

컴퓨터 프로그래밍

Method

# Method

- Function

- 작은 프로그램 (subprogram)
- 매개변수(parameter)를 받아서 반환(return)값을 내는 모듈

- Method

- 클래스 내부 함수 (member function)
- Java의 function은 전부 class 내부에 있으므로 전부 method
- 클래스 멤버 메소드는 객체의 기능(행위)를 정의

- Method 내부에서 sequence of instructions가 수행

# Function

- 함수(상자 수)

- 상자에 수를 넣으면 결과값을 돌려 받음

- In programming,

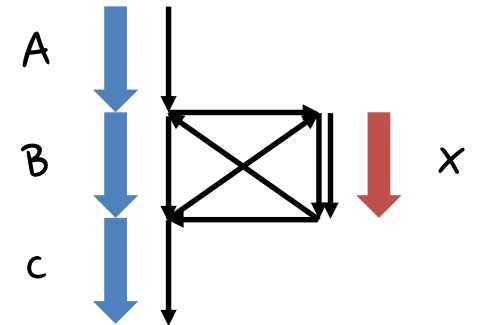
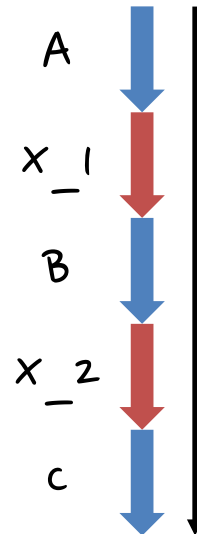
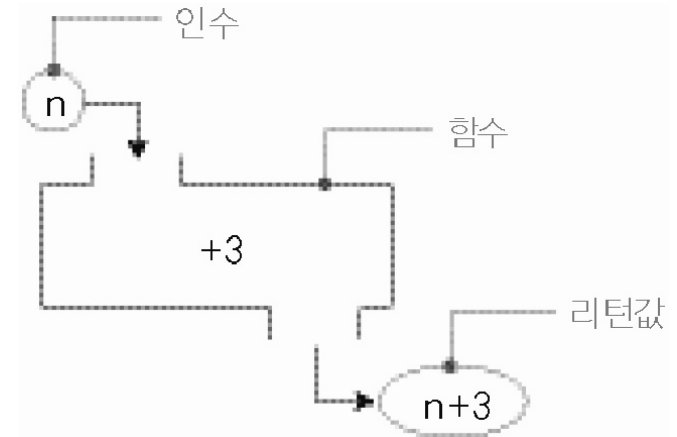
- Subprogram / module

- User-defined operator

- 동일한 코드를 조직화 (반복되는 코드 재사용)

- 체계적인 프로그램의 구조 관리

- Function 수행과정



# Method Definition / call

- Method definition

- `return_type function_name(parameter_list)` — header
  - {
    - statements — body
    - }
  - `int add(int a, int b) {`
  - `int c = a+b;`
    - `return c;`
  - `}`

- Method call

- `return_value = function_name(value_list);`
  - `result = add(x, y);`

# return

- Method의 종료  
— Method의 본체(body)가 모두 수행되었을 때  
— return 키워드를 만날 때
- return expression;
- expression의 결과를 return type과 동일한 data type으로 return
- return type이 void면 return value가 없는 method
- Example
  - ```
int divide(int a, int b) {  
    return (float)a/b;  
}
```

(float)a/b의 결과는 float이나 return type이 int이므로  
divide method는 a/b의 몫을 return

# Example

```
class MethodAdd {  
    public static void main(String[] args) {  
        int a = 1, b = 1;  
        int result;  
        System.out.println("프로그램 시작");  
        result = addReturn(a, b);  
        System.out.println("덧셈 결과는 "+result+"입니다");  
        addNoReturn(2, 2);  
        System.out.println("프로그램 끝");  
    }  
    public static int addReturn(int a, int b) {  
        int c = a+b;  
        return c;  
    }  
    public static void addNoReturn(int a, int b) {  
        System.out.println("덧셈 결과는 "+(a+b)+"입니다");  
    }  
}
```

method call

# Parameter Passing

- 모든 parameter는 그 값이 복사(copy)되어 전달

```
public class Test {  
  
    public static void main(String[] args) {  
        int n = 0;  
  
        add(n);  
  
        System.out.println(n);  
    }  
  
    public static void add(int n) {  
        n = n+1;  
    }  
}
```

- 출력 값은 무엇인가?
- main method의 n과 add method의 n은 어떤 관계인가?

# variable Scope

- variable scope
  - 변수가 보이는 범위
  - 변수의 선언 위치에 따라 scope가 다름
- 지역 변수 (local variable)
  - Method 내에서 선언된 변수
  - 선언된 method 내부에서만 볼 수 있음
  - Method가 호출될 때 생성 (dynamic variable)
  - Method 종료 시 소멸
- 멤버 변수 (member variable)
  - Method 밖, class 내부에 선언된 변수
  - class 강의 자료 참고



# variable Scope

```
class MethodAdd {  
    public static void main(String[] args) {  
        int a = 1, b = 1;  
        int result;  
        System.out.println("프로그램 시작");  
        result = addReturn(a, b);  
        System.out.println("덧셈 결과는 "+result+"입니다");  
        addNoReturn(2, 2);  
        System.out.println("프로그램 끝");  
    }  
    public static int addReturn(int a, int b) {  
        int c = a+b;  
        return c;  
    }  
    public static void addNoReturn(int a, int b) {  
        System.out.println("덧셈 결과는 "+(a+b)+"입니다");  
    }  
}
```

scope



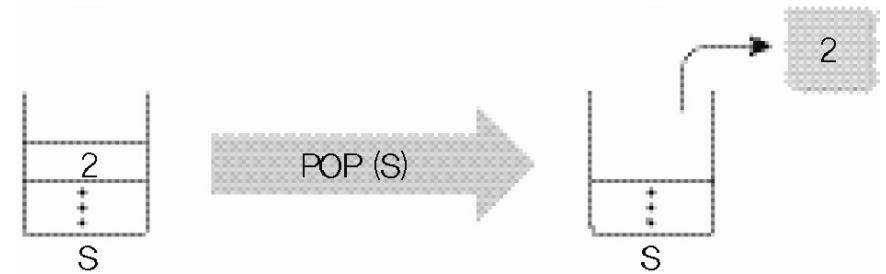
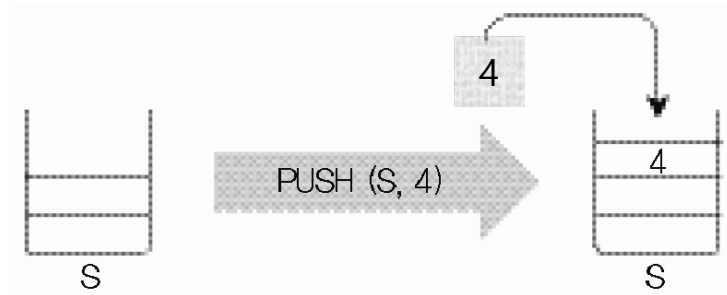
a, b, result는  
main method의  
지역 변수



a, b, c는  
addReturn method의  
지역 변수

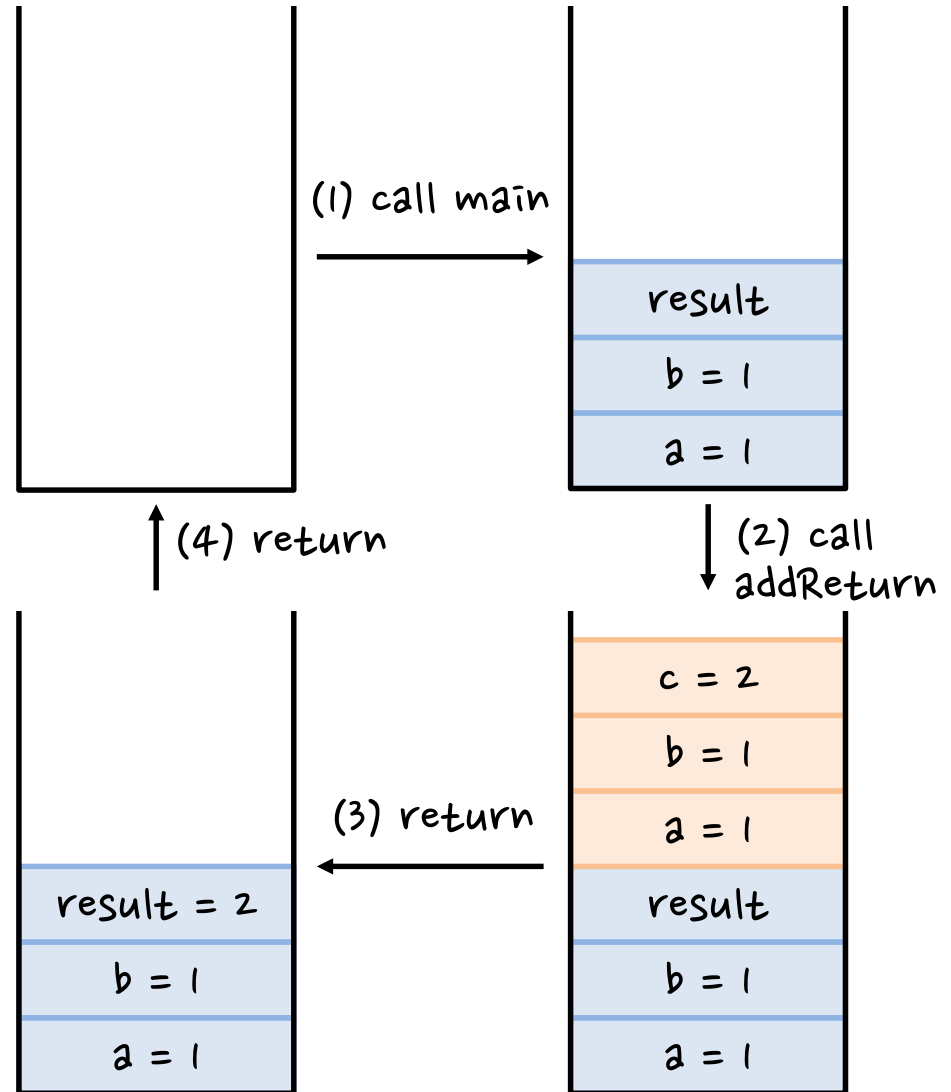
# Method call & Local variables

- 지역 변수는 저장 공간(memory)에 stack 형태로 생성 및 소멸
- Stack
  - 후입선출(LIFO: Last-In First-Out) 저장 방식
  - push & pop



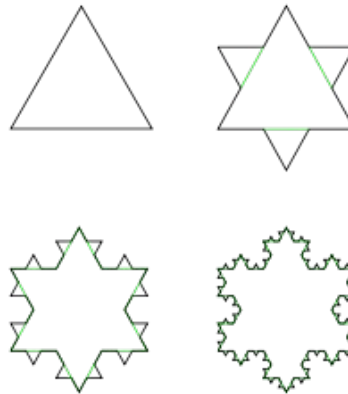
# Method call & Local variables

```
class MethodAdd {  
    public static void main(String[] args) {  
        int a = 1, b = 1;  
        int result;  
        System.out.println("프로그램 시작");  
        result = addReturn(a, b);  
        System.out.println("덧셈 결과는 "+result+"입니다");  
        addNoReturn(2, 2);  
        System.out.println("프로그램 끝");  
    }  
    public static int addReturn(int a, int b) {  
        int c = a+b;  
        return c;  
    }  
    public static void addNoReturn(int a, int b) {  
        System.out.println("덧셈 결과는 "+(a+b)+"입니다");  
    }  
}
```



# Recursive call

- Method가 종료되기 전에 자신을 다시 호출
- Fractal
  - 작은 구조가 전체 구조와 비슷한 형태로 끝없이 되풀이
  - 불규칙해 보이는 현상에서 찾을 수 있는 규칙적 구조



# Recursive call

- Factorial

- $n! = n \times (n-1) \times \dots \times 1$

- 재귀적 표현

- $0! = 1$

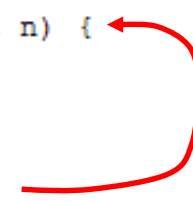
- $n! = n \times (n-1)!$

```
import java.util.Scanner;

public class Factorial {
    public static void main(String[] args) {
        int n;

        Scanner input = new Scanner(System.in);
        System.out.println("자연수를 하나 입력하십시오");
        n = input.nextInt();

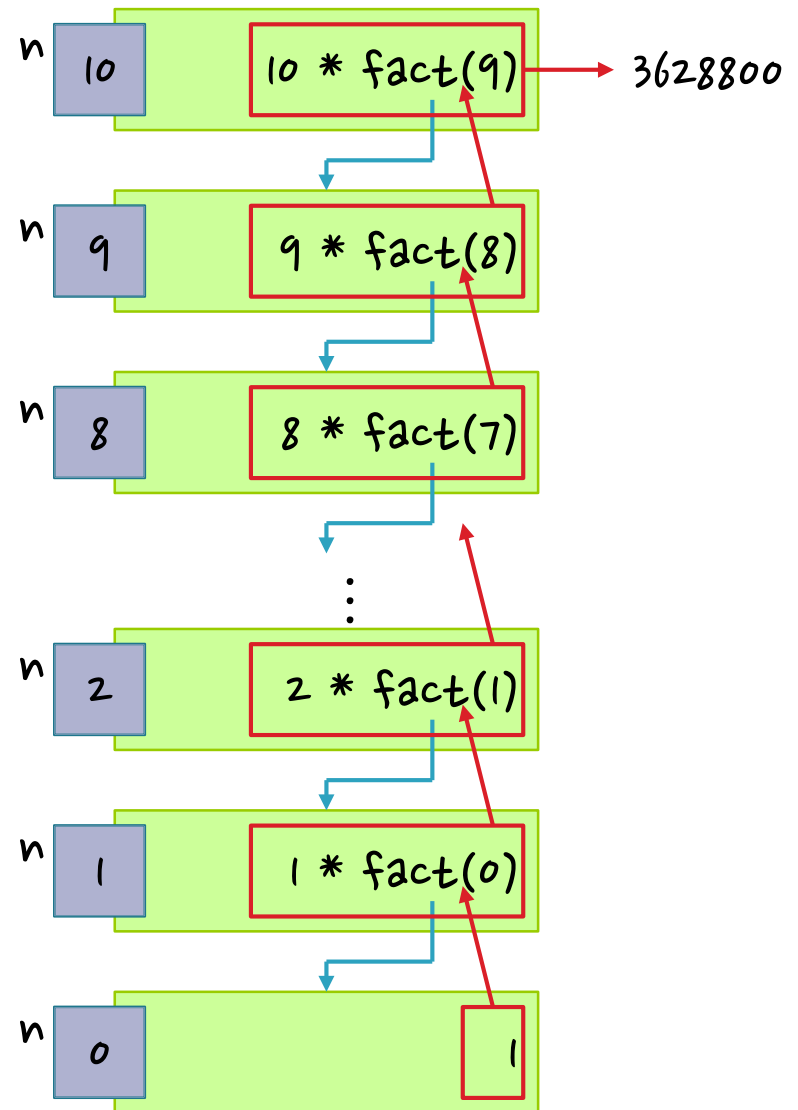
        System.out.println(n+"! = "+fact(n));
    }
    public static long fact(int n) {
        if (n == 0)
            return 1;
        else
            return n*fact(n-1);
    }
}
```



# Recursive call

- Factorial
  - Method calls

```
public static long fact(int n) {  
    if (n == 0)  
        return 1;  
    else  
        return n*fact(n-1);  
}
```



# Summary

- Method

- Definition

- Method의 header + body

- Function call

- Method에 정의된 parameter를 전달하여 호출
    - return이 정의된 경우 method는 연산 값을 return
    - Parameter와 return value는 복사되어 전달
    - recursive call

- variable scope

- Local variables은 변수가 선언된 block 내에서만 유효

시작  
결과

- 다음 프로그램의 예상 결과와 실제 결과는?

```
class Addition {  
  
    public static int add(int number) {  
        return ++number;  
    }  
  
    public static void main(String[] args) {  
        int number = 10;  
        add(number);  
        System.out.println("number = " + number);  
    }  
}
```



시  
스  
리  
브

- 정수를 입력 받고, 1부터 입력 받은 정수까지의 합을 구하는 프로그램
  - sequentialSum 메소드를 정의
  - sequentialSum 메소드는 1부터 인자로 입력 받은 정수까지의 합을 리턴

시작

- 연도를 인자로 받아서 윤년인지 여부를 판단하는 프로그램
  - isLeapYear 메소드를 정의
  - isLeapYear 메소드는 인자로 입력 받은 연도가 윤년이면 true, 그렇지 않으면 false 리턴
    - 연도를 4로 나누어 떨어지면 윤년
    - 4로 나누어 떨어지는 연도 중에서 100으로 나누어 떨어지는 연도는 평년
    - 100으로 나누어 떨어지는 연도 중 다시 400으로 나누어 떨어지는 연도는 윤년

## 시습

- 원의 반지름을 입력 받은 후 원의 둘레와 넓이를 구하는 프로그램
  - 3개의 메소드를 작성
    - 반지름을 입력 받는 setRadius 메소드
    - 둘레를 구하는 calcPerimeter 메소드
    - 넓이를 구하는 calcArea 메소드

---

반지름을 입력하세요 : 5

원의 둘레는 31.415920

원의 면적은 78.539800

시작

- Recursive call을 이용하여 피보나치(Fibonacci) 수열 계산

(피보나치 수열)

$$F_0 = 0, F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2} \quad \text{if } n \geq 2$$

몇 번째 피보나치 수열 값을 계산할까요? 10

10번째 피보나치 수열 값은 55입니다