

능동적 사고 방식의

java

강사 박주병



Part06 객체지향

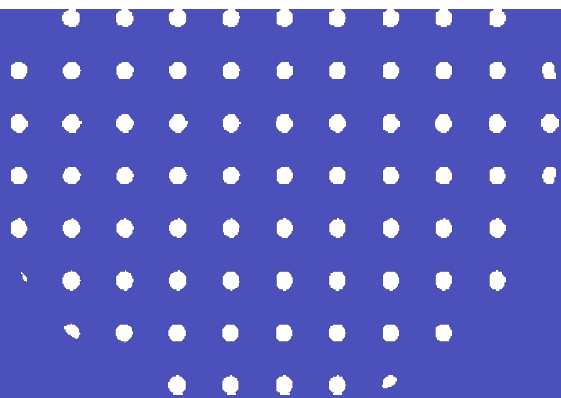
01 객체지향의 이해

02 메서드

03 메서드 활용

04 실습 문제





ParkJuByeong

01

객체지향의 이해



객체 = 실제로 존재하는것



데이터 + 기능

ParkJubyeong



객체



데이터:

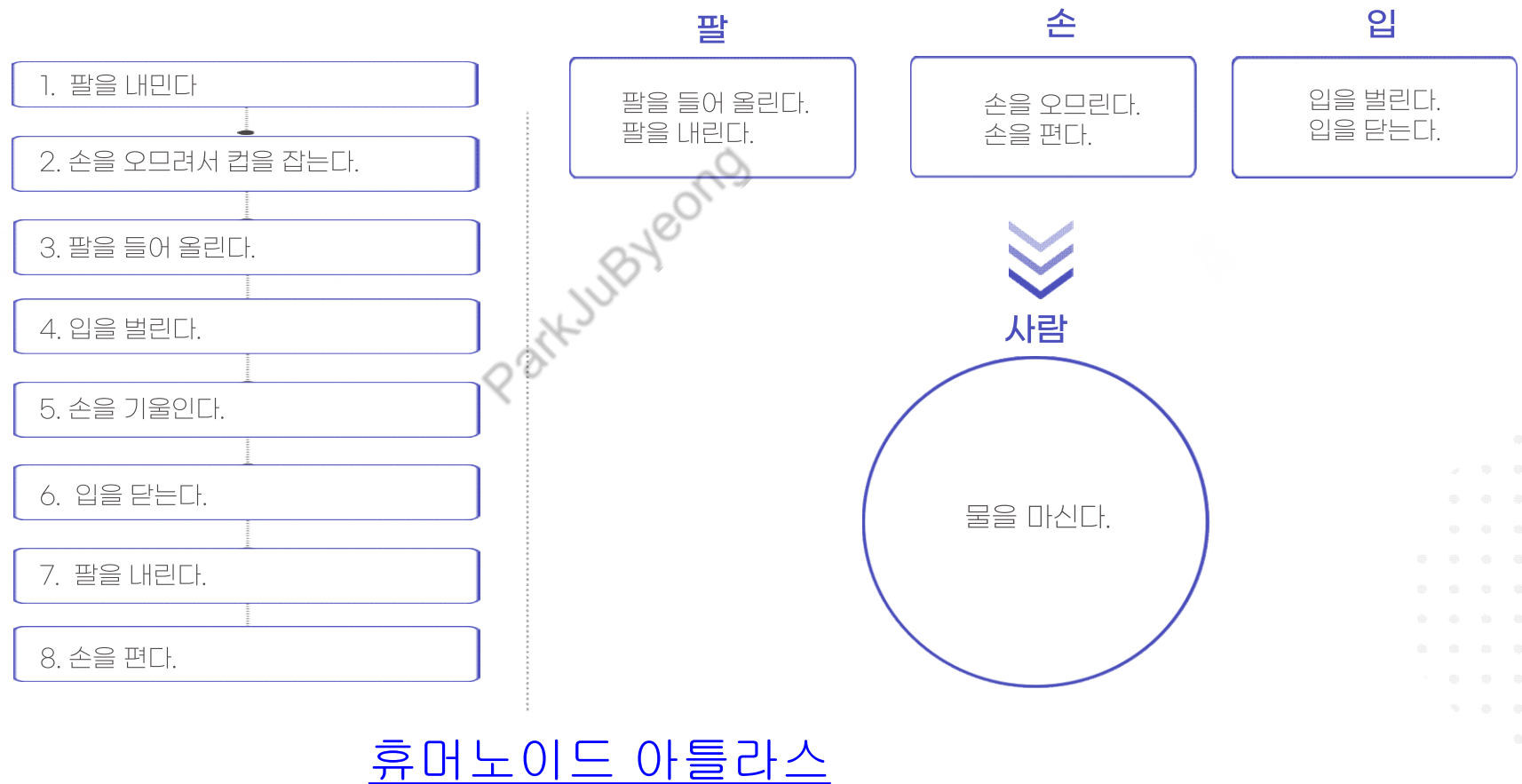
색상, 기름량, 주행거리,기어상태

기능:

시동, 에어컨 가동, 주행, 브레이크

현실을 그대로 프로그래밍에 반영하기 위해서

절차지향 VS 객체지향



휴머노이드 아틀라스

C, FORTRAN, COBOL



C++



JAVA

클래스



인스턴스화

»» ParkJuByeong

객체(인스턴스)





데이터:

색상, 크기, 볼륨, 전원

기능:

켜기, 끄기, 볼륨높이기, 볼륨 낮추기

```
class Tv
{
    //데이터 (멤버변수)
    String color;
    boolean power = false;
    int channel = 0;

    //기능 (메서드)
    void TurnOn()
    {
        power = true;
    }
    void TurnOff()
    {
        power = false;
    }
    void ChannelUp()
    {
        channel++;
    }
    void ChannelDown()
    {
        channel--;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Tv t1 = new Tv();

        t1.color = "RED";
        t1.TurnOn();
        t1.ChannelUp();
    }
}
```

객체의 생성과정

```
public class Main {  
    public static void main(String[] args) {  
        |  
        Tv t1 = new Tv();  
  
        t1.color = "RED";  
  
        t1.TurnOn();  
        t1.ChannelUp();  
  
    }  
}
```

T1

0x000A

0x000A

객체의멤버	값
color	"RED"
power	false
channel	0
TurnOn	
TuronOff	
ChannelUp	
ChannelDown	

```
Tv t1 = new Tv();
Tv t2 = new Tv();

t1.color = "RED";
t2.color = "BLACK";
t1.TurnOn();
t1.ChannelUp();
```

T1

0x000A

0x000A

객체의멤버	값
color	"RED"
power	true
channel	1
TurnOn	
TuronOff	
ChannelUp	
ChannelDown	

T2

0x300A

0x300A

객체의멤버	값
color	"BLACK"
power	false
channel	0
TurnOn	
TuronOff	
ChannelUp	
ChannelDown	

객체들은 서로 다른 메모리 영역을 가지고 있다.

```

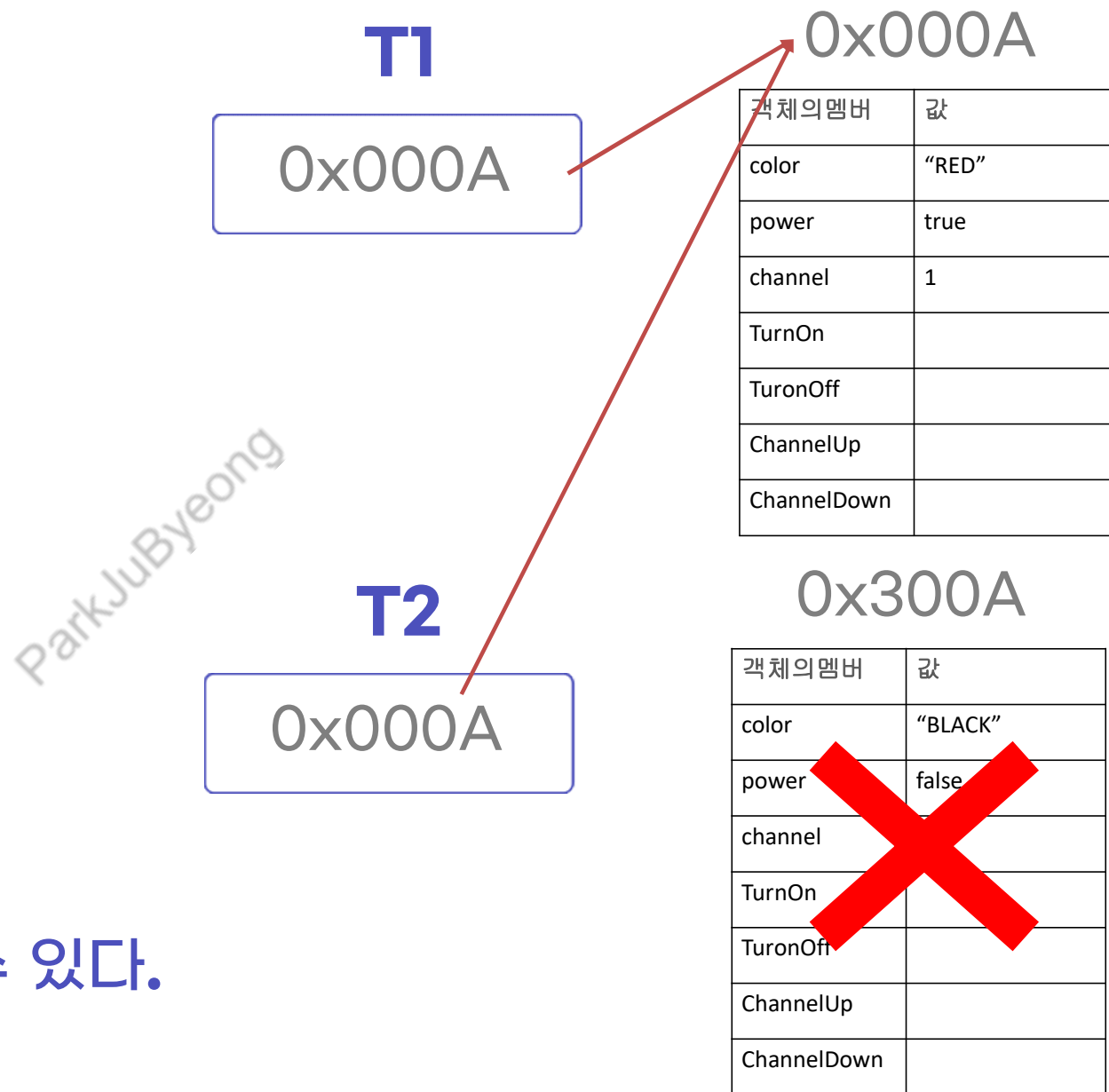
Tv t1 = new Tv();
Tv t2 = new Tv();

t1.color = "RED";
t2.color = "BLACK";
t1.TurnOn();
t1.ChannelUp();

t2 = t1;

```

참조변수들 끼리 값을 교환할수 있다.

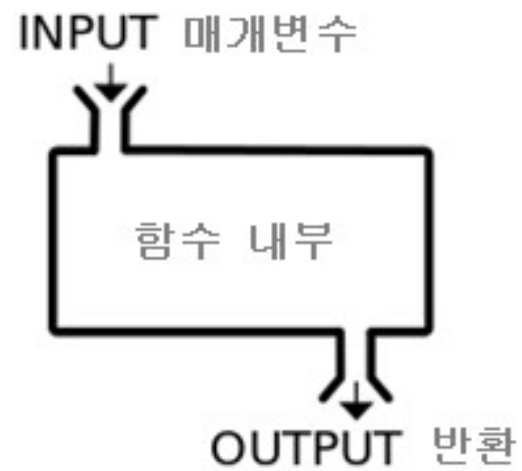


02

메서드

ParkJuByeong

메서드



```
int add(int x, int y)
{
    int result = x + y;
    return result;
}
```

Annotations for the code snippet:

- 함수 리턴값의 데이터 타입 (Function return value data type) points to `int`.
- 함수의 이름 (Function name) points to `add`.
- 함수의 매개변수 (Function parameter) points to `(int x, int y)`.
- 함수 내부 (Function internal) points to the block of code inside the curly braces.
- 함수 내부에서 가공한 데이터를 반환 (Return data processed inside the function) points to `return result;`.

코드의 재사용성 및 가독성이 좋아진다

```
public static String getKeyBoardValue(String str)
{
    Scanner scanner = new Scanner(System.in); //입력을 받기 위한 스캐너 객체를 생성
    System.out.println(str);
    String input = scanner.nextLine(); //키보드로부터 값을 입력 받아 input 변수에 저장한다.
    return input;
}
```

```
String name1 = Util.getKeyBoardValue("이름을 입력하세요:");
String name2 = Util.getKeyBoardValue("이름을 입력하세요:");
String name3 = Util.getKeyBoardValue("이름을 입력하세요:");
```

```
Scanner scanner = new Scanner(System.in); /
System.out.println("이름을 입력하세요");
String name1 = scanner.nextLine(); //키보드로

System.out.println("이름을 입력하세요");
String name2 = scanner.nextLine(); //키보드로

System.out.println("이름을 입력하세요");
String name3 = scanner.nextLine(); //키보드로
```

메서드 선언

```
7 void test()  
8 {  
9  
10 }  
11  
12 |  
13 class Tv  
14 {  
15     //데이터 (멤버 변수)  
16     String color;  
17     boolean power = false;  
18     int channel = 0;  
19  
20     String state;  
21     //기능 (메서드)  
22     void TurnOn()  
23     {  
24         if(state.equals("고장"))  
25             return;  
26  
27         power = true;  
28  
29     }  
30
```

클래스의 범위가 아니면
선언 할수 없다.

메서드는 클래스의 범위 안에
선언되어야 한다.

메서드 시그니처 : 메서드 명, 파라미터 타입, 갯수

```
int add(int x, int y)
{
    int result = x+y;
    return result;
}
```

```
int add(int x, int y)
{
    int result = x+y;
    return result;
}

void add(int x, int y)
{
    int result = x+y;
}
```

```
int add(int x, int y)
{
    int result = x+y;
    return result;
}

void add(int x, int y, int z)
{
    int result = x+y+z;
}
```

같은 클래스 안에 시그니처가 동일한 함수를 2개 선언할수 없다.

```
8 class Test1
9 {
10     int a()
11     {
12         return 0;
13     }
14 }
15
16 class Test2
17 {
18     int a()
19     {
20         return 0;
21     }
22 }
```

ParkJuByeong

클래스가 다르면 시그니처가 동일해도 별개이므로 상관없다.

메서드 호출

```
public class Main {  
  
    static int add(int x, int y)  
    {  
        int result = x+y;  
        return result;  
    }  
  
    public static void main(String[] args) {  
  
        int a = add(15,20);  
  
    }  
    int a = 35;  
}
```

인자, 매개변수, parameter

인수, argument

```
static int add(int x, int y)
{
    int result = x+y;
    return result;
}
```

```
int a = add("13", 20);
```

매개변수의 타입과 일치 혹은 자동타입캐스팅 되는
타입이어야한다 ERROR!

```
int a = add(15, 20, 13, 23);
```

개수가 일치해야 한다. ERROR!


```
add(15, 20);
```

리턴 되는 데이터는 안받아도 된다.

같은 클래스에서의 메서드 호출

```
class Test1
{
    int a()
    {
        System.out.println("메서드 a 호출");
        int result = b();
        return result;
    }

    int b()
    {
        System.out.println("메서드 b 호출");
        return 0;
    }
}
```



같은 클래스에서는 객체 생성없이 그냥 사용한다

```
class Test1
{
    int a()
    {
        int result = b();

        return result;
    }

    int b()
    {
        int result = c();
        return result;
    }
}

class Test2
{
    int c()
    {
        return 0;
    }

    int d()
    {
        return 0;
    }
}
```



```
int b()
{
    Test2 temp = new Test2();

    int result = temp.c();
    return result;
}
```

같은 클래스에서는 객체 생성 없이 그냥 사용한다

메서드 리턴

//기능 (메서드)

```
void TurnOn()  
{  
    power = true;  
}
```

리턴하는것이 없다면 void를 사용한다

```
void TurnOn()  
{  
    if(state.equals("고장"))  
        return;  
    power = true;  
}
```

void라도 특정라인에서 함수를
종료하고 싶을때 return을
사용할수있다.

```
int compare(int a , int b)
{
    if(a>b)
        return a;
}
```

무조건 return이 있어야 하는데 조건문 내부에만 있다면 컴파일 에러

```
int compare(int a , int b)
{
    if(a>b)
        return a;
    else
        return b;
}
```

```
int compare(int a , int b)
{
    if(a>b)
        return a;

    return b;
}
```

```
int add(int x, int y)
{
    return x+y;
}
```

변수에 담지 않아도 된다.

객체 배열

```
Tv tv1 = new Tv();  
Tv tv2 = new Tv();  
Tv tv3 = new Tv();
```



```
Tv[] tv = new Tv[3];
```

```
int[] arr = new int[5];  
System.out.println(arr[1]);
```

기본형은 배열로 생성시 초기화를 안해도
기본값이 들어가 있다.

```
Tv[] tv = new Tv[3];  
System.out.println(tv[1]);
```

배열의 요소가 객체라면 배열만 생성시 Null 이
들어가 있다.

```
tv[0] = new Tv();  
tv[1] = new Tv();  
tv[2] = new Tv();  
  
System.out.println(tv[1]);
```

각각의 요소들에 객체를 생성해서
넣어줘야한다.(반복문을 사용하자)

객체 배열 사용

```
Tv myTv = new Tv();  
  
myTv.color = "BLACK";  
myTv.TurnOn();
```

```
Tv[] tv = new Tv[3];  
  
tv[0] = new Tv();  
tv[1] = new Tv();  
tv[2] = new Tv();  
  
tv[0].color = "RED";  
tv[0].TurnOn();  
  
System.out.println(tv[0].color);
```

요소 안에 객체가 있다.
객체.멤버 변수 or 멤버 메서드를 사용한다

최종 정리

```
class MyMath
{
    int add(int x, int y)
    {
        int result = x+y;
        return result;
    }

    int add(int x, int y , int z)
    {
        return x+y+z;
    }
}

public class Main {

    public static void main(String[] args) {

        MyMath m1 = new MyMath();

        int result = m1.add(10, 25);

        System.out.println(result);

        System.out.println(m1.add(10, 25, 5));

    }

}
```

실습문제1

1. 오른쪽의 클래스를 구현 하여 아래의 식의 답을 출력하자(변수는 result 한 개만 사용)

```
System.out.println(result);
System.out.println(500/5+3*27-5);
```

```
<terminate>
176
176
```

2. Student 클래스를 구현 후 아래와 같이 출력하자(객체배열활용)

국어	영어	수학	합계	평균
73	1	69	143	97
66	50	33	149	127
51	92	49	192	159
41	90	55	186	149
72	75	72	219	171

객체배열 활용
성적은 Math.random()
으로 1~100점 사이

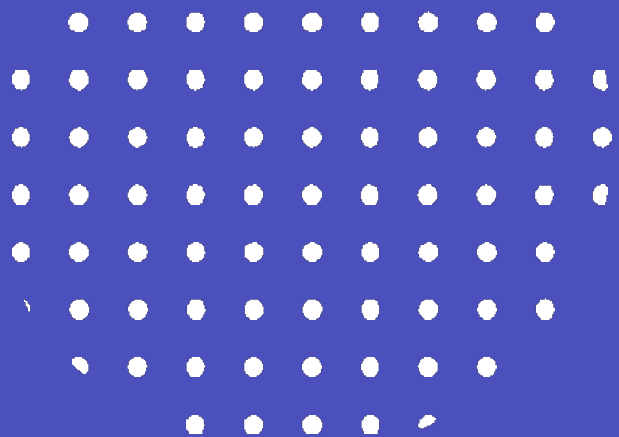
3. Archer 클래스를 구현하여 객체배열 을 이용하여 10개의 객체를 생성 후 공격력 기준 오름차순 정렬하자(삽입 정렬 구현)

클래스명	Archer	
멤버변수	String name	캐릭터명
	int power	공격력(1~50) random
	int hp	체력(1~100) random
	int armer	방어력(1~10) random
메서드	attack()	공격 내부는 일단 비워두자
	showState()	공격력, 체력, 방어력 출 력

캐릭터명: 유저0	공격력: 2	체력: 43	방어력: 1
캐릭터명: 유저1	공격력: 12	체력: 70	방어력: 10
캐릭터명: 유저2	공격력: 35	체력: 85	방어력: 3
캐릭터명: 유저3	공격력: 50	체력: 12	방어력: 5
캐릭터명: 유저4	공격력: 32	체력: 4	방어력: 7
캐릭터명: 유저5	공격력: 26	체력: 28	방어력: 5
캐릭터명: 유저6	공격력: 31	체력: 42	방어력: 2
캐릭터명: 유저7	공격력: 31	체력: 74	방어력: 5
캐릭터명: 유저8	공격력: 41	체력: 58	방어력: 3
캐릭터명: 유저9	공격력: 33	체력: 27	방어력: 7
--- 공격력순으로 정렬 ---			
캐릭터명: 유저0	공격력: 2	체력: 43	방어력: 1
캐릭터명: 유저1	공격력: 12	체력: 70	방어력: 10
캐릭터명: 유저5	공격력: 26	체력: 28	방어력: 5
캐릭터명: 유저7	공격력: 31	체력: 74	방어력: 5
캐릭터명: 유저6	공격력: 31	체력: 42	방어력: 2

클래스명	MyMath	
멤버변수	없음	
메서드	add()	매개변수: int 2개 리턴: 더하기 결과
	subtract()	매개변수: int 2개 리턴: 빼기 결과
	multiply()	매개변수: int 2개 리턴: 곱하기 결과
	divide()	매개변수: int 2개 리턴: 나누기 결과

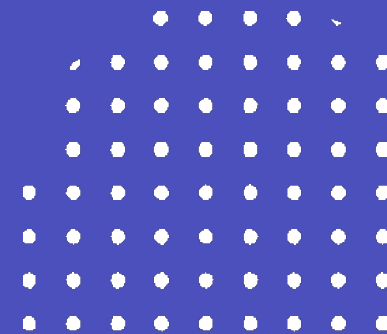
클래스명	Student	
멤버변수	String name	학생이름
	int kor	국어
	int eng	영어
	int math	수학
메서드	getTotal()	매개변수: 없음 리턴: 합계
	getAverage()	매개변수: int 2개 리턴: 평균



ParkJuByeong

03

메서드 활용



재귀호출

메서드 내에서 자기 자신을 다시 호출하는것

```
void method()  
{  
    System.out.println("메서드 호출");  
    // 자기자신을 내부에서 다시 호출한다.  
    method();  
}
```

```
void method(int a)  
{  
    if(a == 0)  
        return ;  
  
    System.out.println("메서드 호출"+a);  
  
    // 자기자신을 내부에서 다시 호출한다.  
    method(--a);  
}
```

무한루프 이므로 반드시 나갈 수 있는 조건이 있어야 한다.

재귀호출 사용

```
class Test
{
    void method(int a)
    {
        if(a == 0)
            return ;

        System.out.println("메서드 호출"+a);

        // 자기자신을 내부에서 다시 호출한다.
        method(--a);
    }
}

public class Main {
    public static void main(String[] args) {

        Test t1 = new Test();
        t1.method(5);

    }
}
```

반복문 VS 재귀호출

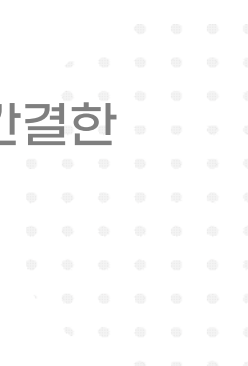
ParkJuByeong



반복문

- 빠르다
- 제약이 없다.

재귀호출

- 호출될수록 느려진다.
 - 특정상황에서 반복문보다 간결한 코드가 나온다.
- 

재귀호출을 이용한 팩토리얼 구현

ex) $!5 = 5 \times 4 \times 3 \times 2 \times 1$

```
class Test
{
    int factorial(int n)
    {
        if(n == 1)
            return 1;

        return n*factorial(n-1);
    }
}

public class Main {

    public static void main(String[] args) {

        Test t1 = new Test();
        //!5 를 계산한다.
        int result = t1.factorial(5);

        System.out.println(result);
    }
}
```

```
int factorial(int n) 5
{
    if(n == 1)
        return 1;
    return n*factorial(n-1);
}
```

```
int factorial(int n) 2
{
    if(n == 1)
        return 1;
    return n*factorial(n-1);
}
```

```
int factorial(int n) 4
{
    if(n == 1)
        return 1;
    return n*factorial(n-1);
}
```

```
int factorial(int n) 1
{
    if(n == 1)
        return 1;
    return n*factorial(n-1);
}
```

```
int factorial(int n) 3
{
    if(n == 1)
        return 1;
    return n*factorial(n-1);
}
```

```
int factorial(int n)
{
    // 입력값 필터링
    if (n >= 13 || n < 0)
        return 1;

    if (n == 1)
        return 1;

    return n * factorial(n-1);
}
```

→ 13보다 큰 팩토리얼은
int형보다 크고 음수는
고려하지 않는다.

메서드 오버로딩

```
int add(int x, int y)
{
    return x+y;
}

int add(int x, int y, int z)
{
    return x+y+z;
}

int add2(int x, int y)
{
    return x+y;
}

float add(float x, float y)
{
    return x+y;
}

float add(int x, int y)
{
    return x+y;
}

int add(int a, int b)
{
    return a+b;
}
```

오버로딩

add2 새로운 메서드 선언

오버로딩

리턴타입은 오버로딩의 조건 아님.
메서드 중복으로 ERROR!

변수 이름만 바뀐 것은 오버로딩이 아님.
메서드 중복으로 ERROR!

1. 메서드 이름이 같아야 한다.
2. 매개변수의 개수 혹은 타입이 달라야 한다.
3. 리턴 타입은 조건이 아니다.

가변인자

1. 매개변수의 개수를 지정하지 않는것
2. JDK1.5 부터 사용가능

```
int add(int... n)
{
    int sum = 0;
    for(int i : n)
        sum += i;

    return sum;
}
```

```
Test t1 = new Test();
t1.add(10,32);
t1.add(10,32,50);
t1.add(10,32,3,2);
t1.add(10,32,32,1,2,3);
t1.add(10,32,32,1,2,3,3,2,1,3,54,65,3,7,5,4);
```

배열로 생성되며 배열의 길이는
호출하는쪽에서 인수(argument)의
개수로 정해진다.

```
int add(int... n, int a)
```

→ add(32,12) 호출 했을때 12가 n 인지 a 인지
불분명 하다. ERROR!

```
int add(int... n, boolean isLogYn)
```

→ 혼란을 방지하기 위해 가변인자는 항상
마지막에 위치해야 한다.ERROR!

```
Test t1 = new Test();  
t1.add();
```

→ 길이0 배열이 만들어진다.

```
Test t1 = new Test();  
t1.add(new int[]{32,12,32});
```

→ 배열을 넘겨줘도 된다.

```
Test t1 = new Test();  
int[] arr = new int[]{32,12,32};  
t1.add(arr);
```


그냥 매개변수를 배열로 만들면 되지 않나?

```
int add(int[] n)
{
    int sum = 0;
    for(int i : n )
        sum +=i;

    return sum;
}
```

```
int add(int... n)
{
    int sum = 0;
    for(int i : n )
        sum +=i;

    return sum;
}
```

```
t1.add(new int[0]);
t1.add(null);
t1.add(32,12,32);
t1.add();
```

```
int add(int a, int b){return a+b;}  
int add(int a, int b,int c){return a+b+c;}  
int add(int a, int b,int c,int d){return a+b+c+d;}
```

=

```
int add(int... n)  
{  
    int sum = 0;  
    for(int i : n )  
        sum +=i;  
  
    return sum;  
}
```

오버로딩으로도 구현가능하나 코드가 확실히 편하고 유연하다

실습문제2

1. 실습문제1에서 구현한 MyMath 클래스의 add메서드를 float 과 double도 가능하도록 오버로딩 해보자

2. MyMath 클래스에 평균, 최대, 최소값을 구하는 메서드를 추가 하자

클래스명	MyMath	
메서드	avg()	매개변수: int 가변인자 리턴: int형 평균
	max()	매개변수: int 가변인자 리턴: int형 최대값
	min()	매개변수: int 가변인자 리턴: int형 최소값

3. MyMath 클래스에 거듭제곱을 계산하는 메서드를 만들자
(재귀호출 사용할것)

power(3,4) -> 3x3x3x3 =81

클래스명	MyMath	
메서드	power()	매개변수: int x, int n 리턴: x를 n제곱한 결과

4. 하노이타워 문제를 재귀호출을 이용하여 풀이과정을 출력해보자.(원판의 개수가 늘어나도 풀려야 한다.)

- 한번에 하나의 원판만 옮길수 있다.
- 작은 원판 위에 큰 원판이 올수 없다.
- A에 있는 원판을 C로 모두 옮기는것이 목표다.



```
int power(int x , int n)
{
    if(n ==1)
        return x;

    return x*power(x,n-1);
}
```

```
MyMath t1 = new MyMath();
|
long startTime = System.nanoTime();
System.out.println(t1.power(3, 1500));

System.out.println(System.nanoTime() - startTime + " 나노세컨드");
```

Problems Javadoc Declaration

<terminated> main [Java Application] C:\WU

6351001347334244913

318000 나노세컨드

시간복잡도 : $O(n)$

```
int power(int x,int n)
{
    if(n==1)
        return x;

    //짝수
    if(n%2 ==0)
        return power(x*x,n/2);
    else
        return x*power(x*x,(n-1)/2);
}
```

<terminated> main [Java Application] C

6351001347334244913

125500 나노세컨드

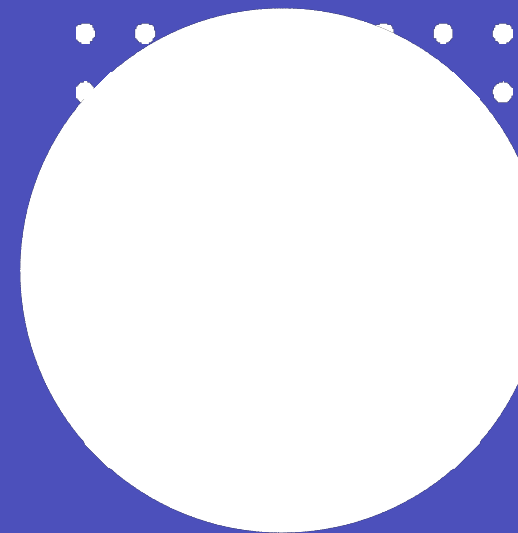
시간복잡도 : $O(\log n)$



THANK YOU



ParkJuByeong



강사 박주병