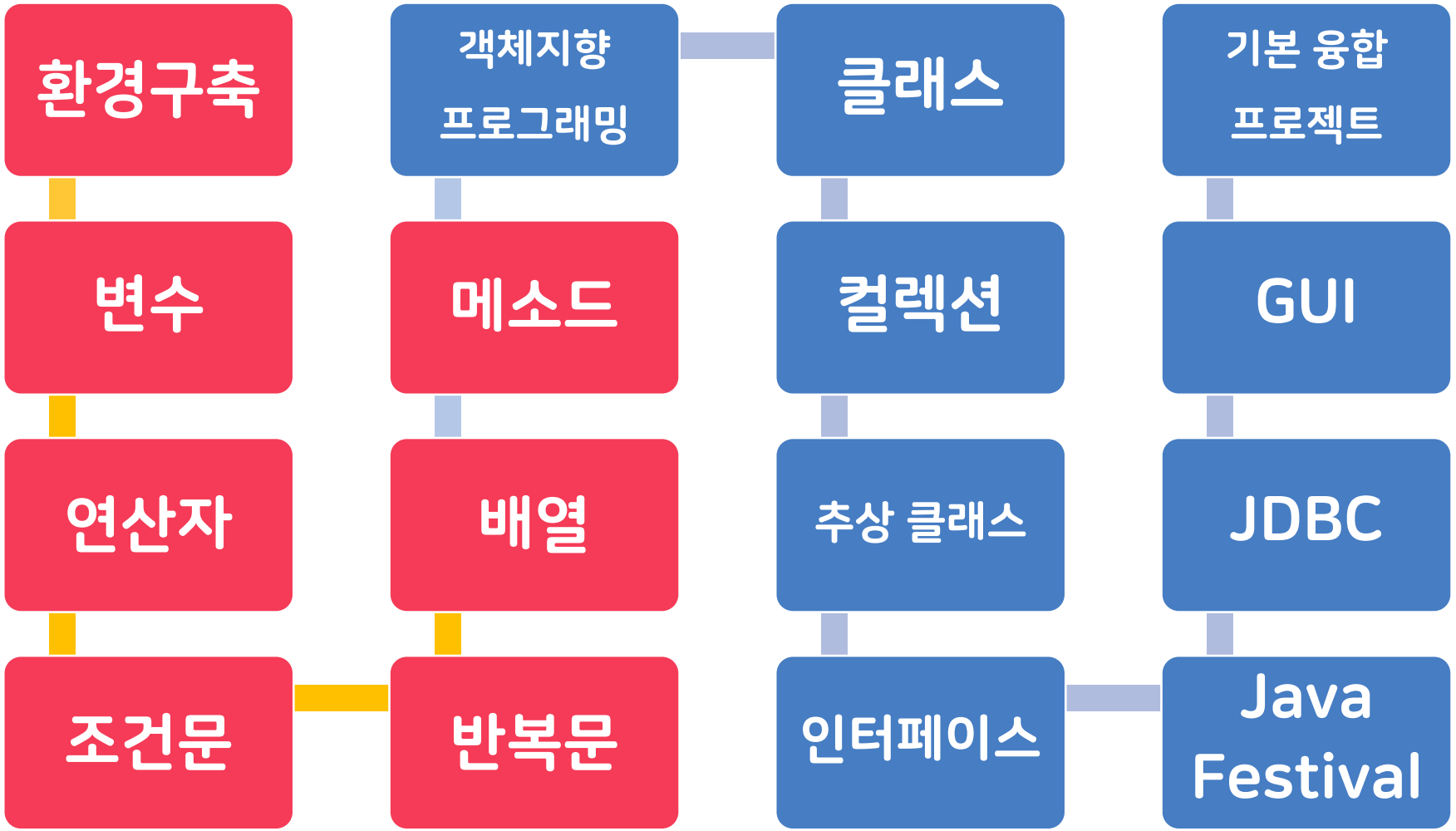
스마트인재개발원
Smart Human Resources Development

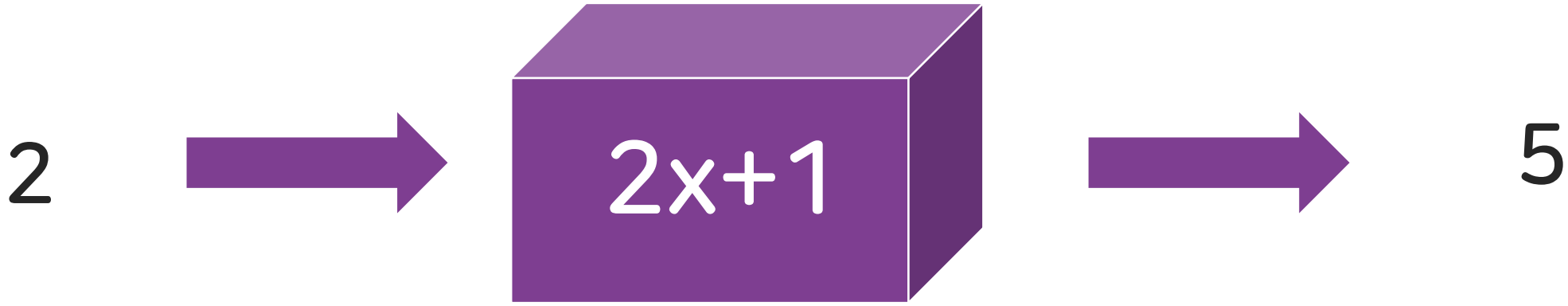
박 수 현 연구원

수업 진행방향


환경구축
변수
연산자
조건문
객체지향 프로그래밍
메소드
배열
반복문
클래스
컬렉션
추상 클래스
인터페이스
기본 융합 프로젝트
GUI
JDBC
Java Festival

학습목표

1. 메소드의 필요성을 이해한다.
2. 메소드를 선언하고 활용할 수 있다.
3. 메소드 오버로딩을 이해하고 활용할 수 있다.


2
2x+1
5


입력값
알고리즘
결과값

메소드(Method)

- 객체의 행위를 표현
- 반복적으로 사용되는 코드 최소화
- 어떤 작업을 수행하기 위한 명령문의 집합

더하다

빼다

곱하다

나누다

메소드(Method)의 기본구조


```
public int addNumber(int num1 , int num2){
    int result = num1 + num2;
    return result;
}
```

접근제한자 리턴타입 메소드이름(매개변수){
    실행문장1
    실행문장2
    return(반환 값)
}

메소드(Method)의 기본구조 - 리턴타입

- 메소드의 수행결과를 어떤 타입(자료형)으로 반환할 것인지 알려주는 것

- 아무것도 반환하지 않을 경우 리턴타입 대신 void를 사용한다.

- 메소드가 결과값을 반환하는 경우 반드시 return문을 사용하여

  메소드의 리턴타입에 맞는 결과값을 지정해야 한다.

메소드(Method)의 기본구조 - 리턴타입

1. 반환값이 없을 경우(void)

```
public void printResult(){


}
```

2. 반환값이 있을 경우

```
public int minus(){

}
```

```
public int minus(){

    return 0;
}
```

메소드(Method)의 종류

매개변수O
반환값 X

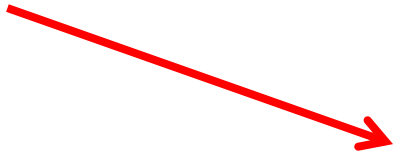
매개변수O
반환값 O

매개변수X
반환값 O

매개변수X
반환값 X

메소드(Method) – 매개변수X, 반환값X


```
public class Test {

    public static void main(String[] args) {
        print();
    }

    public static void print() {
        System.out.println("프로그램이 종료되었습니다.");
    }

}
```



메소드(Method) – 매개변수O, 반환값X


```java
public class Test {

    public static void main(String[] args) {
        addNumber(10, 15);
    }

    public static void addNumber(int num1, int num2) {
        int result = num1 + num2;
        System.out.println("연산결과 : "+result);
    }
}
```

num1 = 10 , num2 = 15

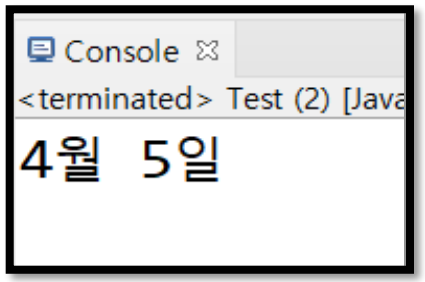
Console

<terminated> Test [Java

연산결과 : 25

메소드(Method) – 매개변수X, 반환값O


```java
public class Test {

    public static void main(String[] args) {
        System.out.println(toDay());

    }

    public static String toDay() {
        String today = "4월 5일";
        return today;
    }
}
```



메소드(Method) – 매개변수O, 반환값O


```
public class Test {

    public static void main(String[] args) {
        int value =addNumber(8, 12);
        System.out.println("연산결과 : "+value);
    }

    public static int addNumber(int num1, int num2) {
        int result = num1 + num2;
        return result;
    }
}
```

num1 = 8 , num2 = 12

Console

<terminated> Test [Jav

연산결과 : 20

메소드(Method) 실습문제

덧셈 뺄셈 나눗셈 곱셈이 가능한 메소드를 작성해봅시다!

```
int add = add(3,5);
int sub = sub(3,5);
int mul = mul(3,5);
int div = div(3,5);

System.out.println("====================");
System.out.println("덧셈은 : "+add);
System.out.println("뺄셈은 : "+sub);
System.out.println("곱셈은 : "+mul);
System.out.println("나눗셈은 : "+div);
```

```
====================
덧셈은 : 8
뺄셈은 : -2
곱셈은 : 15
나눗셈은 : 0
```

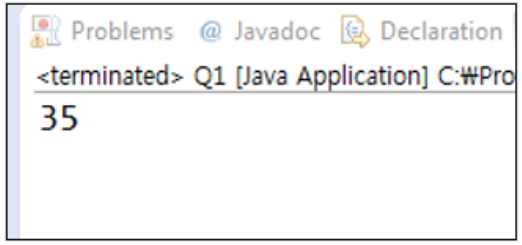
메소드(Method) 실습문제

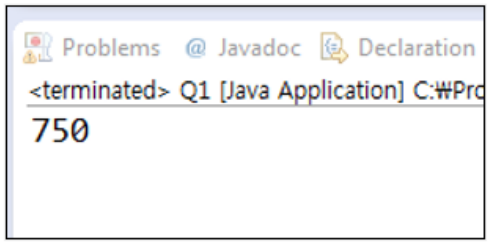
num1, num2, op (+, -, *, /) 를 매개변수로 받아 num1과 num2를 op에 맞게 연산한 결과값을 반환해주는 cal 메소드를 작성하세요.

```java
public static void main(String[] args) {
    // TODO Auto-generated method stub
    int num1 = 50;
    int num2 = 15;
    char op = '-';

    System.out.println(cal(num1, num2 , op));
}
```

```java
public static void main(String[] args) {
    // TODO Auto-generated method stub
    int num1 = 50;
    int num2 = 15;
    char op = '*';

    System.out.println(cal(num1, num2 , op));
}
```

<결과화면>

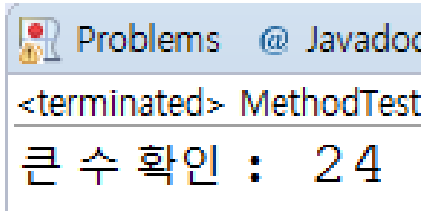




메소드(Method) 실습문제

2개의 양수를 받아 2개의 숫자 중 더 큰 수를
반환하는 메소드 largerNumbers( )을 만들어보세요
(만약 두 숫자가 같다면 0을 반환)

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

    int num1 = 10;
    int num2 = 24;
    int result = LargerNumbers(num1, num2);
    System.out.println("큰 수 확인 : " + result);
}
```

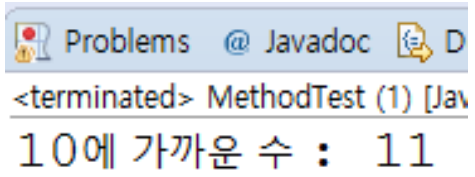


largerNumbers(2,7) -> 7을 반환
largerNumbers(9,5) -> 9를반환
largerNumbers(8,15) -> 15를 반환
largerNumbers(6, 6) -> 0을 반환

메소드(Method) 실습문제

2개의 정수를 받아 2개의 숫자 중 10에 더 가까운 수를
반환하는 메소드 close10을 만들어보세요
(만약 두 숫자 모두 10과의 차이가 같다면 0을 반환)

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

    int num1 = 11;
    int num2 = 5;
    int result = close10(num1, num2);
    System.out.println("10에 가까운 수 : " + result);
}
```

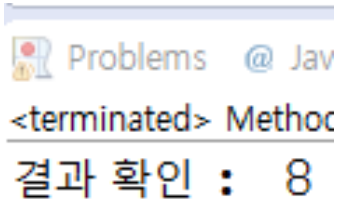


close10(2,7) -> 7을 반환
close10(-5,-1) -> -1을 반환
close10(13,2) -> 13을 반환
close10(8,12) -> 0을 반환

메소드(Method) 실습문제

2개의 정수 base, n을 받아 base의 n제곱 만큼
값을 반환하는 powerN() 메소드를 작성하세요

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

    int base = 2;
    int n = 3;
    int result = powerN(base, n);
    System.out.println("결과 확인 : " + result);
}
```



powerN(2,3) -> 8을 반환
powerN(3,3) -> 27을 반환
powerN(10,2) -> 100을 반환

메소드(Method) 실습문제

num2가 num1의 약수인지 확인하여 true또는 false를 반환하는
isDivisor()메소드를 구현하세요.

```
int num1 = 10;
int num2 = 2;
boolean divisor = isDivisor(num1, num2);

System.out.println(divisor);
```

Problems
<terminated> I
true

```
int num1 = 9;
int num2 = 2;
boolean divisor = isDivisor(num1, num2);

System.out.println(divisor);
```

Problems
<terminated> M
false

메소드(Method) 실습문제

자신을 제외한 약수의 총합을 구하는 getSum() 메소드를 작성하세요.

```
System.out.println(getSum(7));
```

메소드(Method) 실습문제

매개변수가 완전수라면 true, 아니라면 false를 반환하는
isPerfect 메소드를 구현하세요.

```
System.out.println(isPerfect(7));
// 매개변수가 완전수라면 true반환, 아니라면 false반환
```

※ 완전수란?
자기자신을 뺀 나머지 약수의 합이 자기자신과 같은
수

ex) 6 -> 1 + 2 + 3 = 6 -> 완전수
7 -> 1 - > 완전수 아님

메소드(Method) 실습문제

매개변수가 소수라면 true, 아니라면 false를 반환하는
isPrimeNum() 메소드를 구현하세요.

```
System.out.println(isPrimeNum(7));
//7이 소수라면 true반환, 아니라면 false반환
```

메소드(Method) 실습문제

피보나치 수열의 n번째 항을 출력하는 pibo()함수를 구현하세요.

```
System.out.println(getPibo(1));
System.out.println(getPibo(2));
System.out.println(getPibo(3));
System.out.println(getPibo(4));
System.out.println(getPibo(5));
```

```
Console
<terminated> pit
1
1
2
3
5
```

메소드 오버로딩(Method Overloading)

메소드 오버로딩[MethodOverloading]

: 메소드 오버로딩이란 메소드의 이름은 하나만 주고 매개변수를 다르게 함으로써 메소드를 여러 개 만드는 기법

: 오버로딩 메소드의 구별은 매개변수의 개수 및 매개변수의 데이터 타입으로 구분함

더하기

- 정수 더하기
- 소수 더하기

다음시간에는?

객체