

// 클래스 메소드는 클래스 이름을 통해서 객체가 없어도 호출 가능

// Student.get\_num\_of\_student();

Student stu1 = new Student(60, 90, 40, "홍길동");

stu1.info();

Student stu2 = new Student(90, 80, 20, "원빈");

stu2.info();

// 객체를 생성한 이후에 학생 수를 확인해보면

System.out.println("현재 학생 수는 "+Student.num\_of\_student + "명 입니다");

// 정의된 메소드를 통해서 학생 수를 확인

stu1.get\_num\_of\_student(); // <- 모순 : 객체를 생성하기 전 학생 수 확인 불가능 <-> but 클래스 변수

}

}

**14회차(4)**

// 두 파일을 만들거나 다른 클래스에 객체변수를 아래와 같이 선언한다면

// 메인함수에서 변수 값 변경 불가능!

// 은닉성 private

// private int kor;

// private int eng;

// private int math;

// private String name;

// private int tot;

// private double avg;

// 수학점수를 고칠 수 있는 메소드를 정의

// void set\_math(int math){

// 이렇게 메소드를 이용해서 접근하도록 하는 가장 큰 이유는

// 속성에 직접적으로 참조하지 않고 우회해서 참조할 수 있는 방법을 제공

// 그래야 속성을 안전하게 보호

// 예를 들면 수학점수를 고칠 수 있는 권한이 있는 사용자인가? (권한체크)

// 올바른 수학점수인가>

// 등을 메소드 내에서 점검한 이후에 참조할 수 있게 되므로 안전하게 속성을 보호

// this.math = math;

// }

// 후 메인함수에서

stu1.set\_math(100); stu1.info();

**14회차(6)**

1. 객체변수(인스턴스 변수)

- 각 객체별로 고유한 값을 가지게 된다

- 객체 간의 속성(객체변수)는 서로 공유되지 않는다

<-> 클래스변수[공유변수]

2. 클래스변수(정적 변수)

- 변수를 선언할 때, static 키워드를 붙여준다

- 클래스 변수는 모든 객체가 동일한 값을 가지게 된다

- 객체 없이도 참조가 가능하다 <- 클래스 이름으로 참조 가능

3. static method vs instance method

1) instance method

- 생성자를 제외한 지금까지 정의된 메소드는 전부 인스턴스 메소드

- 객체를 통해서 호출

2) static method(클래스 메소드)

- 클래스변수와 비슷하게 객체가 아닌, 클래스를 통해서 호출할 수 있는 메소드

- 객체가 없어도 호출할 수 있다

- 예를 들면, 클래스 변수를 참조하는 메소드가 있다면?

` - 클래스 변수는 객체 없이도 참조 가능한데

- 메소드를 호출하려면 반드시 객체가 있어야만 가능

4. 왜 main은 static으로 선언이 되어야만 하는가?

- 만약에 main 메소드가 static이 아니라면?

- 메인 클래스(main 메소드를 포함하는 클래스)에 대한 객체를 생성해야만 호출할 수 있을 것

(- Main main = new Main(); 코드를 작성하고 실행해줘야 한다)

- 만약에 그렇다면, 메인 클래스에 대한 객체를 생성하는 코드를 실행할 수 있는

또다른 객체를 생성해야 할 것이고, 이는 무한히 많이 발생하게 되는 오류 발생시킴

- 그렇기 때문에, 객체가 없어도 호출될 수 있는 형태로 main 메소드를 작성해야함

5. 객체지향 프로그래밍의 특징

- 클래스란?

- 하나의 타입 정도로 이해

- 비슷한 성질의 변수(속성)의 기능(메소드)을 하나의 블록으로 관리하기 위한 개발 방법론

- 이렇게 하나의 블록으로 묶어 놓으면 소스코드를 관리하기가 편해지는 장점

- 객체란?

- 변수의 더 확장된 개념

- 일반적으로 값을 담아두기만 하지 않고, 그 값을 제어할 수 있는 기능도 같이 제공

6. 그렇다보니 추가적으로 여러가지 특징들이 발생

# 은닉성(캡슐화) : 클래스 내의 속성, 기능을 보호하기 위한 장치

- 클래스 내에서 정의된 속성과 기능에 대해서 외부에서 함부로

접근하지 못하도록 비공개로 하는 것

- 객체지향에서 클래스는 제공되는 기능들이 어떻게 동작하는 지 알 필요 없다고 얘기한다

- 전자레인지, 밥솥과 같은 경우로 비유 -> 외부에 노출된 기능(ex 타이머)만 알고 있고 내부의 과정은 궁금해하지 않음

- 외부에서는 클래스에서 정의된 속성이나 기능을 전부 알 수 없도록 한다

- 딱 필요한만큼만 공개하고 나머지는 몰라도 사용

- 일반적으로 객체지향 프로그래밍에서는 모든 속성은 비공개로 하는 것을 원칙으로 한다

- 외부에서 접근이 필요한 경우가 있다면, 메소드를 통해서 접근할 수 있도록 제공해주는 것이 좋다

\* 은닉성은 어떻게 제공되는가?

- 접근제어자(Access Modifier)

- 접근제어자를 통해서 속성이나 기능에 접근할 수 있는 레벨 설정

(1) private : 동일 클래스 내에서만 참조 가능 (제한 제일 높음) -> 상속도 불가능

(2) default : 동일 패키지 내에서는 참조 가능

(3) protected : 클래스를 상속받을 외부 패키지의 클래스까지 가능

(4) public : 참조에 제한이 없음 (제한 제일 낮음)

# 상속(inherit) : 클래스의 재사용성을 더 확대한 개념

-> 클래스를 확장해서 계속 사용하자!

# 다형성(Polymorphism) : 하나의 객체가 여러 가지 타입을 가질 수 있는 것

- 객체의 다형성

- 메소드의 다형성

**14회차(7)**

package staticc;

//휴대전화를 모델로 해서 클래스를 작성하는 실습

// - main.java : main() 메소드를 가지고 있는 클래스

// - moblie.java : 휴대전화 클래스

// - 속성, 이름, 통신사, 가격

// - 기능 : 속성을 출력

// - 생성자는 알아서

// 이때, 모든 속성은 private로 한다

// 각 속성을 참조할 수 있는 getter와 setter 정의

class mobile{

private String name;

private String store;

private int price;

// 생성자 만들기

mobile(String name, String store, int price){

this.name = name;

this.store = store;

this.price = price;

}

void info() {

System.out.println("이름 : "+name);

System.out.println("통신사 : "+store);

System.out.println("가격 : "+price);

}

}

public class Main {

public static void main(String[] args) {

mobile phone = new mobile("홍길동", "KT", 300000);

// Main이 아닌, mobile에 관한 객체를 생성해야 한다

phone.info();

}

}