**[ 6 ] 객체 지향 프로그램의 시작에 앞서(메소드)**

목표 : 객체지향프로그램이 나오기 전 효율적인 프로그래밍을 위해 등장한 메소드

1. 객체지향 이전의 프로그래밍

<절차지향>

- 지금까지 배운 내용으로 1부터 10까지 정수의 합을 구하고

- 그 결과값이 짝수인지 홀수 인지를 파악하고,

- 홀수라면, "홀수입니다"를 출력하고 짝수라면 "짝수입니다"를 출력

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

**int** tot=0;

**for**(**int** i=1; i<11; i++){

tot += i;

}

System.***out***.println("1~10까지 정수합은 "+tot+"입니다");

System.***out***.println((tot%2==0)? "짝수입니다":"홀수입니다");

tot=0;

**for**(**int** i=10; i<100; i++){

tot += i;

}

System.***out***.println("10~100까지 정수합은 "+tot+"입니다");

System.***out***.println((tot%2==0)? "짝수입니다":"홀수입니다");

}

- 위에서 살펴본 프로그래밍 방식이 절차지향 언어 입니다.

- 위에서부터 순차적으로 실행되는 것이죠.

- 절차지향 방식을 이용하면서 개발자들은 몇 가지 비효율적인 면을 느끼게 됩니다.

① 기존의 1~10까지의 합을 구하는 것을 10~100까지로 변경하게 되면 동일한 작업이지만 다시 코드를 수정해야 됩니다.

② 기존의 로직을 다시 사용하게 될 때, 다시 코딩을 하는 방법과, 기존 코드를 카피해서 재사용 하는 방식이 있으나, 두 방법 모두 비효율적.

③ 두번째 방식으로 코딩을 하면 동일한 로직의 코드가 문서 내에 넘쳐나, 코드의 양이 너무 길어 집니다. 추후에 유지보수가 어렵습니다.

- 하여, 등장한 방식이 함수 또는 **메소드(method)**입니다.

- 로직만 만들어 놓고, 그때 그때 데이터를 주면 메소드가 알아서 결과값을 반환하는 방식 입니다.

- 위의 코드를 수정해 보도록 하겠습니다.

public static void main(String[] args) {

int tot = sum(1,100);

System.out.println(evenOdd(tot));

}

private static int sum(int from, int to){ from to

int result = 0;

**↓**

**↓**

for(int i=from ; i<=to ; i++){

result += i;

}

return result;

} int **result**

private static String evenOdd(int value){

String result = new String(); **value**

**↓**

if(value%2 == 0) {

result = "짝수입니다";

} else {

result = "홀수입니다";

}

return result; String **result**

}

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

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

Ex02 ex02 = **new** Ex02();

**int** sum =ex02.sum(1,0);

System.***out***.println("1~10까지 정수합은 "+sum+"입니다");

System.***out***.println(ex02.evenOdd(sum));

sum = ex02.sum(1,101);

System.***out***.println("1~101까지 정수합은 "+sum+"입니다");

}

**private** String evenOdd(**int** value) {

String result = **null**;

**if**(value%2==0)

result = "짝수입니다";

**else**

result = "홀수입니다";

**return** result;

}

**private** **int** sum(**int** from, **int** to) {

**int** result = 0;

**for**(**int** i=from ; i<=to ; i++) {

result += i;

}

**return** result;

}

}

2. 객체지향의 등장 배경과 이해

- 메소드를 이용한 방식으로 개발자들은 아주 많은 효율성을 느끼게 되어 있습니다.

- 하지만, 메소드를 이용한 방식에도 한계는 있었습니다.

- 데이터가 많아지고, 메소드가 많아짐에 따라 코드의 양이 많이 지고, 어려워지기 시작했습니다.

- 하여, 등장한 방식이 객체지향 프로그래밍 입니다.

- 객체란 동일한 성질의 데이터와 메소드를 한곳에 모아두고 필요한 곳에서 언제든지 이용할 수 있게 만들어 놓은 덩어리 입니다.

**public** **class** Arithmetic {

**public** String evenOdd(**int** value) {

String result = **null**;

**if**(value%2==0)

result = "짝수입니다";

**else**

result = "홀수입니다";

**return** result;

}

**public** **int** sum(**int** from, **int** to) {

**int** result = 0;

**for**(**int** i=from ; i<=to ; i++) {

result += i;

}

**return** result;

}

**public** **int** abs(**int** su) {

**int** result = (su<0)? -su : su;

**return** result;

}

}

**public** **class** ExMain {

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

Arithmetic ar = **new** Arithmetic();

**int** sum =ar.sum(1, 10);

System.***out***.println("1~10까지 정수합은 "+sum+"입니다");

System.***out***.println(ar.evenOdd(sum));

System.***out***.println("-5의 절대값은 "+ar.abs(-5)+"입니다");

}

}

3. 메소드의 이해

메소드란? 작업을 수행하기 위한 명령문의 집합

어떤 값을 입력받아서 처리하고 그 결과를 돌려준다(입력 받는 값이 없을 수도 있고 결과를 돌려주지 않을 수도 있다.)

메소드의 장점과 작성지침 ; 반복적으로 수행되는 여러 문장을 메소드로 작성한다.

**접근제한자 [static] 리턴type 메소드명([매개변수1, 매개변수2,..]){**

**//return이 없을 경우 리턴type은 void**

**처리할 프로세스들**

**[return 리턴값;]**

**}**

1. 매개변수를 갖지 않는 메소드

**public** **class** method1 {

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

*printCh*();

System.***out***.println("Hello, Java");

*printCh*();

System.***out***.println("Hello, Java");

*printCh*();

System.***out***.println("Hello, Java");

*printCh*();

}

pravate **static** **void** printCh(){

**for**(**int** cnt=1 ; cnt<=20 ; cnt++)

System.***out***.print('\*');

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

}

}

1. 매개변수를 갖는 메소드

**public** **class** method1 {

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

*printCh*('■',15);

System.***out***.println("Hello, Java");

*printCh*('\*',12);

System.***out***.println("Hello, Java");

*printCh*('\*',12);

System.***out***.println("Hello, Java");

*printCh*('■',15);

}

**static** **void** printCh(**char** ch, **int** i){

**for**(**int** cnt=1 ; cnt<=i ; cnt++)

System.***out***.print(ch);

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

}

}

1. Return값을 갖는 함수

**public** **class** method1 {

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

Scanner sc = **new** Scanner(System.***in***);

**int** input = sc.nextInt();

**int** inputAbs = *abs*(input);

System.***out***.printf("입력하신수 %d의 절대값은 %d", input, inputAbs);

}

**static** **int** abs(**int** in){

**int** result = (in<0)? -in : in;

**return** result;

}

}

1. 메소드의 재귀적 호출

factorial 계산을 위한 메소드 구현 소스

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

Scanner sc = **new** Scanner(System.***in***);

**int** su = sc.nextInt();

**int** result = *factorial*(su);

System.***out***.println(su+"! = "+result);

}

**private** **static** **int** factorial(**int** s){

// s\*(s-1)\*(s-2)\*(s-3)\*......2\*1

**int** fact = 1;

**for**(**int** i=s ; i>=1 ; i--)

fact = fact\*i;

**return** fact;

}

factorial 계산을 위한 재귀적 메소드 호출 소스

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

Scanner sc = **new** Scanner(System.***in***);

**int** su;

**do**{

System.***out***.print("양수를 입력하세요 ☞ ");

su = sc.nextInt();

}**while**(su<=0);

System.***out***.println(su+"! = "+*factorial*(su));

sc.close();

}

**private** **static** **int** factorial(**int** s){

// 3! = 3\*2! -> s! = s\*(s-1)! 재귀적호출

//return (s==1)? 1:s\*factorial(s-1);

**if**(s==1)

**return** 1;

**else**

**return** s \* *factorial*(s-1);

}

4. 오늘의 문제I : 사용자로부터 원하는 단수(2~9) 구구단을 출력하는 프로그램을 구현하시오.

(단, 단수를 매개변수로 받아 해당 단수의 구구단을 출력하는 부분을 method로 처리한다. 사용자가 2~9 사이의 수가 아닌 수를 입력할 시 2~9사이의 수를 입력할 때까지 계속 입력 받는다.)

[yisy0703@naver.com](mailto:yisy0703@naver.com) – method(int dansu) / do~while문이나 while문 이용

4. 오늘의 문제 II : 사용자로부터 수를 입력받아 절대값을 출력하는 로직을 구현하시오 (단, 매개변수를 받아 절대값을 리턴하는 함수=메소드를 이용합니다)