05/26 목

Public static ~~~~~~//

Public – 접근 제한자 위치(public은 모두가 사용가능)

Static – 정적 제한자 → 언제나 존재. 언제든 호출하여 사용가능.

Today; 요약목록

메소드 Method;

메소드 헤드 : 반환형 이름 (파라미터 목록)

→ 메소드 이름 규칙

메소드 바디 : { ~~문장~~ }

→ 메소드에서 값 반환하기 (return)

메소드 호출

→ 호출 시의 값 전달방식

-Method 만들어 호출-

참고 : TestMethod.java

**‘method’ → ‘동작’에 이름이 붙은 것.**

**Method 는 '기계’**

프로그램의 시작은 언제나 ‘main’ method;

public class TestMethod {

public static void printRepeat() {// printRepeat 부분은 method의 '이름'부분 →→→→ **메소드 머리 (head)**

// 메소드 몸통(body) – 기능 이름, 동작

for (int i = 0; i < 3; i++) { // <<<< 메소드 만들기.

System.out.println("반복 중");

// 프로그램의 시작은 언제나 main 메소드로 시작하기떄문.

} // 처음엔 인식되지 않고 넘어감.

}

public static void main(String[] args) {

System.out.println("프로그램 시작");

printRepeat(); // 메소드 호출

// 위의 printRepeat() 메소드 중괄호 까지 인식후, 다시 제자리로 돌아와서 흐름이 내려간다.

System.out.println("프로그램 끝~");

}

}

// 메소드는 항상 몸통 {} 중괄호 를 포함해야한다. 비어있더라도 적어줘야함

메소드 이름 관련;

// 메소드의 이름 == 영소문자; 여러 단어가 있으면 다음 단어의 첫번째 글자 -> 대문자 / 동사로 많이 작성

// 이름을 다르게 해도 컴파일은 되나, 통용되는 방법은 위의 방법으로 이름 작성

// 메소드 이름과 변수 이름이 같아도 되긴함 -> 메소드에는 () 가 항상 붙어있기 때문.... but 읽을 때 헷갈릴 수 있음.

public static void sum() {

int a = 10;

int b = 20;

int sum = a + b;

}

// return type : void (없음, 공허);

// return type 이 void 다 = 없는 것을 반환한다 = 반환이 없다.

Public static void method() {} → void 자리는 return 타입

public static int sum() { // return type : void (없음, 공허);

int a = 10;

int b = 20;

int sum = a + b;

return sum; // 값을 반환(뱉어냄) 해줌. 여기서 sum 값은 30; 30이라는 값을 가지고 main메소드의 sum(); 으로 이동.

}

public static void main(String[] args) {

System.out.println("시작");

int result = sum(); // result에 초기화

System.out.println(result);

System.out.println("끝");

}

// pi 메소드 { 3.14 실수값을 반환하는 메소드}

public static double pi() { // return 하고자 한다면 -> void는 절대 X. 아주 엄격함.

return 3.14;

}

double pi = pi();

System.out.println(pi);

→ main method에 입력해주면 출력 가능

--- 입력 받은 수를 method에 집어넣기 ---

참고 : TestMethod3.java

public class TestMethod3 { **// parameters == 개수, 타입, 순서**

public static int getSum(int a, int b) { // () 부분 = parameters. 메소드의 입력부분

return a + b; // return 이 나가는 부분이면, ()부분은 들어오는 입력 부분.

}

public static void main(String[] args) {

int sum = getSum(30, 60);

System.out.println(sum);

// System.out.println(a); // parameters도 당연히 지역변수. => 컴파일 에러

System.out.println(getSum(100,200));

* 큰 값 비교하기 -

참고 : TestMax.java

public class TestMax {

public static int max(int left, int right) {

return left > right ? left : right;

}

public static void main(String[] args) {

// 3개 정수 중 가장 큰 값 구하기

int a = 3;

int b = 4;

int c = 5;

// 4개

int d = 6;

// 5개

int e = 7;

System.out.println(max(e, max(d, max(c, max(a, b)))));

}

}

* 메소드 이용해서 점수 별 출력, 총 합, 평균 구하기 -

참고 : TestMethod4.java

기본 문장 (if else 등) 사용하여 하나 씩 문장 만들기

총합, 평균 계산 메소드 제작

While문, For문으로 잘 못 입력된 값 반복시키는 반복문 만들기

결과 확인

* 메소드 규칙 -

참고 : Methods.java

Parameters 는 메소드 안에서 하나의 박스를 새로 만든다.

→ 지역변수 값이 됨. 값만 땡겨오는 것.

결국 메소드 밖 {} 를 나가게 되면 그 변수는 사라짐.

public class Methods {

public static void increase(int a) {

a++;

}

public static void main(String[] args) {

int a = 10;

increase(a); // Call By Value

System.out.println(a);

}

}

출력 값 :10

→ 출력 값을 바꾸기 위해선 return 값을 주어야함.

public class Methods {

public static **int** increase(int a) {

a++;

**return a;**

}

public static void main(String[] args) {

int a = 10;

**a =** increase(a);

System.out.println(a);

System.out.println(increase(a)); → 출력은 11;

**→ 하지만 main 의 a 값은 10 그대로임.**

}

}

★ method는 return값을 1개만 가진다.

// 참조형 변수에서는 달라짐

method는 return값을 1개만 가진다. 참조형 변수에서는 달라짐

public static int,int createTwoNumber() {

// return 10,20;

} // 컴파일 에러

Head부분과 반환의 값 일치시키기

public static boolean test() {

return 10;

}

→ boolean return 값 타입 일치 하지않음 – 컴파일 에러;

참조형 타입의 메소드

참고 : MethodsRefParam.java

참조형 타입 String 사용

public class MethodsRefParam {

public static String concat(String left, String right) {

return left + right;

}

public static void main(String[] args) {

String h = "Hello";

String w = "World";

String result = concat(h, w);

System.out.println(result);

}

}

출력 : HelloWorld

→ String left 는 String h의 Hello를 참조, String right 는 String w의 World를 참조.