

JAVA 프로그램 실습


Method

경북대학교
소프트웨어융합과
배희호 교수

JAVA Program 구조

■ 지금까지 작성한 Program 문제점

```
public static void main(String[] args) {  
    여러가지를 ;  
    고려해도;  
    반복되고;  
    복잡하고;  
    보기 힘들다;  
    혼자서 작성해야 한다;  
}
```



- 혼자 작성해야 함 -> 나누어 작성 할 수 있도록
- 반복되는 부분이 많음 -> 재 사용하는 방법

JAVA Program 구조

- JAVA Program은 1개 이상의 Class들로 구성
 - Class는 1개 이상의 Method들로 구성
 - Method는 Program의 문장(*statements*)들로 구성
 - JAVA Program은 항상 *main()* Method를 먼저 수행

```
public class MyProgram {  
    public static void main(String[] args){    //Method Header  
  
        //Method Body  
    }  
  
    public static void test( ){  
  
    }  
}
```

JAVA Program 구조

■ Class

- Program을 구성하는 단위로 개념(concept)을 표현
- Class 이름은 File 이름과 일치해야 하므로 File 이름의 대소문자를 정확히 구분

■ Method

- Class 내부를 구성하는 하나의 요소가 Method
- Method는 절차적 언어에서 함수(function)에 해당
- Method 이름 앞에는 반드시 Method return value의 Data Type인 return Type을 기술해야 함
- Method 이름 이후의 괄호 사이에 기술되는 것을 Method 인자 목록(parameter list)
- 예) main() 메소드의 main은 메소드 이름이고, void는 반환 값 자료형이며, String[] args가 메소드 인자
- Method의 반환 값 자료형 앞에 기술된 여러 단어인 public과 static은 Method의 특징을 기술하는 수정자(modifiers)

JAVA Program 구조

- 모든 JAVA Application에는 main() Method가 있어야 함
 - JAVA Application은 main() Method로부터 시작됨

```
public static void main(String[] args) {  
    문장;  
    문장;  
    ...  
}
```

main() 메소드는 자바 실행기(java interpreter)에 의해 호출되며 값을 반환하지 않음
(Program을 실행하기 위해 java Test라는 명령을 입력하면 JAVA 실행기인 java.exe가 실행되며 이것이 Test 클래스의 main() 메소드를 호출)

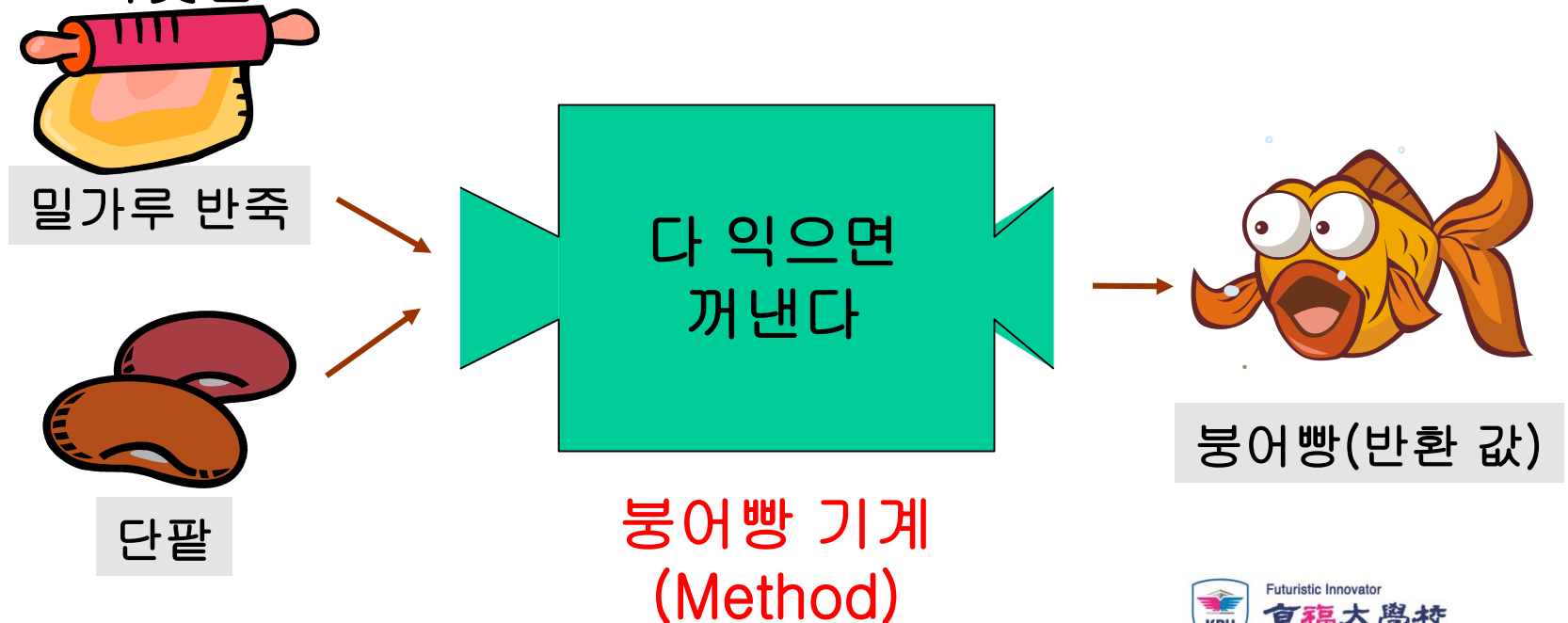
JAVA Program 구조

- 보통 다른 언어에는 함수라는 것이 별도로 존재
- JAVA는 Class를 떠나 존재하는 것은 있을 수 없기 때문에
JAVA의 함수는 따로 존재하지 않고 Class내에 존재
- JAVA는 이 Class내의 함수를 메소드(Method)라고 부름
- 보통 함수(Function)와 메소드(Method)가 공존하는 언어(
예: Python)에서는 2 개를 구분하여 말하기도 하지만,
JAVA는 보통 메소드와 함수를 구분하여 말하지 않음
- JAVA에서 사용되는 함수의 정확한 명칭은 메소드

Method

■ Method 개념

- 특정한 일을 수행하는 “메소드”
- 자주 반복하여 사용하는 내용에 대해 특정 이름으로 정의한 묶음
- 다른 언어의 함수(function)나 프로시저(procedure)와 비슷함



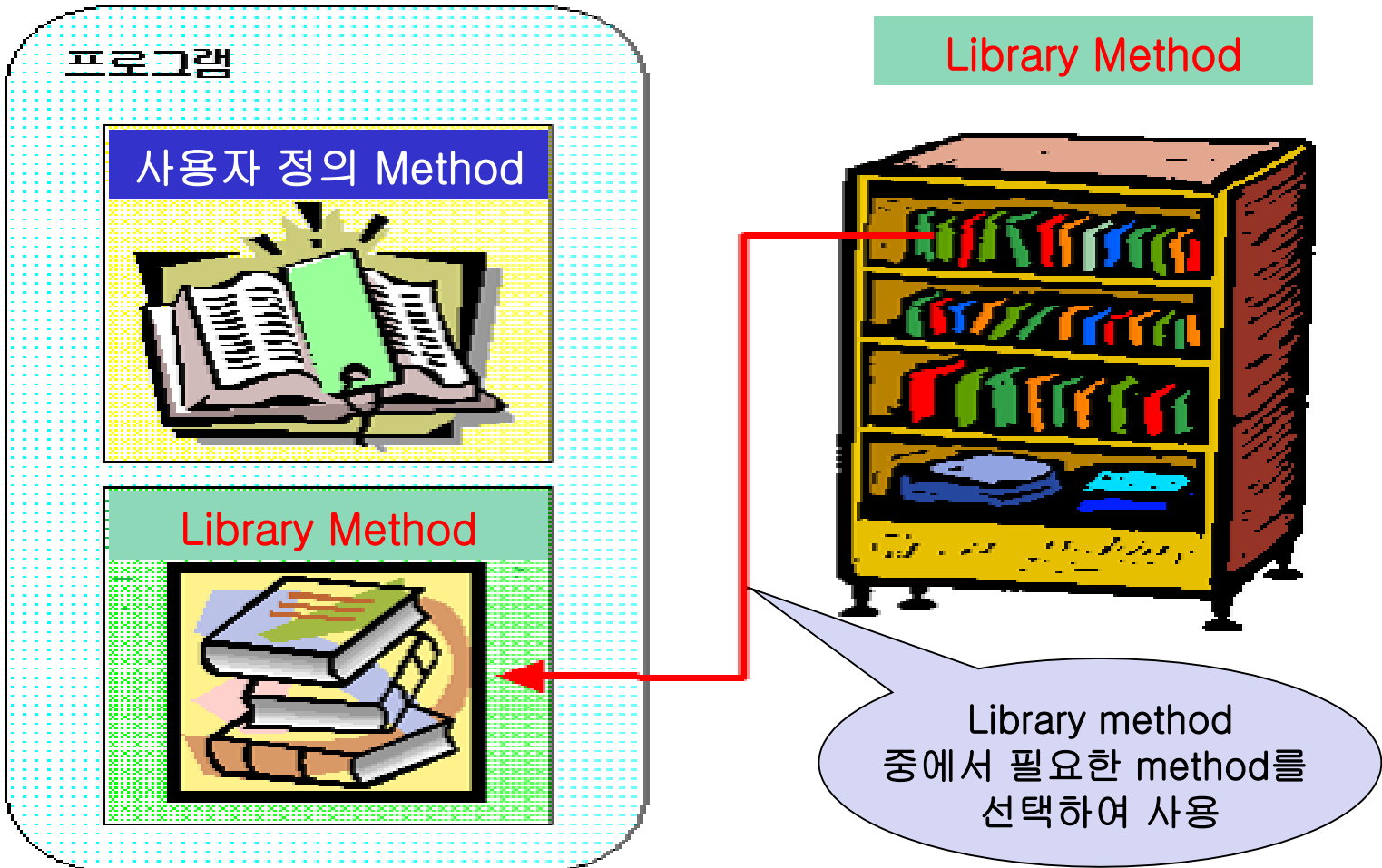
Method

■ 정의

- 객체가 할 수 있는 행동(Behavior)을 정의
- 어떤 작업을 수행하기 위해 **실행할 문장들(Statements)**을 묶어 놓은 것
- Class내에서 객체가 제공하는 행위를 작성하는 방법으로 **객체의 상태를 변경하고 참조하기 위한 것**
- 어떤 값을 입력 받아서 처리하고 그 결과를 되 돌려줌 (입력 받는 값이 없을 수도 있고 결과를 돌려주지 않을 수도 있음)
- Method는 자신의 작업을 수행할 때 제어와 필요한 Data를 전달받고 처리 한 후 호출한 곳으로 처리 결과를 반환하는 가상적인 상자(Black Box)와 같음

Method

■ Method의 종류

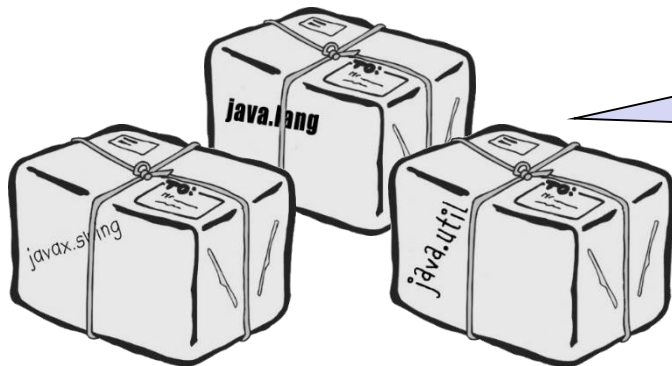


Library Method

- 매우 많은 수의 Method가 이미 Library에 들어 있음
 - 이들 Method를 사용하여 효율적으로 Program을 개발을 할 수 있음
- Library에 있는 Method들의 내부(구조)는 몰라도 됨
- Library에 어떤 Method가 있나?
 - JAVA 기본 Library에 들어 있는 Class와 Method 사용법 (사용자 설명서)
 - API: Application Programming Interface
 - <http://docs.oracle.com/javase/8/docs/api/>
- JAVA Homepage API에 영어 해석 하기 어려우면 아래에 있는 글씨 뒤에 K라고 붙은 건 다 한글 해석
 - [개요 \(Java 2 Platform SE 5.0\) \(joongbu.ac.kr\)](http://joongbu.ac.kr)

Library Method

- Library(JAVA API)
 - JAVA에는 미리 만들어진 Class가 엄청나게 많이 있음
 - 이렇게 미리 만들어져 있는 Class는 우리가 작성하는 Instance Code와 비슷하다고 보면 되는데 **모두 미리 Compile이 된 상태로 제공**
 - Library에 들어있는 Class는 필요에 맞게 잘 찾아서 사용하기만 하면 됨
 - JAVA API에서 Class는 Package 단위로 묶여 있음
 - API에 들어있는 Class를 사용하려면 그 Class가 어떤 Package에 들어있는지 알아야 함



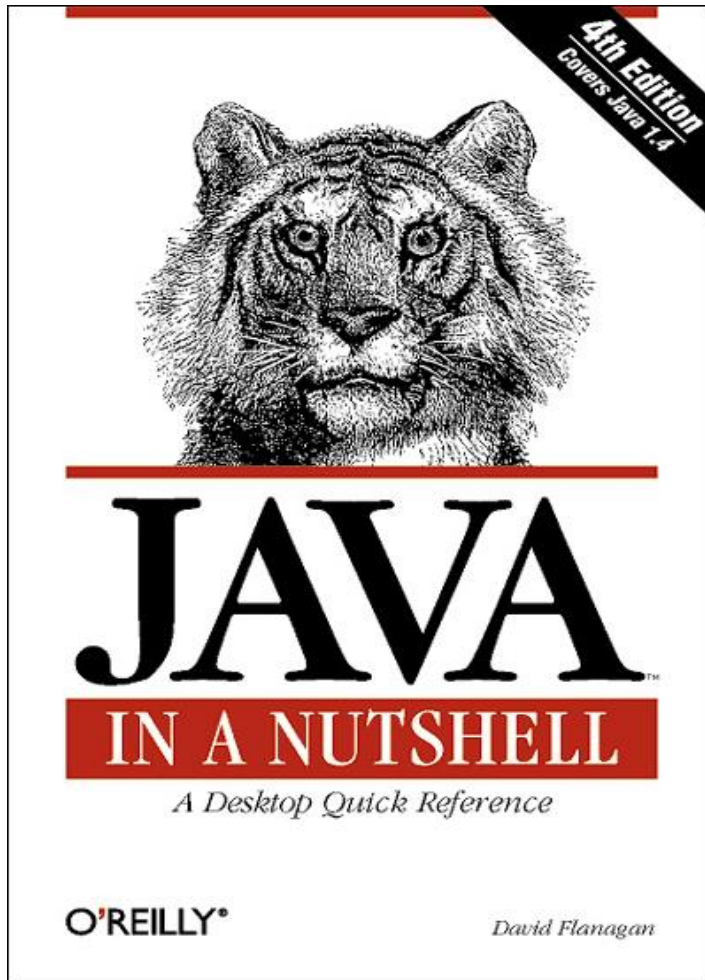
`System(System.out.println())`

`Math(Math.random())`

`String`

Library Method

책을 뒤져보는 방법



Currency

Returned By: java.text.DecimalFormat.getCurrency(), java.text.DecimalFormatSymbols.getCurrency(), java.text.NumberFormat.getCurrency(), Currency.getInstance()

Date

Java 1.0

java.util

cloneable serializable comparable

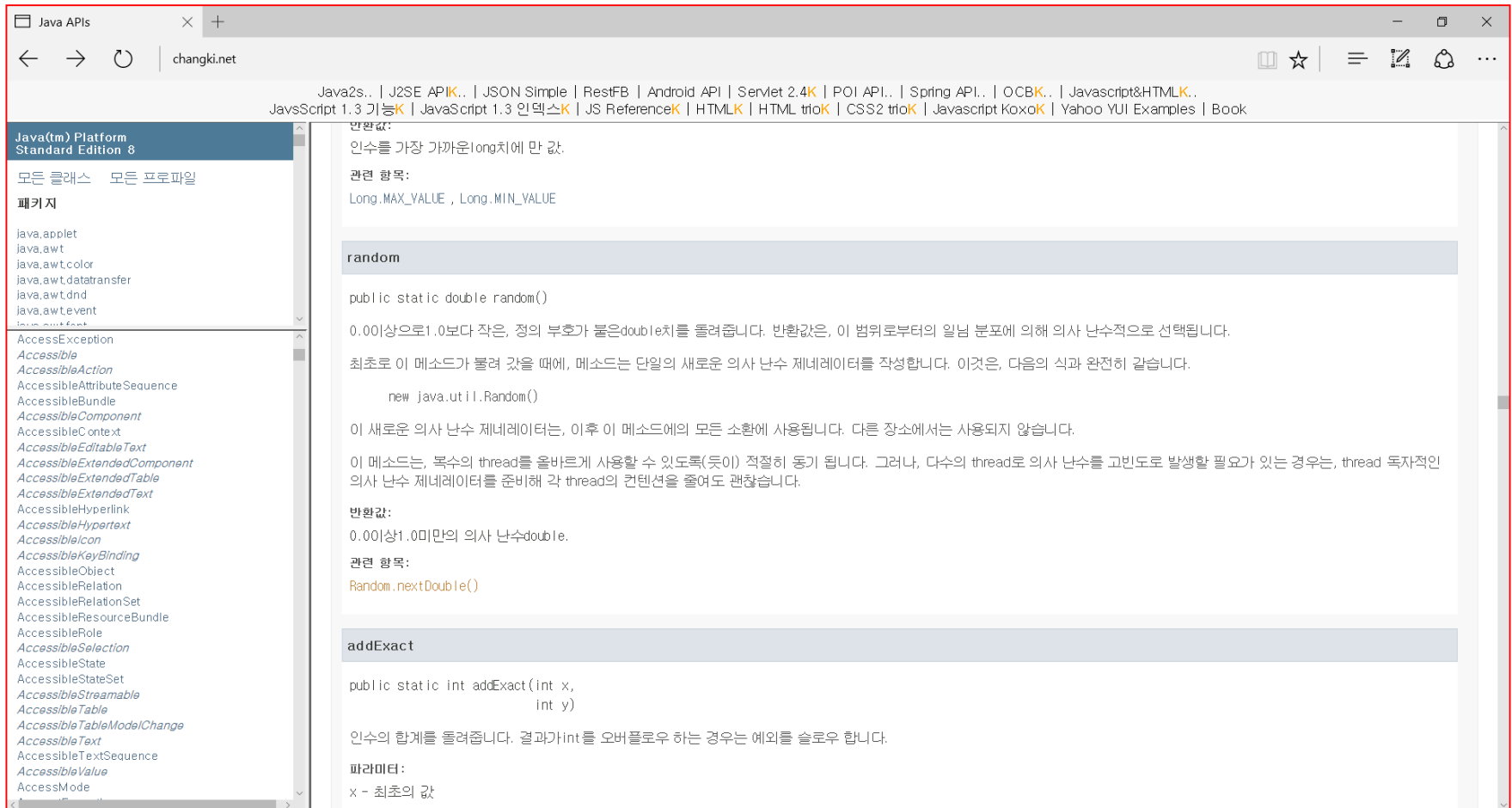
This class represents dates and times and lets you work with them in a system-independent way. You can create a `Date` by specifying the number of milliseconds from the epoch (midnight GMT, January 1st, 1970) or the year, month, date, and, optionally, the hour, minute, and second. Years are specified as the number of years since 1900. If you call the `Date` constructor with no arguments, the `Date` is initialized to the current time and date. The instance methods of the class allow you to get and set the various date and time fields, to compare dates and times, and to convert dates to and from string representations. As of Java 1.1, many of the date methods have been deprecated in favor of the methods of the `Calendar` class.



```
public class Date implements Cloneable, Comparable, Serializable {
    // Public Constructors
    public Date();
    public Date(long date);
    public Date(String s);
    public Date(int year, int month, int date);
    public Date(int year, int month, int date, int hrs, int min);
    public Date(int year, int month, int date, int hrs, int min, int sec);
    // Property Accessor Methods (by property name)
    public long getTime();
    public void setTime(long time);
    // Public Instance Methods
    public boolean after(java.util.Date when);
    public boolean before(java.util.Date when);
    1.2 public int compareTo(java.util.Date anotherDate);
    // Methods Implementing Comparable
    1.2 public int compareTo(Object o);
    // Public Methods Overriding Object
    1.2 public Object clone();
    public boolean equals(Object obj);
    public int hashCode();
    public String toString();
    // Deprecated Public Methods
    public int getDate();
    public int getDay();
    public int getHours();
    public int getMinutes();
    public int getMonth();
    public int getSeconds();
    public int getTimezoneOffset();
    public int getYear();
    public static long parse(String s);
    public void setDate(int date);
    public void setHours(int hours);
    public void setMinutes(int minutes);
    public void setMonth(int month);
```

Library Method

■ random() 메소드



The screenshot shows the Java API documentation for the `random()` method in the `java.util` package. The left sidebar lists various Java classes and packages, including `java.applet`, `java.awt`, and `java.util`. The main content area displays the following information:

- 반환값:** 인수를 가장 가까운 long치에 만 값.
- 관련 항목:** Long.MAX_VALUE, Long.MIN_VALUE
- random**
 - `public static double random()`
 - 0.00이상으로1.0보다 작은, 정의 부호가 붙은double치를 돌려줍니다. 반환값은, 이 범위로부터의 일임 분포에 의해 의사 난수적으로 선택됩니다.
 - 최초로 이 메소드가 불러 갔을 때에, 메소드는 단일의 새로운 의사 난수 제네레이터를 작성합니다. 이것은, 다음의 식과 완전히 같습니다.
 - ```
new java.util.Random()
```
  - 이 새로운 의사 난수 제네레이터는, 이후 이 메소드에의 모든 소환에 사용됩니다. 다른 장소에서는 사용되지 않습니다.
  - 이 메소드는, 복수의 thread를 올바르게 사용할 수 있도록(듯이) 적절히 동기 됩니다. 그러나, 다수의 thread로 의사 난수를 고빈도로 발생할 필요가 있는 경우는, thread 독자적인 의사 난수 제네레이터를 준비해 각 thread의 컨텐션을 줄여도 괜찮습니다.
- 반환값:** 0.00이상1.00미만의 의사 난수double.
- 관련 항목:** Random.nextDouble()
- addExact**
  - `public static int addExact(int x, int y)`
  - 인수의 합계를 돌려줍니다. 결과가int를 오버플로우 하는 경우는 예외를 슬로우 합니다.
- 파라미터:** x - 최초의 값

# Library Method

## ■ Method 사용법 해석(random(랜덤 값))

메소드 이름

```
public static double random(_)
```

Method를 사용할 때  
인수를 입력하지 않음

Method가 실행되고 나면  
double Type 결과를 반환  
(0.0 이상 1.0 미만 난수가 반환됨)

# Library Method

- Method 사용법 해석(abs: absolute value (절대값))

메소드 이름

```
public static double abs(double a)
```

메소드를 사용할 때  
double 타입 인수를  
하나 입력하여야 함

메소드가 실행되고 나면  
double 타입 결과를 반환  
(인수로 주어진 수의 절대값)

# Library Method

- Class를 사용하려면 Class의 전체 이름을 알아야 함
  - 예외: java.lang Package에 들어있는 Class
  - ArrayList의 전체 이름
    - java.util.ArrayList



# Library Method

- Class의 이름을 알려주는 방법
  - import 선언문을 사용

```
import java.util.ArrayList;
```

- 일일이 입력 방법

```
java.util.ArrayList list = new java.util.ArrayList();
public void go(java.util.ArrayList list) { }
public java.util.ArrayList foo() { }
```

# Library Method

- Method는 다음과 같은 형식의 헤더(header)로 선언

*return-type* name(*parameter-list*)

- return-type(반환 유형)은 Method에 의해 반환되는 값의 Data Type을 지정
- parameter-list(매개변수 목록)은 Method로 전달되는 인수의 개수(묵시적)와 유형(명시적)을 나열
- 매개변수 목록에서 이름을 형식 매개변수(formal parameter)또는 간단히 매개변수(parameter)라고 함

# Library Method

## ■ Method Header 작성

```
double area(double length, double width)
```

(형식) 매개변수  
(형식) 파라미터  
(formal) parameter

## ■ Parameter마다 Type을 작성해야 함

```
int example(int a, int b, double c) // OK
```

```
int example(int a, b, double c) // Error
```

# 사용자 정의 Method

- Method를 만드는 이유
  - Abstraction(추상화)
    - 세부적인 내용을 생략하고 핵심적인 개념만을 표현함
  - 작은 덩어리로 만들기(Modularization)
    - 큰 덩어리 하나로 만드는 대신 가능한 서로 독립적인 기능의 작은 덩어리 여러 개로 분해함
    - 작은 덩어리는 큰 덩어리보다 덜 복잡함
- 코드 재사용(Code reuse)

# 사용자 정의 Method

## ■ Method의 장점

### ■ 추상화

- 분할 정복 기법(Divide and Conquer)을 적용하여 작은 단위 Program을 구현하므로 개발과 관리가 용이
- 알아야 할 필요가 없는 세세한 작업이나 연산 등을 Method의 내부에 숨길 수 있음

### ■ 단위 Test

- Method 정의를 통해서 Code의 중복을 피할 수 있음
- 단위 Test가 가능함
- Program의 유지보수(maintenance)나 수정이 용이함

### ■ 재 활용

- 정의한 Method는 재 사용(Software reusability)할 수 있음

# 사용자 정의 Method

- Method 작성 지침
  - Code를 재사용 할 수 있음
    - 반복적인 Code를 줄이고 Code의 관리가 용이함
    - 반복적으로 수행되는 여러 문장을 Method로 작성함
  - 하나의 Method는 한 가지 기능만 수행하도록 작성하는 것이 좋음
  - 관련된 여러 문장을 Method로 작성함

# 사용자 정의 Method



## <입력>

Method를 수행하기 위하여 필요한 Data는 여러 개가 있을 수 있음. **다수일수록 Method는 유연성이 있음**

예) **약** 사와라

**돈**을 주면서 **무슨** 약 사와라

**돈**을 주면서 **무슨** 약을 **어디서** 사와라

## <출력>

Method의 수행결과를 수행을 요청한 Program에 반환하는 요소 (**오직 하나만 전달 가능**)

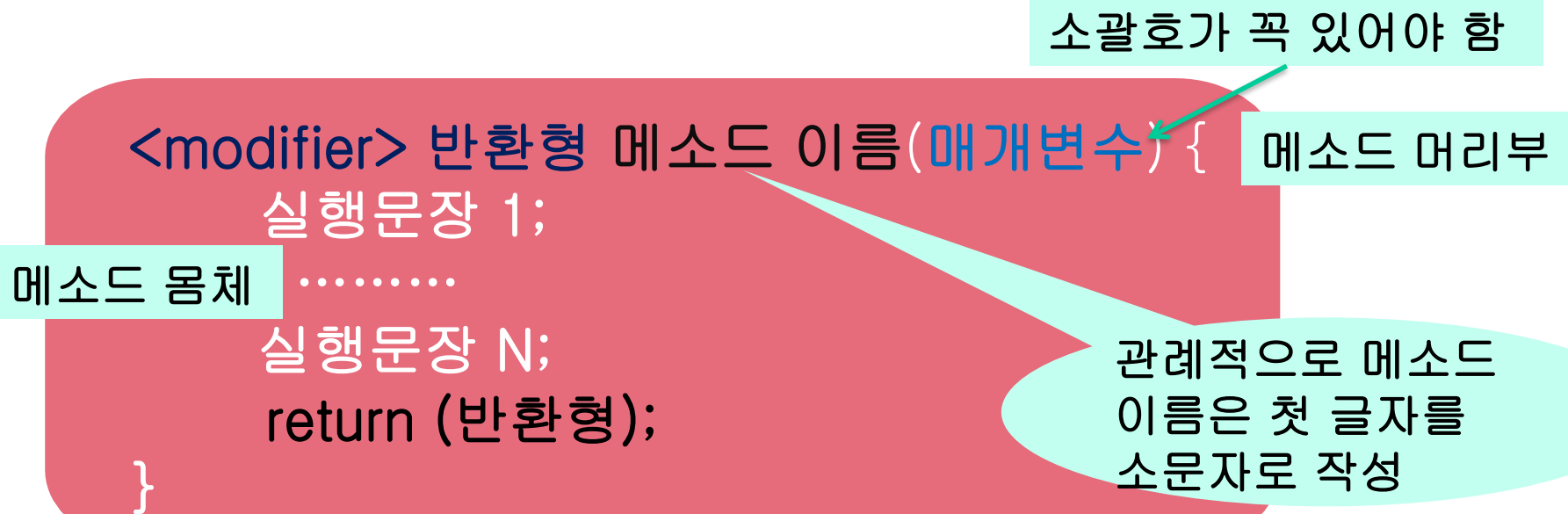
# 사용자 정의 Method

- Defining Methods(메소드 정의 방법)
- Calling Methods(메소드 호출 방법)
- Overloading(메소드 다중 정의 방법)
- Passing Parameters(매개 변수 전달 방법)



# 사용자 정의 Method

## ■ Method의 구조



Method는 Method의 머리부와 몸체로 구성  
Method 몸체에는 Method의 시작 부분에는 열린 중괄호({)를  
Method의 끝부분에는 닫힌 중괄호(})를 사용하고  
열린 중괄호와 닫힌 중괄호 사이에 다양한 명령문들을 작성

# 사용자 정의 Method

- Member 변수, Method 접근 지정자
  - 여러 종류가 있으며 어디서나 자유롭게 접근할 수 있도록 public으로 지정
  - 접근 지정자는 생략할 수도 있음 (default)

| 접근 지정자    | 클래스 | 같은 패키지 | 서브 클래스 | 모든 클래스 |
|-----------|-----|--------|--------|--------|
| private   | ○   |        |        |        |
| default   | ○   | ○      |        |        |
| protected | ○   | ○      | ○      |        |
| public    | ○   | ○      | ○      | ○      |

# 사용자 정의 Method

- Method가 정상적으로 종료되는 경우
  - Method의 블럭{}의 끝에 도달했을 때
  - Method의 블럭{}을 수행 도중 return 문을 만났을 때
- return 문
  - 현재 실행 중인 Method를 종료하고 호출한 Method로 되돌아감

1. 반환값이 없는 경우 - return문만 써주면 된다.

```
return;
```

2. 반환값이 있는 경우 - return문 뒤에 반환값을 지정해 주어야 한다.

```
return 반환값;
```

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

타입이 일치해야한다.

# 사용자 정의 Method

## ■ return 문 주의사항

- 반환 값이 있는 Method는 return 문이 있어야 함

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



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

## ■ return문의 개수는 최소화하는 것이 좋음

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



```
int max(int a, int b) {
 int result = 0;
 if(a > b)
 result = a;
 else
 result = b;
 return result;
}
```

# 사용자 정의 Method

## ■ return 문

- 수행을 마친 후 결과 값을 되돌려 줄 때 사용
- 반환되는 값의 Data Type이 Method의 Data Type이 됨


```
static double average(int firstNumber, int secondNumber) {

 double result;

 result = (firstNumber + secondNumber) / 2.0;

 return result;

}
```



# 사용자 정의 Method

## ■ Method에 주석 달기

- Class에 대해 설명하는 주석을 작성 하듯이 각 Method에 대해서도 주석을 작성함

```
/**
 * 섭씨 온도를 화씨 온도로 변환한다.
 * @param c 섭씨 온도.
 * @return 화씨 온도.
 */

public static double convert(double c) {
 double f = c * 9.0 / 5.0 + 32.0;
 return f;
}
```

이 메소드가 무슨 일을 하는지  
설명해 준다

파라미터 c가 어떤 용도로  
사용되는지 설명해 준다

반환되는 값이 무엇인지  
설명해준다

# 사용자 정의 Method

- Parameter가 2개 이상이면 각 Parameter마다 설명을 따로 작성

```
/**
 * 주어진 두 수 사이의 자연수의 합을 구한다.
 * @param from 작은 수.
 * @param to 큰 수.
 * @return from부터 to까지의 자연수의 합.
 */
public static int add(int from, int to) {
 ...
 return sum;
}
```

# 사용자 정의 Method

## ■ Method Parameter의 개수

### ■ Method에 여러 개의 값을 넘겨줄 수 있음 (7~8개 정도)

```
public class TwoParameters {
 public static void main(String[] args) {
 int result = add(1, 3); // 두 개의 값(인자)을 넘겨줌
 System.out.println("1 + 3 = " + result);

 System.out.println("5 + 3 = " + add(5, 3));
 }
 public static int add(int i, int j) { // 파라미터가 두 개!
 // int 타입 i, int 타입 j
 return i + j;
 }
}
```

int 타입 값이 반환됨

값이 복사됨



# 사용자 정의 Method

## ■ double Type의 값을 반환하므로 double Type Method

```
public class CelciusToFarenheit {
 public static void main(String[] args) {
 System.out.print("C = -40.0 --> ");
 System.out.print("F = ");
 double fahr = convert(-40.0);
 System.out.print(fahr);
 }
 public static double convert(double c)
 {
 double f = c * 9.0 / 5.0 + 32.0;
 return f;
 }
}
```

Method로부터 반환된 값을 저장  
할 변수는 반환되는 값의 Type  
에 맞춰 선언해야 함

반환(return)되는 값의 Type  
을 지정

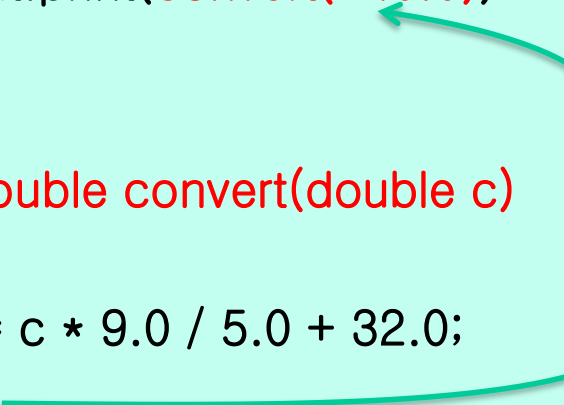
복사

온도 변환 계산을 하고  
얻어진 화씨 온도를 **반환**  
하는 Method를 선언함

# 사용자 정의 Method

```
public class CelciusToFarenheit {
 public static void main(String[] args) {
 System.out.print("C = -40.0 --> ");
 System.out.print("F = ");
 System.out.print(convert(-40.0));
 }

 public static double convert(double c)
 {
 double f = c * 9.0 / 5.0 + 32.0;
 return f;
 }
}
```



convert 메소드 실행 후  
반환되는 값이  
출력됨

# 사용자 정의 Method

- 실제로 반환하는 값의 Data Type이 선언과 일치해야 함

Method의 반환 Type 선언

```
public static int add(int i, int j)
{
 return i + j;
}
```

int Type 값을 반환한다고 선언했으면  
실제로 int Type 값을 반환해야 함

Method의 반환 Type 선언

```
public static double convert(double c)
{
 double f = c * 9.0 / 5.0 + 32.0;
 return f;
}
```

double Type 값을 반환한다고 선언했으면  
실제로 double Type 값을 반환해야 함

# 사용자 정의 Method

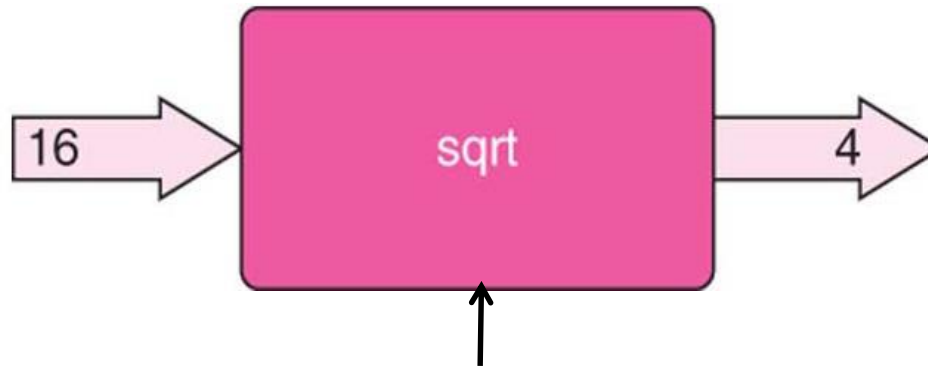
- JDK 5부터 가변 길이 인수(variable-length arguments) 사용 가능

```
class VarArgs {
 void sub(int... v) {
 System.out.println("인수의 개수 : " + v.length);
 for (int x : v)
 System.out.print(x + " ");
 System.out.println();
 }
}
```

매개변수에 점 3개가 오는데 이것은 varargs 또는 가변인자라고 한다.  
그 의미는 “int 객체가 0개부터 여러 개까지 매개변수로 올 수 있다”라는 것이다

# 사용자 정의 Method

- 제곱근을 구하는 작업을 하는 메소드, sqrt : square root(제곱근)

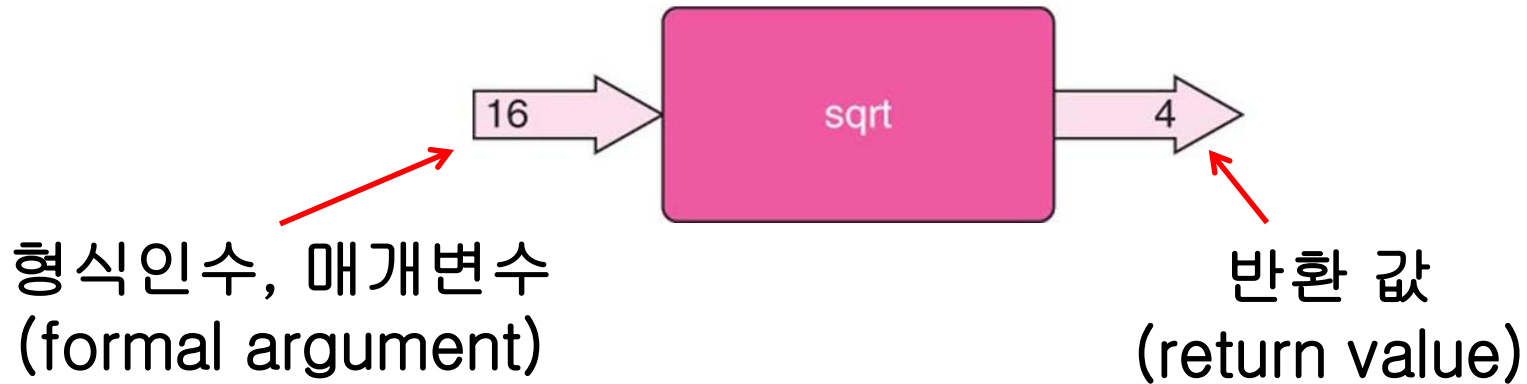


박스 안에 작업을 수행하는  
문장들이 들어 있음

- 사용자 입장에서는 직접 문장을 작성할 필요가 없음
- 사용법을 확인하여 사용하면 됨

# 사용자 정의 Method

## ■ 메소드 사용 예



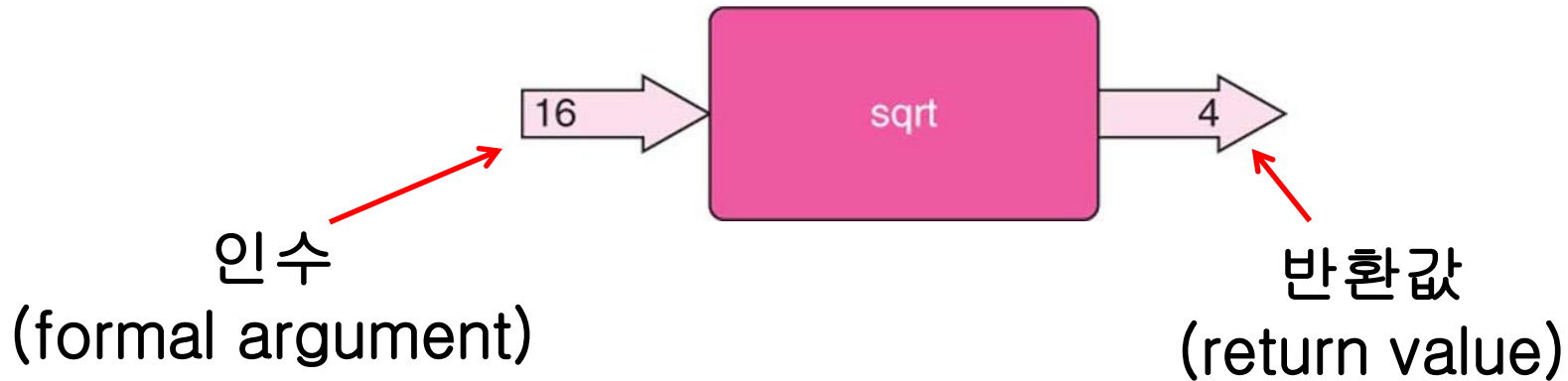
```
double result;
result = sqrt(16.0);
```

반환값  
(return value)

실인수, 매개변수  
(actual argument)

# 사용자 정의 Method

## ■ 메소드 사용 예



```
System.out.println(sqrt(16.0));
```

메소드의  
반환값이 출력됨

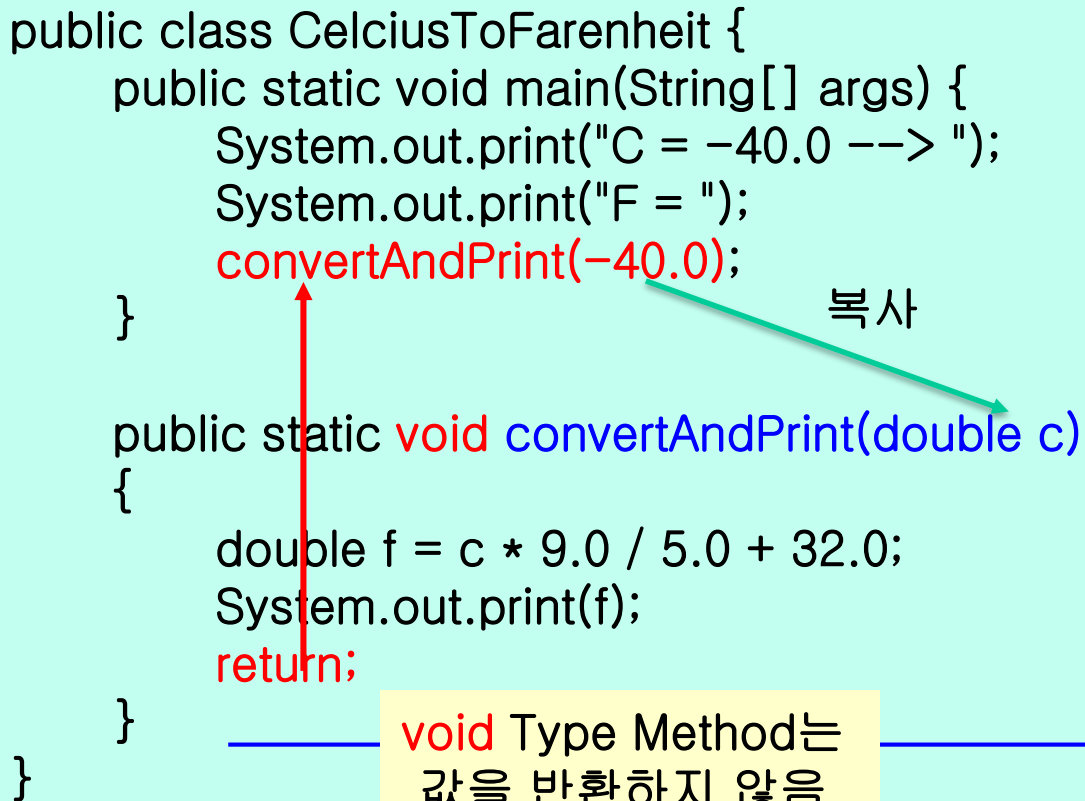
인수  
(actual argument)

# 사용자 정의 Method

- Method의 반환 값이 없는 경우 “리턴이 없다”라는 것을 적극적으로 표현하기 위해서 **void**라는 특별한 자료 형이 있음

```
public class CelciusToFarenheit {
 public static void main(String[] args) {
 System.out.print("C = -40.0 --> ");
 System.out.print("F = ");
 convertAndPrint(-40.0);
 }

 public static void convertAndPrint(double c)
 {
 double f = c * 9.0 / 5.0 + 32.0;
 System.out.print(f);
 return;
 }
}
```



복사

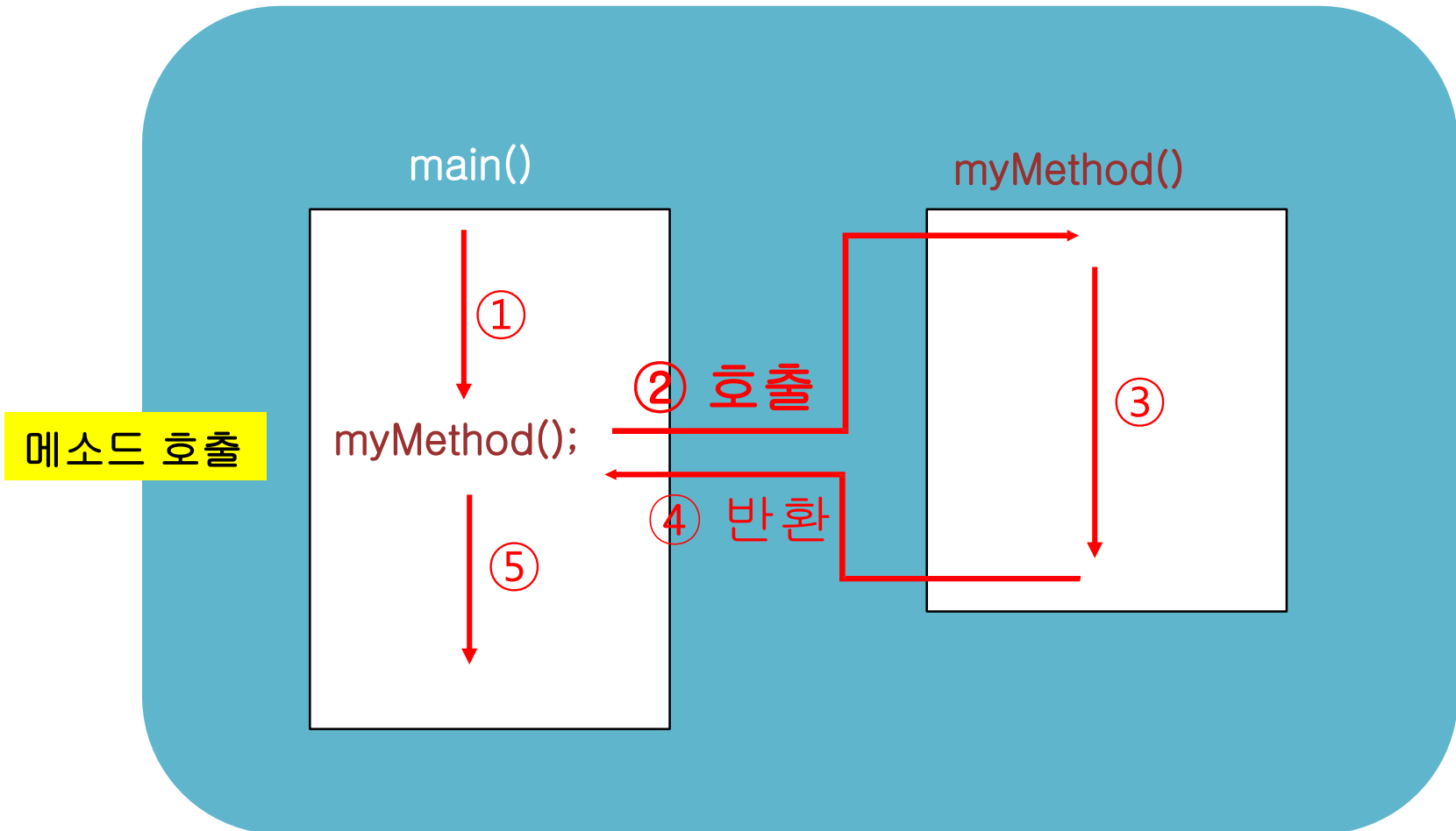
온도 변환 계산을 하고  
얻어진 화씨 온도를 출력  
하는 Method를 선언함

**void** Type Method는  
값을 반환하지 않음



# 사용자 정의 Method

## ■ main() 메소드의 호출



# 사용자 정의 Method

- main() 메소드와 사용자 정의 메소드와의 관계

```
public static void main(String[] args) {
 design();
 construct();
 paint();
 operate();
}
```

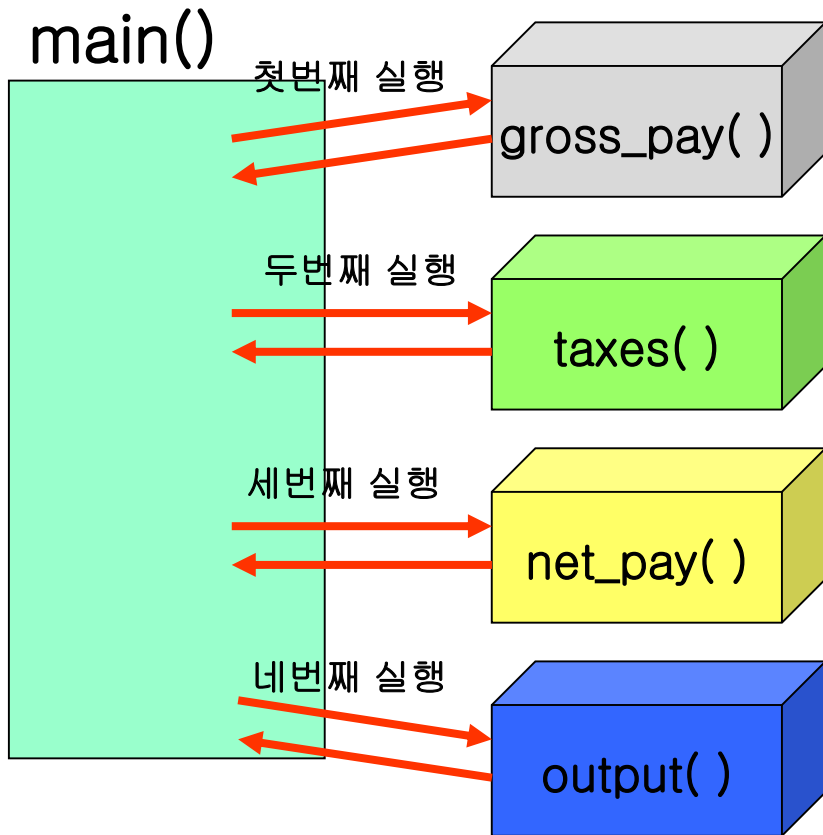
추상화된 단위 작업들로  
전체 작업을 서술  
(Divide and Conquer)

```
static void design() { ... 미주왈 ... }
static void construct() { ... 고주왈 ... }
static void paint() { ... 예쁘게 ... }
static void operate() { ... 신나게 ... }
```

- ✓ 각 메소드에서 각 단위 작업의 세부적인 내용을 구현
- ✓ 각 메소드를 따로따로 Test함 (Unit Test)
- ✓ 작은 단위 프로그램은 Test, Debugging 쉬움

# 사용자 정의 Method

- main() Method는 모든 다른 Method의 실행을 제어함
  - main() 메소드에서 호출하지 않으면 실행되지 않음



메소드 헤더

```
public static void main(String[] args)
{
 gross_pay();
 taxes();
 net_pay();
 output();
}
```

메소드 본체

# 사용자 정의 Method

## ■ Method 호출 방법

### ■ Library Class Method 접근 형식(static으로 선언)

클래스 이름.클래스 메소드 이름(실 매개변수)

### ■ 다른 Class의 일반 Method 접근 형식

객체이름.객체 메소드 이름(실 매개변수)

### ■ 같은 Class의 Method(static)인 경우

메소드 이름(실 매개변수)

### ■ 실 매개변수

■ 메소드에 넘겨주는 매개변수로 형식 매개변수와 Data형과 개수와 순서가 일치해야 함

■ 실 매개변수 값 -> 형식 매개변수에 값 복사

# 사용자 정의 Method

- Method의 매개변수 전달 방법
  - Call By Name
    - 메소드의 이름에 의해 호출되는 메소드로 **특정 매개변수 없이 실행**
  - Call By Value
    - 메소드를 이름으로 호출할 때 **특정 매개변수를 전달하여 그 값을 기초로 실행하는 메소드**
    - 실 매개변수가 형식 매개변수로 복사됨
    - 실 매개변수는 변경되지 않음
  - Call By Reference
    - 메소드 호출 시 매개변수로 사용되는 값이 특정 위치를 참조하는 reference 변수
    - 실 매개변수로 참조형(배열 명, 사용자 정의 객체)을 전달 함 (주소가 전달됨)
    - 실 매개변수가 메소드에 의해 변경 가능함

# 사용자 정의 Method 예제 1

```
public class CallByName {
 private static void print_star() { // 메소드 정의
 System.out.println("*****");
 }

 public static void main(String[] args) {
 print_star(); // 메소드 호출(call by name)
 System.out.println("가나다라");
 print_star(); // 메소드 호출(call by name)
 print_star(); // 메소드 호출(call by name)
 System.out.println("마바사아");
 System.out.println("자차카타");
 print_star(); // 메소드 호출(call by name)
 }
}
```

# 사용자 정의 Method 예제 2

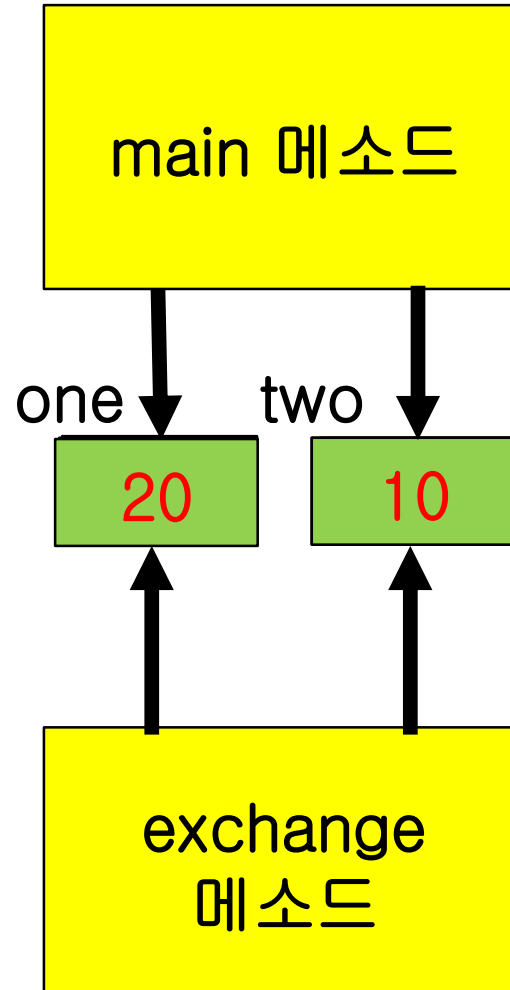
```
public class Main {
 static int one, two; // 멤버 변수

 private static void exchange(){ // 메소드 정의
 int temp;
 temp = one;
 one = two;
 two = temp;
 }

 public static void main(String[] args) {
 Scanner keyboard = new Scanner(System.in);
 System.out.print(" 두수를 입력 : ");
 one = keyboard.nextInt();
 two = keyboard.nextInt();
 System.out.printf("one = %d, two = %d\n", one, two);
 exchange(); //call by name
 System.out.printf("one = %d, two = %d\n", one, two);
 }
}
```

# 사용자 정의 Method 예제 2

## ■ Call by Name 실행 방법





# 사용자 정의 Method 예제 3

```
public class CallByValue {
 public static void print_star(String str, int x) { // 메소드 정의
 for(int i = 0; i < x; i++) {
 System.out.print(str);
 }
 System.out.println();
 }
 public static void main(String[] args) {
 print_star("*", 9); // 메소드 호출 (call by name)
 System.out.println("가나다라");
 print_star("!", 7); // 메소드 호출 (call by name)
 print_star("$", 7); // 메소드 호출 (call by name)
 System.out.println("마바사아");
 System.out.println("자차타카");
 print_star("*", 9); // 메소드 호출 (call by name)
 }
}
```

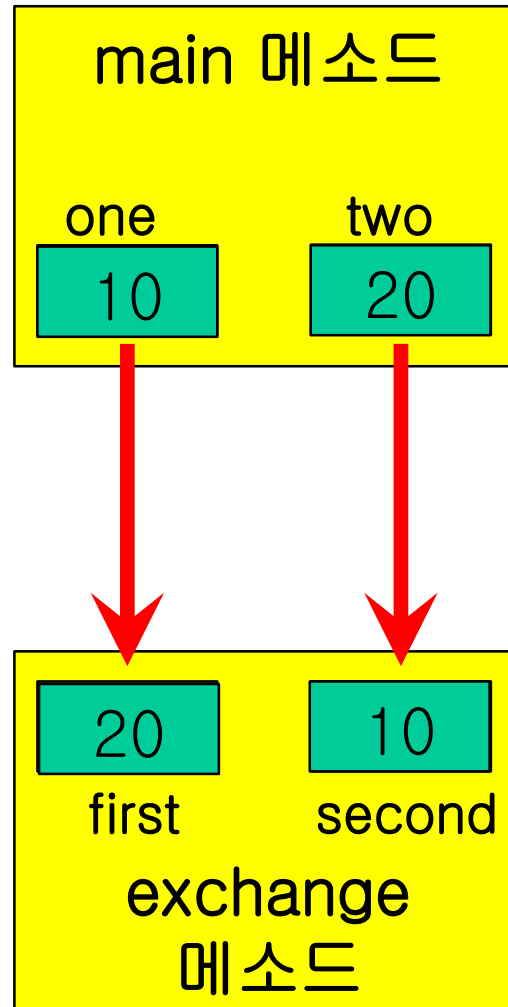
# 사용자 정의 Method 예제 4

```
static void exchange(int first, int second){
 int temp;
 temp = first;
 first = second;
 second = temp;
}
```

```
public static void main(String[] args) {
 Scanner keyboard = new Scanner(System.in);
 int one, two;

 System.out.print(" 두수를 입력 : ");
 one = keyboard.nextInt();
 two = keyboard.nextInt();
 System.out.printf("one = %d, two = %d\n", one, two);
 exchange(one, two); //Call by value
 System.out.printf("one = %d, two = %d\n", one, two);
}
```

# 사용자 정의 Method 예제 4



# 사용자 정의 Method 예제 5

- 메소드를 호출할 때 값을 넘겨줄 수 있음 (call by value)

```
public class Test {
 public static void main (String[] args) {
 printNumber(3);
 }

 private static void printNumber(int number) {
 System.out.println();
 System.out.println("*****");
 System.out.printf("%12d", number);
 System.out.println("*****");
 }
}
```

실 인자(argument)

```

 3

```

복사

메소드를 호출할 때  
실 인자(argument)가  
형식 매개변수로 복사됨

number는 int 타입 변수!  
이 변수를 메소드 형식 매개변수(parameter)라고 함

# 사용자 정의 Method 예제 5

## ■ Parameter 변수의 탄생과 소멸

```
public class Test {
```

Program 실행 점이 변수의 유효 범위에 들어 올 때 변수가 만들어지고  
유효 범위를 벗어나면 그 변수는 사라짐 (Memory 자체가 없어짐)

printNumber 메소드가 시작될 때 변수 number가 만들어지고  
printNumber 메소드가 끝날 때 변수 number가 사라짐

```
 public static void printNumber(int number) {
 System.out.println();
 System.out.println("*****");
 System.out.printf("%12d", number);
 System.out.println("*****");
 }
}
```

# 사용자 정의 Method 예제 5

- 메소드를 호출할 때 값을 넘겨줄 수 있음 (call by value)

```
public class Test {
 public static void main (String[] args) {
 printNumber(3);
 }
}
```

```
public static void printNumber(int number) {
 System.out.println();
 System.out.println("*****");
 System.out.printf("%12d", number);
 System.out.println("*****");
}
```

number의 유효 범위

변수의 유효 범위는 변수가 선언된 곳으로부터  
변수가 선언된 블록의 끝까지!

# 사용자 정의 Method 예제 5

■ 변수도 인자가 될 수 있음

■ 인자와 변수의 Data Type이 일치함이 원칙 임

```
public class Test {
 public static void main (String[] args) {
 Scanner input =new Scanner(System.in);
 int n = input.nextInt(); // 변수 사용
 printNumber(n);
 }
}
```

일치하는 타입이 원칙 임

```
public static void printNumber(int number) {
 System.out.println();
 System.out.println("*****");
 System.out.printf("%12d", number);
 System.out.println("*****");
}
}
```

# 사용자 정의 Method 예제 5

## ■ 인자와 변수의 Data Type

- 작은 타입(int) 값은 큰 타입(double) 변수에 저장할 수 있음

```
int i = 2;
double x = i; // i 값이 double 타입으로 자동 변환
 되어 x에 저장됨
 // x에는 2.0이 저장됨
```

- 큰 타입(double) 값은 작은 타입(int) 변수에 저장할 수 없음

```
double x = 2.0;
int i = x; // 에러! 컴파일 안 됨
```



# 사용자 정의 Method 예제 5

## ■ 인자와 변수의 타입

```
public class Test {
 public static void main (String[] args) {
 Scanner input =new Scanner(System.in);
 int n = input.nextInt();
 printNumber(n);
 }
}
```

작은 타입 값을 큰 타입 변수에 복사  
(큰 타입 값(3.0)으로 자동 변환됨)

```
public static void printNumber(double number) {
 System.out.println();
 System.out.println("*****");
 System.out.printf("%12.0f", number);
 System.out.println("*****");
}
}
```

# 사용자 정의 Method 예제 5

## ■ 인자와 변수의 타입

```
public class Test {
 public static void main (String[] args) {
 Scanner input =new Scanner(System.in);
 double n = input.nextDouble();
 printNumber(n);
 }

 public static void printNumber(int number) {
 System.out.println();
 System.out.println("*****");
 System.out.printf("%12d", number);
 System.out.println("*****");
 }
}
```

큰 타입 값을 작은 타입 변수에 복사  
에러!

# 사용자 정의 Method 예제 6

```
public class InfiniteSeries {
 public static void main(String[] args) {
 final double MAX = 1E8; // n을 얼마까지나 증가시킬 것인가?
 double sum;
 for (double n = 1; n <= MAX; n *= 10.0) {
 // 1 + 1/2 + 1/3 + 1/4 + 1/5 + ... + 1/n 를 출력한다.
 sum = addInverses(n); // n까지 역수들의 합을 구함
 System.out.println("n = " + n + " : " + "sum = " + sum);
 }
 }
 // to까지의 역수들의 합을 구하는 메소드 (1/1 + 1/2 + 1/3 + ... + 1/to)
 static double addInverses(double to) {
 double sum = 0.0;
 for (double i = 1; i <= to; i++)
 sum += 1.0/i;
 return sum;
 }
}
```

인수

메소드가 호출될 때 인수 n이  
파라미터 to로 복사된다.

(형식)파라미터

지역변수는 자기가 선언된 블록 안에서만  
유효하므로 sum과 sum은 서로 관계 없는  
별개의 변수들이다

# 도입 예제

- 섭씨 온도를 화씨 온도로 변환하는 Program을 메소드를 이용하여 작성해보자

```
public static void main(String[] args) {
 System.out.print("C = -40.0 --> ");
 System.out.print("F = ");
 double f = -40.0 * 9.0 / 5.0 + 32.0;
 System.out.println(f);

 System.out.print("C = 30.0 --> ");
 System.out.print("F = ");
 f = 30.0 * 9.0 / 5.0 + 32.0;
 System.out.println(f);

 System.out.print("C = 40.0 --> ");
 System.out.print("F = ");
 f = 40.0 * 9.0 / 5.0 + 32.0;
 System.out.println(f);
}
```

```
C = -40.0 --> F = -40.0
C = 30.0 --> F = 86.0
C = 40.0 --> F = 104.0
```

# 도입 예제

- 메소드의 대상을 어떤 기준으로 선정하며 만들면 장점이 무엇인가 ?
  - 반복적인 사용
  - 하나의 단일 기능
  - 재 사용성이 높음
- Data와 Information을 정의
  - Data의 개수는 상관 없음
  - Information이 하나일 것

# 도입 예제

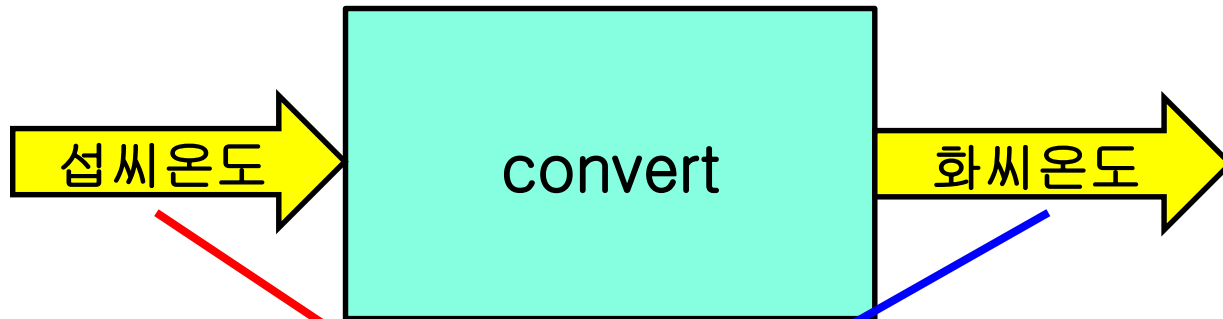
- Service를 제공자와 Service를 이용자 사이의 약속
  - Service 제공자 - 메소드
  - Service 이용자 - 메소드를 호출하는 것
- 메소드에 적어 주는 주석은 인터페이스를 규정
- Service를 제공하는 자 (메소드)
  - 약속한 범위 내에서만 충실하게 작동하면 됨
  - 약속을 벗어난 조건에서는 아무 것도 보장하지 않음

# 도입 예제

- 사용자 정의 method를 정의하는 절차
  - Method의 기능에 맞는 이름을 정함
    - Method가 어떤 기능을 수행하는가?
    - Method의 이름 :
  - Parameter를 정의
    - Method에 필요한 입력 값 들은 몇 개이며, Data Type은 무엇인가?
    - 매개변수 :
  - 반환 값(출력)의 형태를 정의
    - 함수가 실행된 후의 결과는 몇 개이며 Data Type은 무엇인가 ?
    - 반환 값의 형태 :
- Method의 몸체를 작성
  - Method 수행하는 기능을 작성

# 도입 예제

- 섭씨 온도를 화씨 온도로 변환하는 convert() 메소드를 만들어보자



```
private static double convert(double temp) {
 double result;
 result = temp * 9.0 / 5.0 + 32.0;

 return result;
}
```



# 도입 예제

```
public static void main(String[] args) {
 System.out.print("C = -40.0 --> ");
 System.out.print("F = ");
 System.out.println(convert(-40.0));

 System.out.print("C = 30.0 --> ");
 System.out.print("F = ");
 System.out.println(convert(30.0));

 System.out.print("C = 40.0 --> ");
 System.out.print("F = ");
 System.out.println(convert(40.0));
}
```

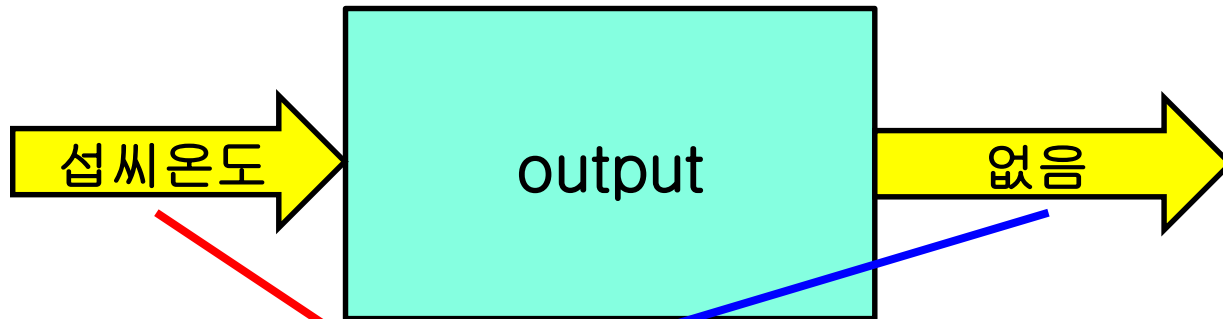


Futuristic Innovator

京福大學校  
KYUNGBOK UNIVERSITY

# 도입 예제

- 반복되는 부분을 output() 메소드로 만들어보자



```
private static void output(double temp) {
 System.out.printf("C = %4.1f --> ", temp);
 System.out.print("F = ");
 System.out.println(convert(temp));
}
```

# 도입 예제

```
public static void main(String[] args) {
 output(-40.0);
 output(30.0);
 output(40.0);
}
```

# 원의 면적

- 원의 반지름(radius)을 입력 받아 원의 면적(area)을 구하는 `circleArea( )` 메소드를 정의하고, 원의 면적을 구하는 프로그램을 작성하여라.
- 단 원의 반지름은 정수임

# 원의 면적

■ 원의 면적을 구하는 circleArea() 메소드를 작성하라

■ 알고리즘

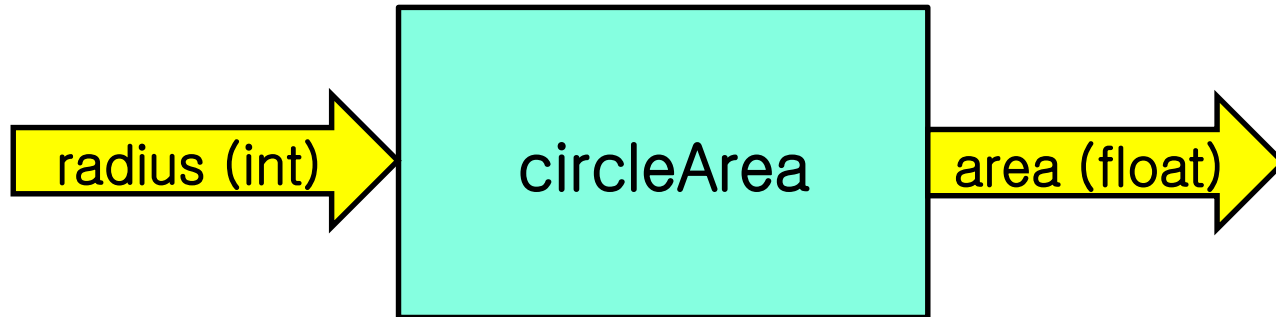
1. 원주율  $\pi$ 를 저장하는 변수를 선언
2. 원의 면적을 구하는 공식 구현
3. 구한 면적 값을 반환

■ 메소드 이름 생각

■ 반환 값의 Data Type 생각

# 원의 면적

## ■ circleArea( ) 메소드 정의



```
private static float circleArea(int 반지름) {
 final float PI = 3.141592f;
 float area;

 area = PI * 반지름 * 반지름;

 return area;
}
```

# 원의 면적

```
public static void main(String[] args) throws IOException {
 Scanner keyboard = new Scanner(System.in);
 int radius;

 while(true) {
 System.out.print(" 원의 반지름 입력 : ");
 radius = keyboard.nextInt();
 if (radius > 0)
 break;
 else {
 System.err.println("Error - 1 이상의 숫자 이어야 함");
 System.in.read();
 }
 }

 System.out.printf("원의 반지름 = %,d\n", radius);
 System.out.printf("원의 면적 = %,2f\n", circleArea(radius));
}
```

# 원의 면적[심화]

- 원의 반지름이 실수일 때를 고려하여 작성하자.

```
float circleArea(float radius) {
 final float PI = 3.141592f;
 float area;

 area = PI * radius * radius;

 return area;
}
```



# 원의 둘레 [심화]

- 원의 둘레를 계산하는 circumference() 메소드 정의



```
private static float circumference(int 반지름) {
 final float PI = 3.141592f;
 float length;

 length = 2 * PI * 반지름;

 return length;
}
```

# 원의 둘레 [심화]

```
public static void main(String[] args) throws IOException {
 Scanner keyboard = new Scanner(System.in);
 int radius;

 while(true) {
 System.out.print("원의 반지름 입력 : ");
 radius = keyboard.nextInt();
 if (radius > 0)
 break;
 else {
 System.err.println("Error - 1 이상의 숫자 이어야 함");
 System.in.read();
 }
 }
 System.out.printf("원의 반지름 = %,d Cm\n", radius);
 System.out.printf("원의 면적 = %,2f \n", circleArea(radius));
 System.out.printf("원 둘레 = %,2f Cm\n", circumference(radius));
}
```

# 원 [심화]

## ■ main() 메소드 재 정의 방법

- Program의 흐름을 고려하여 입력 -> 처리 -> 출력을 구현 하도록 메소드로 구분

```
public static void main(String[] args) throws IOException {
 int radius;
 float area;
 float length;

 radius = inputData();
 area = circleArea(radius);
 length = circumference(radius);

 output(radius, area, length);
}
```

# 원 [심화]

```
private static int inputData() throws IOException {
 Scanner keyboard = new Scanner(System.in);
 int radius;

 while(true) {
 System.out.print(" 원의 반지름 입력 : ");
 radius = keyboard.nextInt();
 if (radius > 0)
 break;
 else {
 System.out.println("Error-1 이상의 숫자 이어야 함");
 System.in.read();
 }
 }

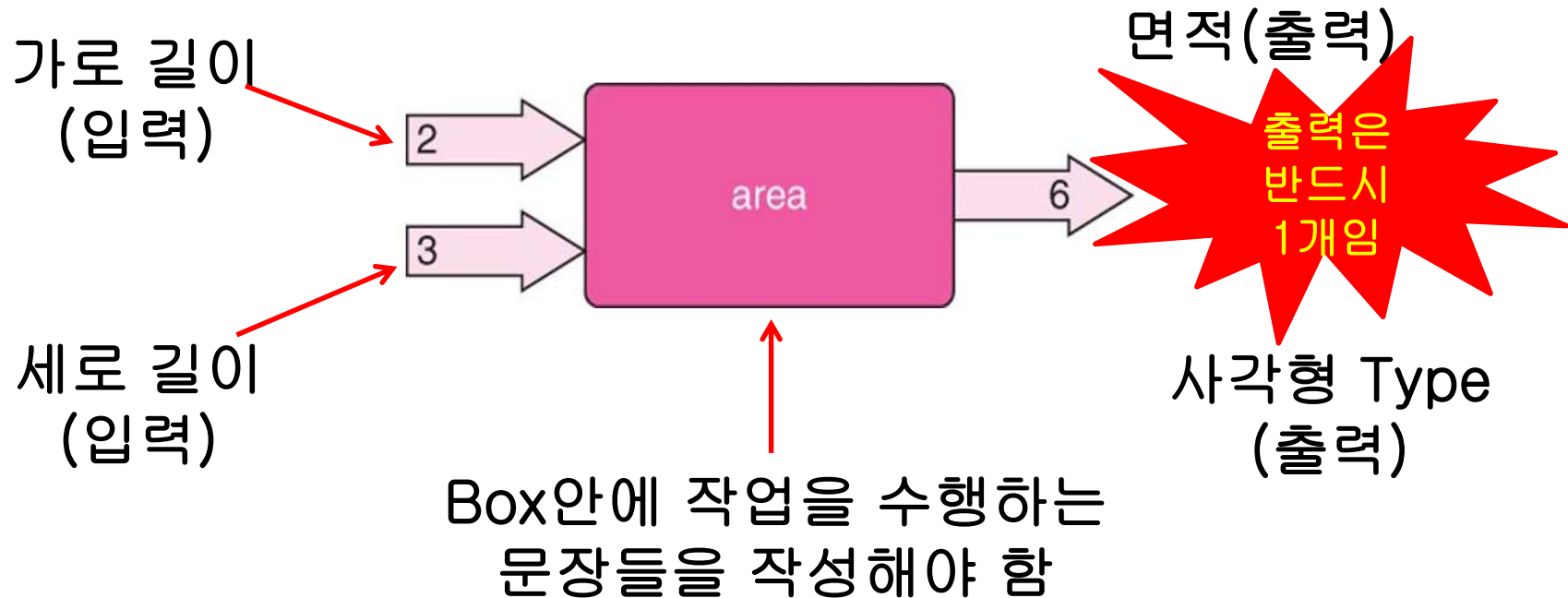
 return radius;
}
```

# 원 [심화]

```
private static void output(int radius, float area, float length) {
 System.out.printf("원의 반지름 = %,d Cm\n", radius);
 System.out.printf("원의 면적 = %,2f\n", area);
 System.out.printf("원 둘레 = %,2f\n Cm", length);
}
```

# 사각형 면적

- 사각형의 가로(garo)와 세로(sero)로부터 면적(area)을 구하는 작업을 하는 메소드 area()를 정의 하여보자



# 사각형 면적

## ■ 정의한 area() Method

메소드  
Data Type

메소드 이름

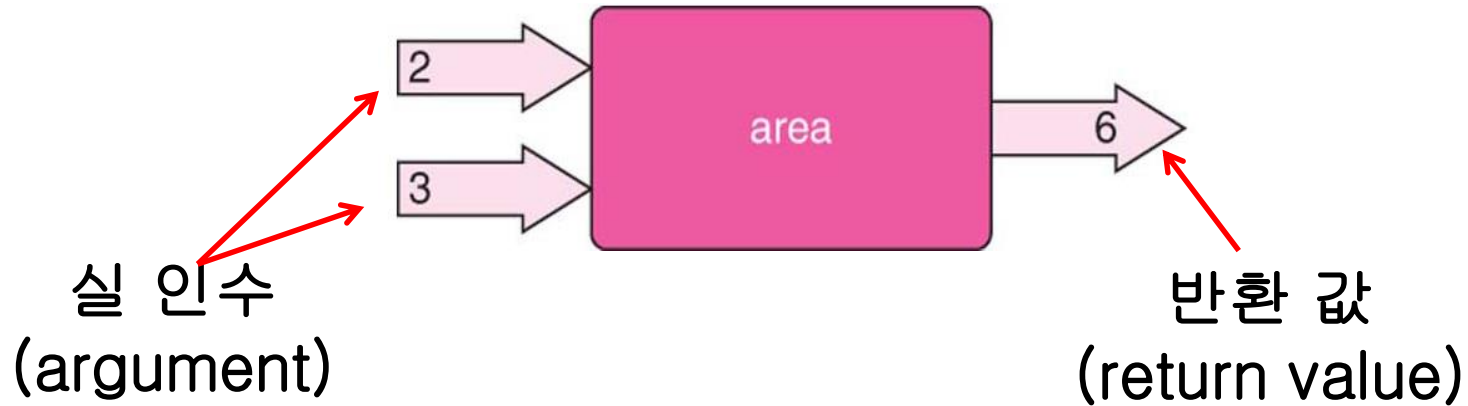
가(형식) 인수  
(parameter)

```
private static long area(int garo, int sero) {
 long area = 0;
 area = (long) garo * sero;
 return area;
}
```

메소드 Type은  
반환 값의 Data  
type

# 사각형 면적

## ■ 사용 방법



```
long result;
result = area(2, 3);
```

`반환 값 (return value)` points to `result`.  
`실 인수 (argument)` points to `2` and `3`.



# 사각형 면적

```
public static void main(String[] args) throws IOException {
 Scanner keyboard = new Scanner(System.in);
 int garo, sero;
 long area;

 while(true) {
 System.out.print(" 사각형의 가로와 세로 길이 입력 : ");
 garo = keyboard.nextInt();
 sero = keyboard.nextInt();
 if (garo > 0 && sero > 0)
 break;
 else {
 System.out.println("Error - 1 이상의 숫자 이어야 함");
 System.in.read();
 }
 }
}
```

# 사각형 면적

```
area = area(garo, sero);
```

```
System.out.printf("사각형의 가로 길이 = %,d Cm\n", garo);
```

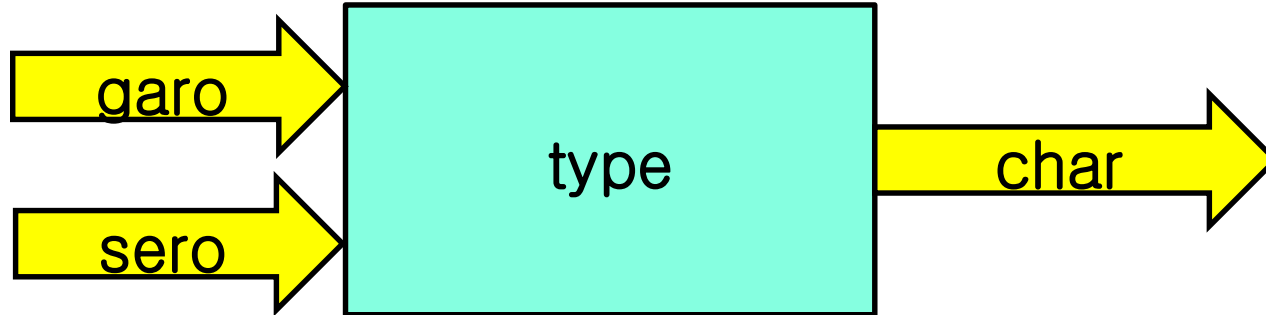
```
System.out.printf("사각형의 세로 길이 = %,d Cm\n", sero);
```

```
System.out.printf("사각형의 면적 = %,d \nu33A0\n", area);
```

```
}
```

# 사각형 Type[심화]

- 사각형의 Type을 알려주는 Type() 메소드



```
private static char type(int garo, int sero) {
 char result = '직';

 if (garo == sero)
 result = '정';

 return result;
}
```

# 사각형 Type[심화]

```
public static void main(String[] args) throws IOException {
 Scanner keyboard = new Scanner(System.in);
 int garo, sero;
 long area;
 char type;

 while(true) {
 System.out.print(" 사각형의 가로와 세로 길이 입력 : ");
 garo = keyboard.nextInt();
 sero = keyboard.nextInt();
 if (garo > 0 && sero > 0)
 break;
 else {
 System.out.println("Error - 1 이상의 숫자 이어야 함");
 System.in.read();
 }
 }
}
```

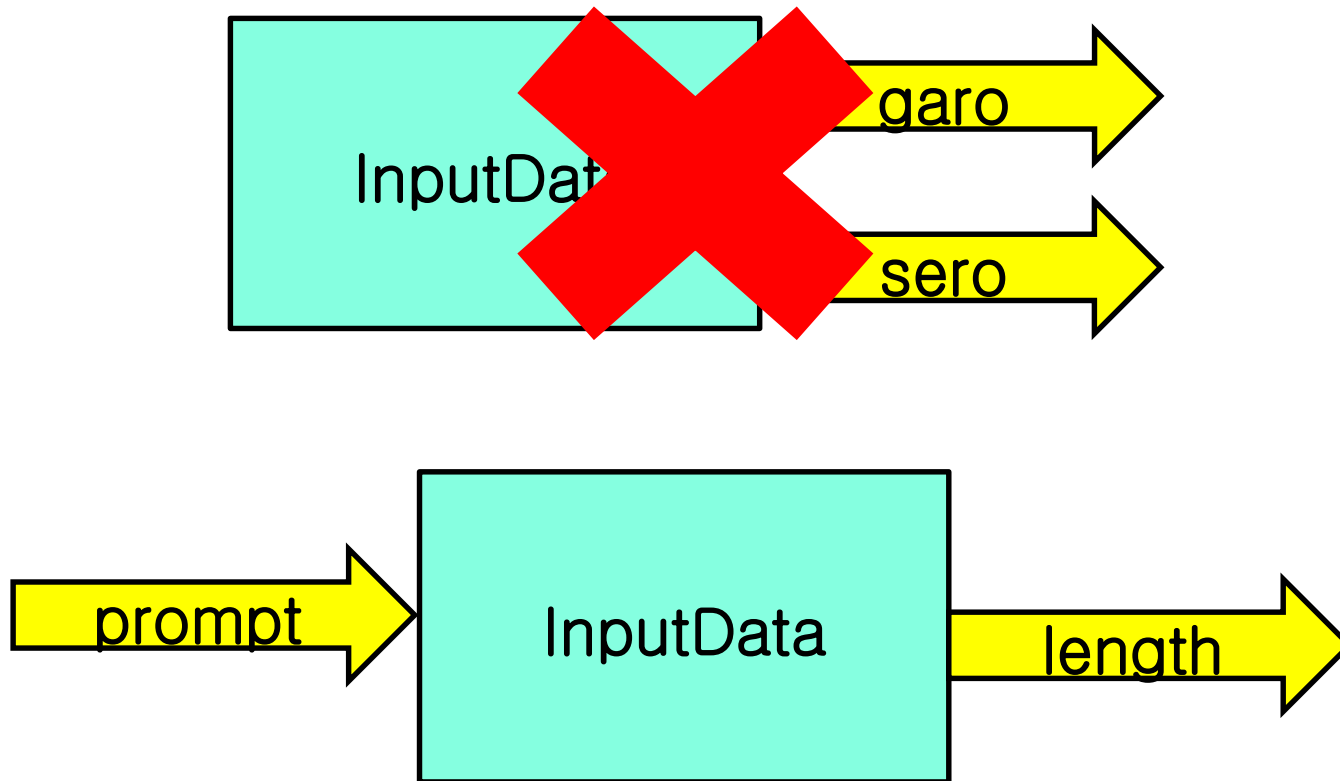
# 사각형 Type[심화]

```
type = type(garo, sero);
area = area(garo, sero);
```

```
System.out.printf("%c사각형의 가로 길이 = %,d Cm\n", type, garo);
System.out.printf("%c사각형의 세로 길이 = %,d Cm\n", type, sero);
System.out.printf("%c사각형의 면적 = %,d Wu33A0\n", type, area);
}
```

# 사각형[심화]

- 가로 길이와 세로 길이를 입력 받는 inputData() 메소드 정의



# 사각형 [심화]

```
private static int inputData(String prompt) throws IOException {
 Scanner keyboard = new Scanner(System.in);
 int length;

 while (true) {
 System.out.print(prompt);
 length = keyboard.nextInt();
 if (length > 0)
 break;
 else {
 System.out.println("Error - 1 이상의 숫자 이어야 함");
 System.in.read();
 }
 }

 return length;
}
```

# 사각형 [심화]

```
private static void output(char type, int garo, int sero, long area) {
 System.out.printf("%c사각형의 가로 길이 = %,d Cm\n", type, garo);
 System.out.printf("%c사각형의 세로 길이 = %,d Cm\n", type, sero);
 System.out.printf("%c사각형의 면적 = %,d \n", type, area);
}
```



# 사각형 [심화]

```
public static void main(String[] args) throws IOException {
 int garo, sero;
 long area;
 char type;

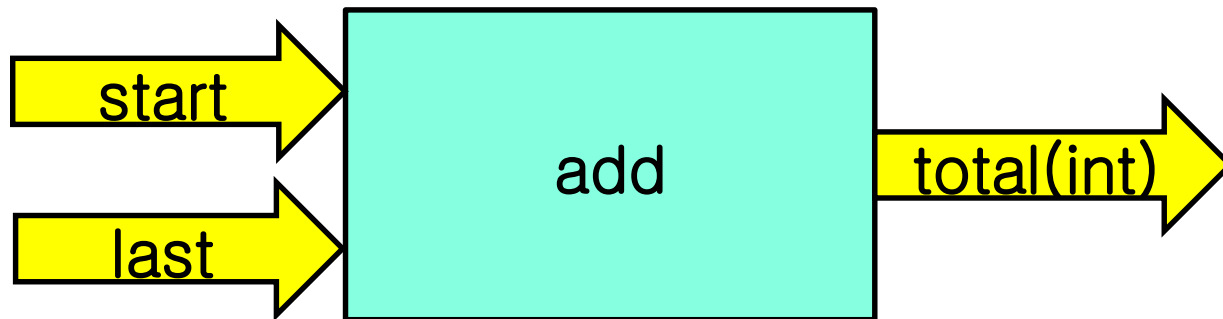
 garo = inputData(" 사각형의 가로 길이 입력 : ");
 sero = inputData(" 사각형의 세로 길이 입력 : ");

 type = type(garo, sero);
 area = area(garo, sero);

 output(type, garo, sero, area);
}
```

# 1에서 100까지의 정수의 합

- 양의 정수 X에서 Y까지의 합을 구하는 `add(X, Y)` 메소드를 정의하고 이 메소드를 이용하여 1부터 100까지의 합을 구하는 Program을 작성하여라.
- 단,  $x < y$



# 1에서 100까지의 정수의 합

형식 파라미터

```
static int add(int start, int last) {
 int total = 0;
 for (int i = start; i <= last; i++)
 total += i;
 return total;
}
```

메소드가 호출될 때 인수 1이 파라미터 start, 100이 파라미터 last로 복사

실 인수

```
public static void main(String[] args) {
 System.out.printf("1부터 100까지의 합 : %,d", add(1, 100));
}
```

call-by-value

# 1에서 100까지의 정수의 합

```
static int add(int start, int last) {
 int total = 0;
 for (int i = start; i <= last; i++)
 total += i;
 return total;
}
```

- ✓ 파라미터 start는 1, last는 100으로 초기화된 지역변수와 같음
- ✓ start, last는 add( ) 메소드 전체에 걸쳐 유효함

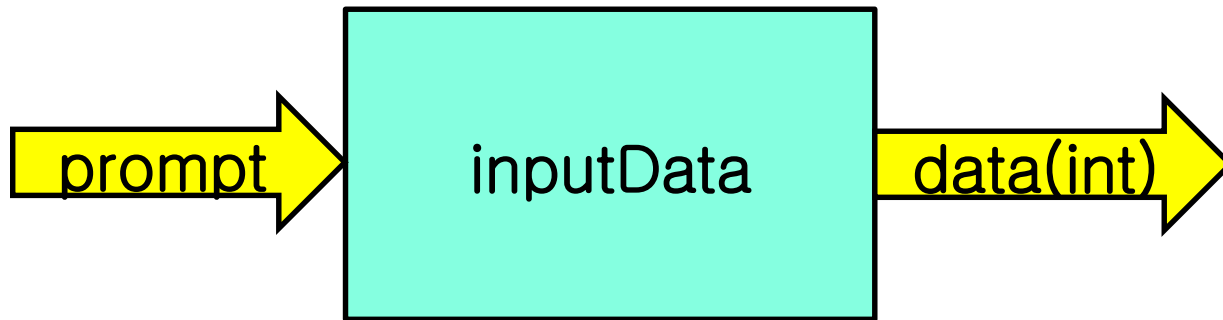
return total 문장 실행에 의해 제어가 main으로 되돌아 가고, total 값이 반환됨

```
public static void main(String[] args) {
 System.out.printf("1부터 100까지의 합 : %,d", add(1,100));
}
```

call-by-value

# 1에서 Y까지의 정수의 합

- Y 값을 keyboard로부터 받는 메소드를 생각해보자



# 1에서 Y까지의 정수의 합

```
static int inputData() throws IOException {
 Scanner keyboard = new Scanner(System.in);
 int data;

 do {
 System.out.print("어디까지 더할까요 ?");
 data = keyboard.nextInt();
 if (data > 1)
 break;
 else {
 System.err.print(" ERROR !!");
 System.in.read();
 }
 } while (true);

 return data;
}
```

# 1에서 Y까지의 정수의 합

```
public static void main(String[] args) throws IOException {
 int last = inputData();
 System.out.printf("1부터 %d까지의 합 : %,d", last, add(1, last));
}
```

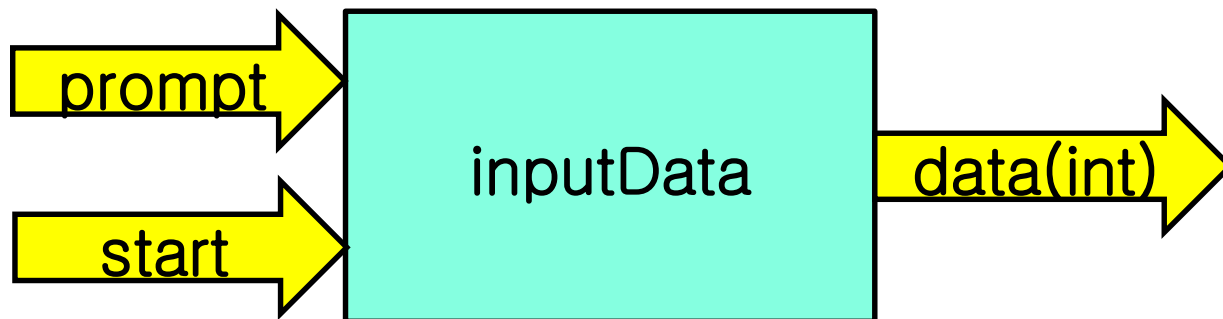
call-by-name

call-by-value

■ add() 메소드는 재활용

# X에서 Y까지의 정수의 합

- 앞에서 만든 Y값을 keyboard로부터 받아들이는 inputData() 메소드를 수정해보자





# X에서 Y까지의 정수의 합

```
static int inputData(String prompt, int start) throws IOException {
 Scanner keyboard = new Scanner(System.in);
 int data;

 do {
 System.out.print(prompt);
 data = keyboard.nextInt();
 if (data > 0 && data > start)
 break;
 else {
 System.err.print(" ERROR !!");
 System.in.read();
 }
 } while (true);

 return data;
}
```

# X에서 Y까지의 정수의 합

```
public static void main(String[] args) throws IOException {
 int start = inputData(" 어디서부터 더할까요 ? ", 0);
 int end = inputData(" 어디까지 더할까요 ? ", start);
 int sum = add(start, end);

 System.out.printf("Wn %d + %d + + %d = %,dWn",
 start, start + 1, end, sum);
}
```

# X에서 Y까지의 정수의 합

- 여러 개의 File로 나누어 볼까요 ?
  - main() 메소드와 다른 메소드를 분리하자
  - 협업이 가능함
  - 팀별로 일을 할 수 있는 기능

# X에서 Y까지의 정수의 합

## ■ Adder 클래스를 만들어보자

```
public class Adder {
 public int add(int start, int last) {
 int total = 0;
 for (int i = start; i <= last; i++)
 total += i;
 return total;
 }
}
```

# X에서 Y까지의 정수의 합

## ■ Adder 클래스를 만들어보자

```
public int inputData(String prompt, int start) throws IOException {
 Scanner keyboard = new Scanner(System.in);
 int data;

 do {
 System.out.print(prompt);
 data = keyboard.nextInt();
 if (data > 0 && data > start)
 break;
 else {
 System.err.print(" ERROR !!");
 System.in.read();
 }
 } while (true);
 return data;
}
```

# X에서 Y까지의 정수의 합

## ■ Main 클래스

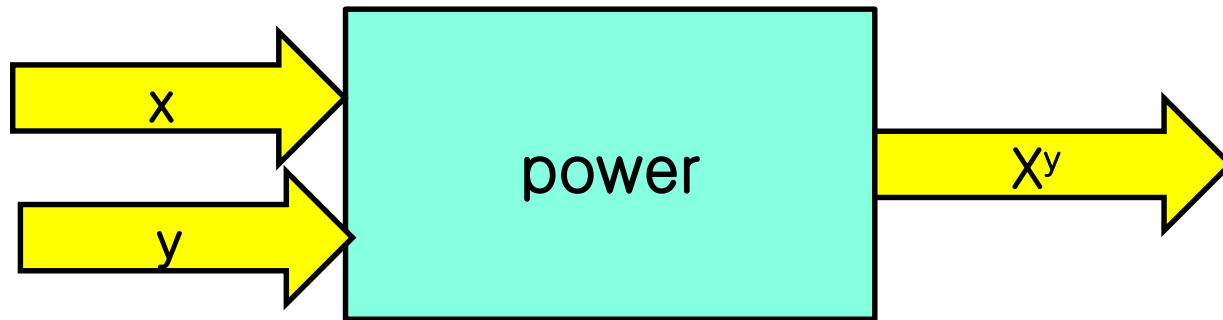
```
public class Main {
 public static void main(String[] args) throws IOException {
 Adder adder = new Adder();

 int start = adder.inputData(" 어디서부터 더할까요 ? ", 0);
 int end = adder.inputData(" 어디까지 더할까요 ? ", start);
 int sum = adder.add(start, end);

 System.out.printf(" %d + %d + + %d = %,d",
 start, start + 1, end, sum);
 }
}
```

# $X^y$ 을 구해보자

- X의 0제곱부터 10 제곱까지를 화면에 출력하는 프로그램을  $X^y$ 을 구하는 `power(x, y)` 메소드를 이용하여 프로그램 하여라.



```
private static long power(int n, int m) {
 long result = 1L;

 for (int i = 1; i <= m; i++)
 result *= n;
 return result;
}
```

# $X^y$ 을 구해보자

```
public static void main(String[] args) {
 Scanner keyboard = new Scanner(System.in);
 int number;

 System.out.print("정수 입력 : ");
 number = keyboard.nextInt();

 for (int i = 0; i <= 10; i++)
 System.out.printf("%d의 %2d승 = %,12d\n",
 number, i, power(number, i));
}
```

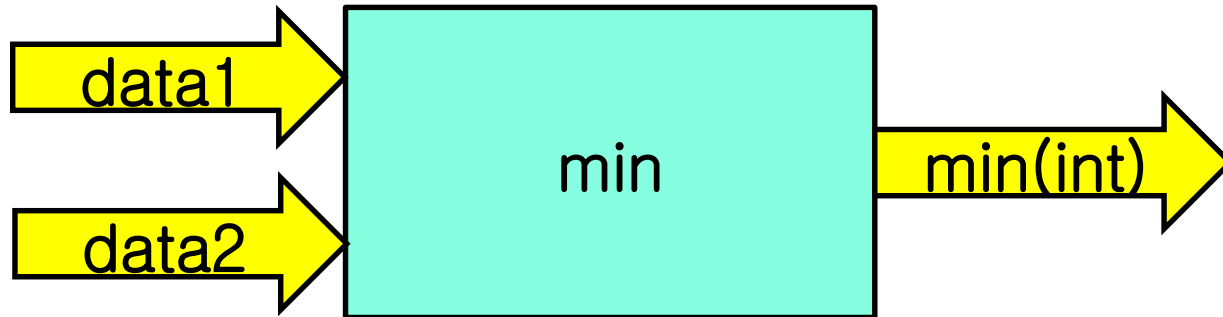


Futuristic Innovator

京福大學校  
KYUNGBOK UNIVERSITY



## 2 정수 중 작은 정수 찾기



```
private static int min(int firstData, int secondData) {
 int minNumber;
 if (firstData < secondData)
 minNumber = firstData;
 else
 minNumber = secondData;
 return minNumber;
}
```

```
private static int min(int firstData, int secondData) {
 return (firstData > secondData) ? firstData : secondData;
}
```

## 2 정수 중 작은 정수 찾기

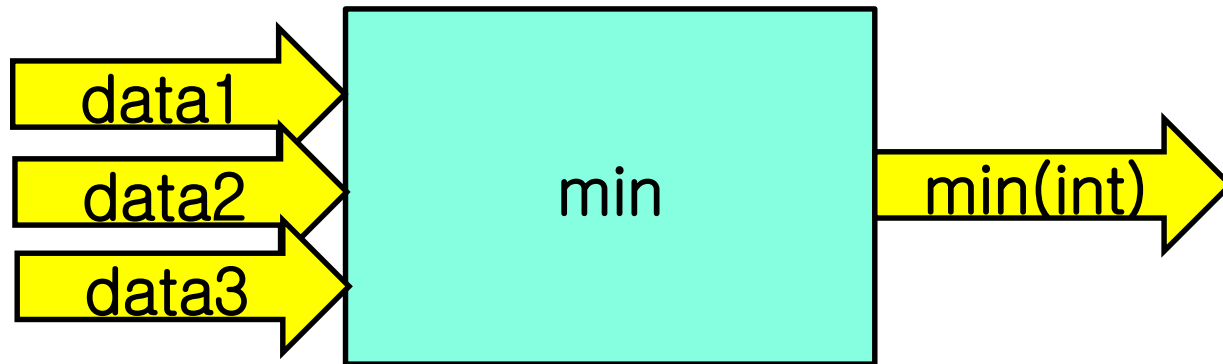
```
public static void main(String[] args) {
 Scanner keyboard = new Scanner(System.in);
 int data1, data2;
 int min;

 System.out.print(" 정수 2개 입력 : ");
 data1 = keyboard.nextInt();
 data2 = keyboard.nextInt();

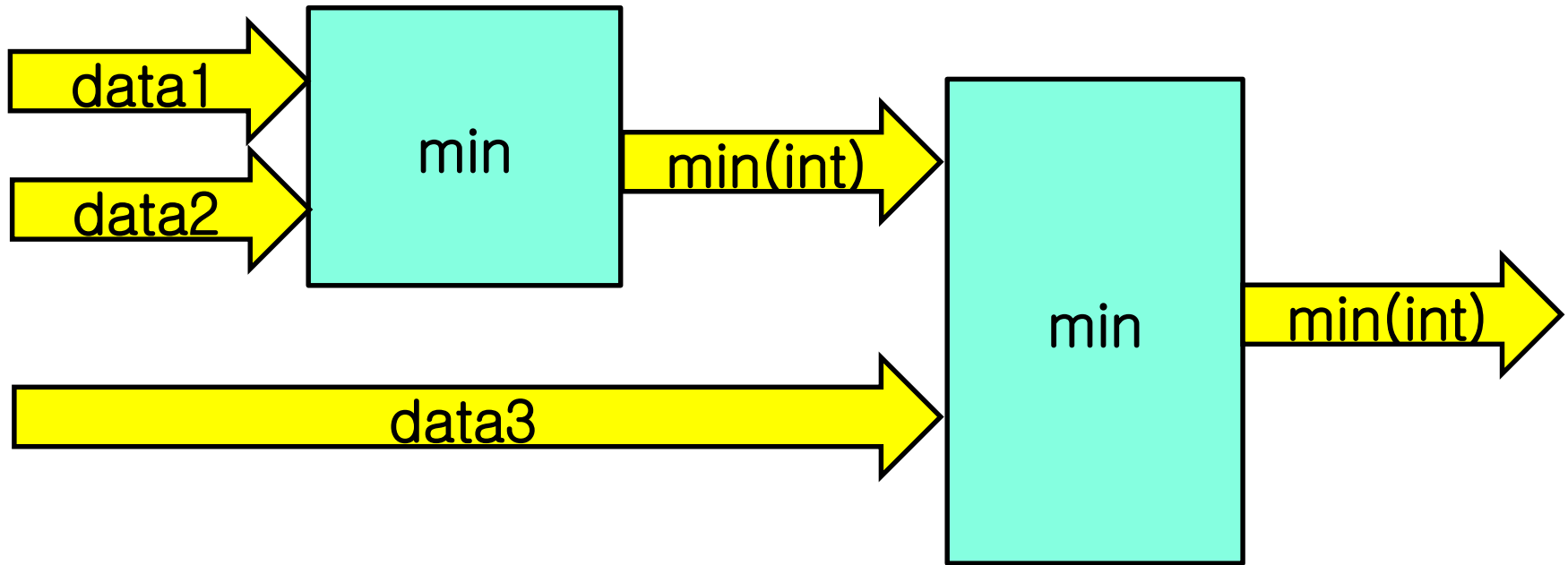
 min = min(data1, data2);
 System.out.printf(" 두 정수 %,d와 %,d중 작은 수는 %,d 이다.\n",
 data1, data2, min);
}
```

# 3 정수 중 작은 정수 찾기[심화]

- 3개의 정수를 전달받아, 최소값을 구하는 min4() 메소드를 만들어보자
- 단, 2개의 정수 중 최소값은 삼항 연산자를 사용하여 min() 메소드를 만들고, 3개 중 최소값은 2개 최소값 메소드 min()를 사용하여 만들어보자



# 3 정수 중 작은 정수 찾기[심화]



```
int min(int data1, int data2) {
 return (data1 > data2 ? data1 : data2);
}
```

```
int min3(int data1, int data2, int data3) {
 return min(min(data1, data2), data3);
}
```

# 3 정수 중 작은 정수 찾기[심화]

```
public static void main(String[] args) {
 Scanner keyboard = new Scanner(System.in);
 int data1, data2, data3;
 int min;

 System.out.print(" 정수 3개 입력 : ");
 data1 = keyboard.nextInt();
 data2 = keyboard.nextInt();
 data3 = keyboard.nextInt();

 min = min4(data1, data2, data3);
 System.out.printf(" 세 정수 %,d, %,d와
 %,d중 가장 작은 수는 %,d 이다.\n",
 data1, data2, data3, min);
}
```

# 정수 입력

- 정수를 입력받을 수 있는 `checkNumber(String value)`를 정의하여라.

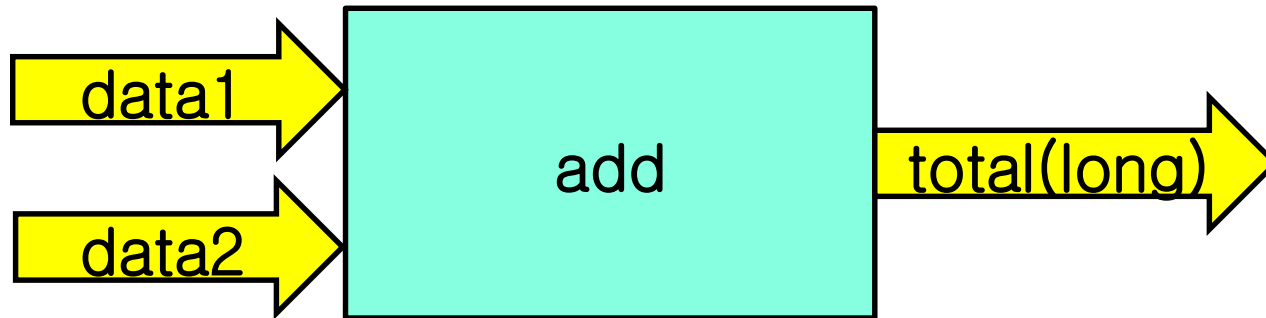
```
public static void main(String[] args) throws IOException {
 Scanner keyboard = new Scanner(System.in);
 String temp;
 do {
 System.out.print("숫자를 입력 : ");
 temp = keyboard.nextLine();
 if (checkNumber(temp)) {
 break;
 } else {
 System.err.println("입력 오류");
 System.in.read();
 }
 } while (true);
 System.out.printf("입력 받은 수 : %,d\n", Integer.parseInt(temp));
}
```

# 정수 입력

```
private static boolean checkNumber(String temp) {
 char num = temp.charAt(0);
 if (num == '+' || num == '-' || (num >= '0' && num <= '9')) {
 for (int i = 1; i < temp.length(); i++) {
 num = temp.charAt(i);
 if (!(num >= '0' && num <= '9')) {
 return false;
 }
 }
 } else {
 return false;
 }
 return true;
}
```

# 2정수 더하기

- 두 정수를 더하는 add() Method를 정의하여라



```
long add (long data1, long data2) {
 long int result;

 result = first + second;
 return (result);
}
```



# Prime Number

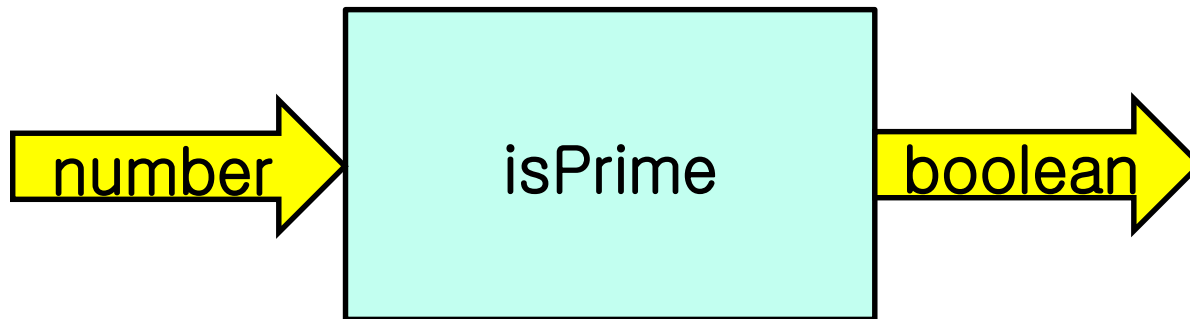
- 소수(Prime Number)인가를 판별하는 boolean isPrime(int number) 메소드를 정의하고, 이 메소드를 이용하여 Keyboard로부터 입력 받은 정수가 소수인가를 판별하는 프로그램을 작성하여라
- 소수(prime number)
  - 자신보다 작은 두 개의 자연수를 곱하여 만들 수 없는 1보다 큰 자연수이다
  - 예) 5는  $1 \times 5$  또는  $5 \times 1$ 로 수를 곱한 결과를 적는 유일한 방법이 그 수 자신을 포함하기 때문에 5는 소수이다

# Prime Number

## ■ 문제 분석

- 매개변수로서 정수 number를 갖고, number가 소수이면 true를, 그렇지 않으면 false를 반환하는 메소드를 작성할 것

```
boolean isPrime(int number)
```



# Prime Number

```
static boolean isPrime(int number) {
 if (number <= 1) // 0, 1, 모든 음수는 소수가 아님
 return false;
 else if (number == 2) // 2는 소수
 return true;
 else if (number % 2 == 0) // number가 짝수이면 소수 아님
 return false;
 for (int index = 3; index < number; index += 2)
 if (number % index == 0)
 return false;

 return true;
}
```

# Prime Number

```
public static void main(String[] args) {
 Scanner keyboard = new Scanner(System.in);
 int number;
 String result;

 System.out.print("정수를 입력하세요 ");
 number = keyboard.nextInt();

 if (isPrime(number))
 result = "는 소수 입니다.";
 else
 result = "는 소수가 아닙니다.";

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

# Prime Number

## ■ 출력

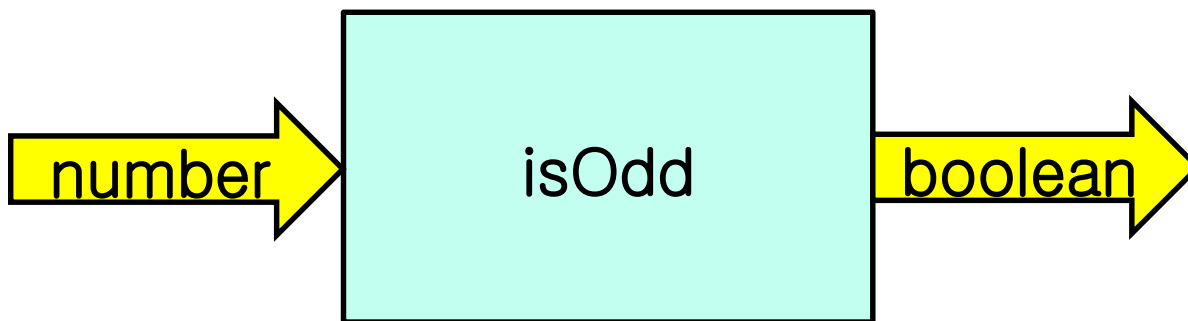
정수를 입력하세요: 6317  
6317는 소수입니다.

정수를 입력하세요: 7163  
7163는 소수가 아닙니다.

# 홀수 판별하기

- 홀수(Odd Number)인가를 판별하는 isOdd(int number) 메소드를 정의하고, 이 메소드를 이용하여 키보드로부터 입력 받은 정수가 홀수인가를 판별하는 프로그램을 작성하여라
- 제어문 수업자료 참조

boolean isOdd(int number)



# 홀수 판별하기

```
static boolean isOdd(int number) {
 if ((number % 2) == 0)
 return false;
 else
 return true;
}
```

# 홀수 판별하기

```
public static void main(String[] args) {
 Scanner keyboard = new Scanner(System.in);
 int num;
 String result = "";
 System.out.print(" 정수 입력 : ");
 num = keyboard.nextInt();

 if (isOdd(num))
 result = "홀수";
 else
 result = "짝수";

 System.out.println(num + "은 " + result + " 입니다.");
}
```



Futuristic Innovator

京福大學校  
KYUNGBOK UNIVERSITY



# 홀수 판별하기 [심화]

```
public static void main(String[] args) {
 Scanner keyboard = new Scanner(System.in);
 int data;

 System.out.print(" 정수 입력 : ");
 data = keyboard.nextInt();

 System.out.printf(" 입력한 %,d는 %c수 이다.\n", data, even(data));
}

static char even(int number) {
 char result;
 if (number % 2 == 0)
 result = '짝';
 else
 result = '홀';
 return(result);
}
```

# Reverse Number

- 숫자를 뒤집으려면 숫자를 10으로 나눈 나머지를 계속 더해주면 됨

Reverse Given Number

1234

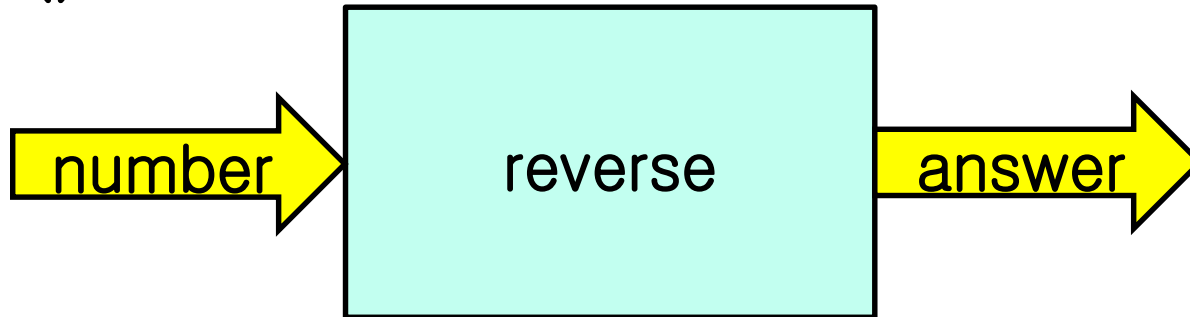


4321



# Reverse Number

## ■ reverse() 메소드



```
public static int reverse(int number) {
 int answer = 0;
 while(number > 0) {
 answer = answer * 10 + number % 10;
 number /= 10;
 }
 return answer;
}
```

# Reverse Number

```
public static void main(String[] args) {
 Scanner keyboard = new Scanner(System.in);
 int number;
 int reverse;

 System.out.print("숫자 입력 : ");
 number = keyboard.nextInt();

 reverse = reverse(number);

 System.out.printf("입력 수 : %,d\n", number);
 System.out.printf("역순 수 : %,d\n", reverse);
}
```

# 진법(2 ~ 16진법) 변환

- 자연수를 입력 받아 원하는 진법(2~16진법)의 수로 변경하는 프로그램을 convert() 메소드를 정의하여 작성하여라.

$$\begin{array}{r} 8 \overline{) 491} \\ 8 \overline{) 61} \dots 3 \\ \quad 7 \dots 5 \end{array}$$

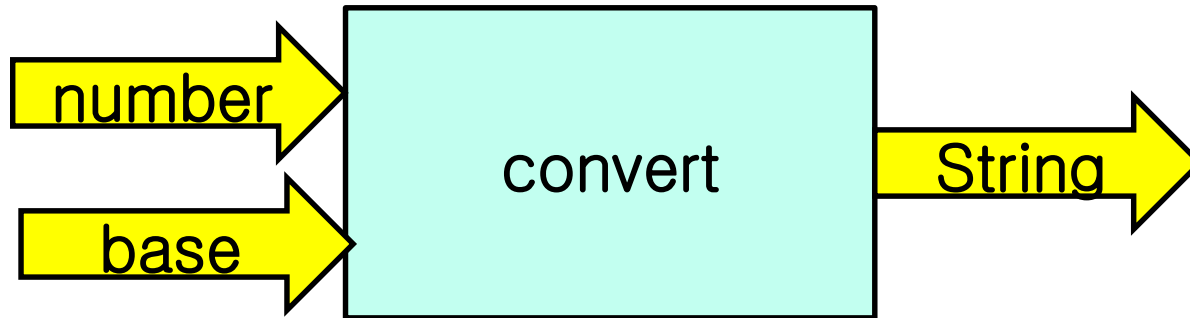
$$491_{(10)} = 753_{(8)}$$

$$\begin{array}{r} 16 \overline{) 935} \\ 16 \overline{) 58} \dots 7 \\ \quad 3 \dots A \end{array}$$

$$935_{(10)} = 3A7_{(16)}$$

# 진법(2 ~ 16진법) 변환

- 자연수를 입력 받아 원하는 진법(2~16진법)으로 의 수로 변경하는 convert() 메소드 정의



# 진법(2 ~ 16진법) 변환

```
private static String convert(int decimal, int base) {
 char[] digit = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
 'A', 'B', 'C', 'D', 'E'};

 String result = "";

 while (decimal != 0) {
 int index = decimal % base;
 result = digit[index] + result;
 decimal /= base;
 }
 return result;
}
```

# 진법(2 ~ 16진법) 변환

```
public static void main(String[] args) throws IOException {
 Scanner keyboard = new Scanner(System.in);
 int number, base;
 while (true) {
 System.out.print(" 자연수 입력 : ");
 number = keyboard.nextInt();
 System.out.print(" 원하는 진법 : ");
 base = keyboard.nextInt();
 if (number > 0 && (base >= 2 && base <= 16))
 break;
 else {
 System.err.println("ERROR : 다시 입력해 주세요");
 System.in.read();
 }
 }
 System.out.printf(" %d의 %d 진법의 값은 %s 입니다\n", number, base,
 convert(number, base));
}
```

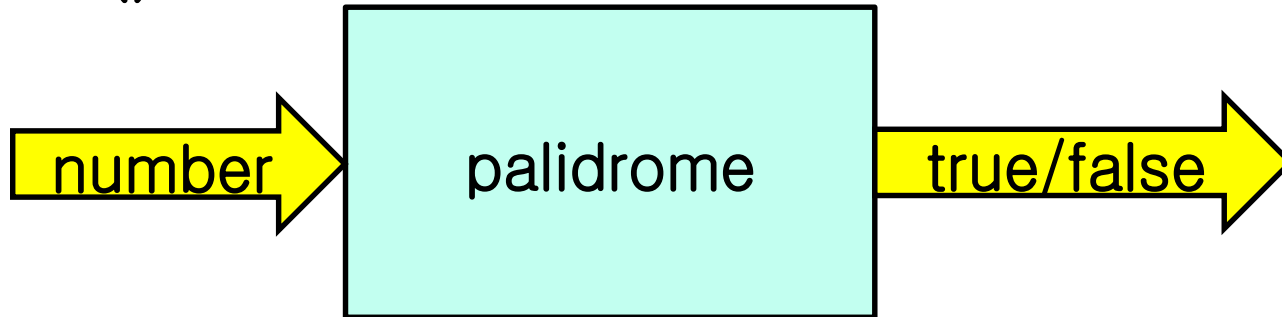


# Palindrome Number

- 회문 수(Palindromic number)
  - 숫자를 역순으로 쓴 수와 같은 수를 말함
  - 즉, 121, 151, 545, 34543 또는 526625와 같은 수

# Palindrome Number

## ■ palidrome() 메소드



```
private static boolean palidrome(int num) {
 int remainder, sum = 0;

 int temp = num;
 while (num > 0) {
 remainder = num % 10;
 sum = (sum * 10) + remainder;
 num /= 10;
 }
 return temp == sum;
}
```

# Palindrome Number

```
public static void main(String[] args) {
 Scanner keyboard = new Scanner(System.in);
 int num;
 String result;

 System.out.print("정수 입력 : ");
 num = keyboard.nextInt();

 if (palindrome(num))
 result = " is palindrome number.";
 else
 result = " is not palindrome number.";

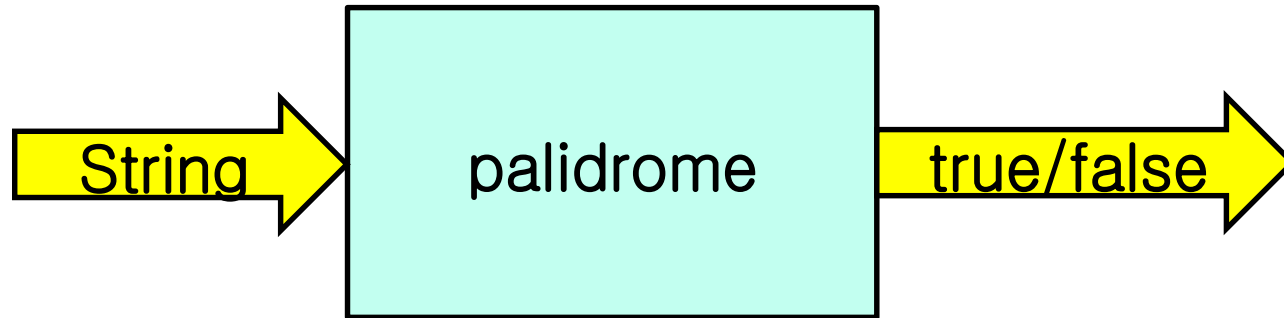
 System.out.println(num + result);
}
```

# Palindrome[심화]

## ■ Palindrome

- 앞 뒤 방향으로 볼 때 같은 순서의 문자로 구성된 문자열을 말함
- 예) 'abba', 'mom', 'dad", 'kayak', 'reviver', 'madam'은 모두 회문
- 예) 우영우, 기러기, 역삼역
- 다 좋은 것은 좋다
- 소주 만 병만 주소
- 여보 안경 안 보여
- 다 큰 도라지라도 크다
- 아들딸이 다 컸다 이 딸들아
- 대한전선 사장이 장사 선전한대
- 다 간다 이 일요일 일요일이 다 간다

# Palindrome[심화]



```
private static boolean palindrome(String word) {
 String reverse = "";
 for (int i = (word.length() - 1); i >= 0; --i) {
 reverse += word.charAt(i);
 }

 return word.equalsIgnoreCase(reverse);
}
```

# Palindrome[심화]

```
public static void main(String[] args) {
 Scanner keyboard = new Scanner(System.in);
 String word;
 String result;

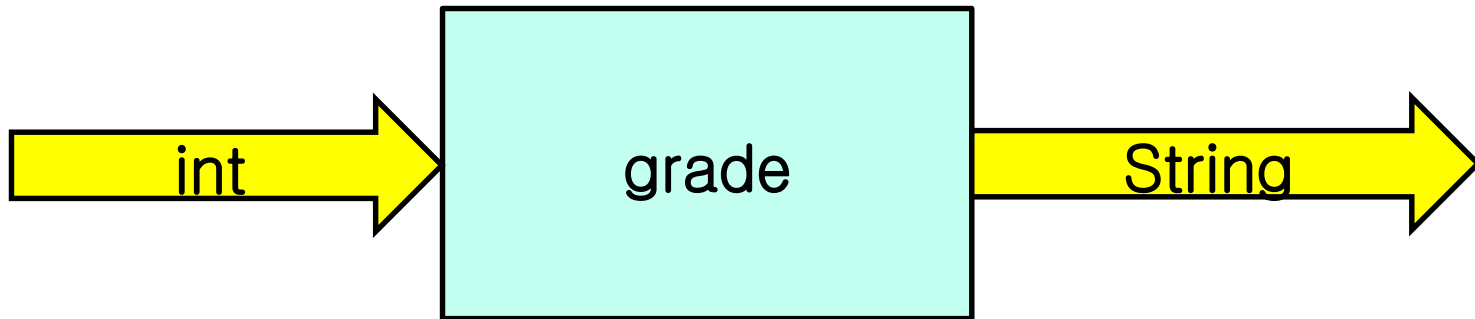
 System.out.print("단어 입력 : ");
 word = keyboard.next();

 if (palindrome(word)) {
 result = " is a Palindrome String.";
 } else {
 result = " is not a Palindrome String.";
 }

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

# 학점 구하기

- 점수를 입력 받아 학점(A+, A0, ..., F)을 구하는 메소드  
String grade(int score)를 정의하여 작성하여라.



# 학점 구하기

```
private static String grade(int score) {
 String result = "";

 if (score >= 90)
 result = "A";
 else if (score >= 80)
 result = "B";
 else if (score >= 70)
 result = "C";
 else if (score >= 60)
 result = "D";
 else
 result = "F";
}
```



Futuristic Innovator

京福大學校  
KYUNGBOK UNIVERSITY



# 학점 구하기

```
if (result != "F") {
 if (score % 10 - 5 >= 0)
 result += "+";
 else
 result += "0";
}
return result;
}
```

# 학점 구하기

```
public static void main(String[] args) {
 Scanner keyboard = new Scanner(System.in);
 int score;
 String result;

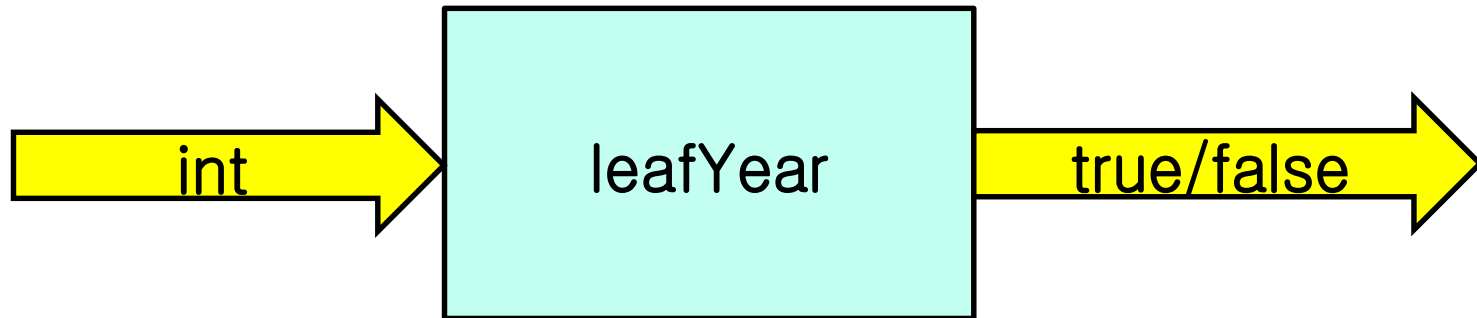
 System.out.printf("\n 점수 입력 : ");
 score = keyboard.nextInt();

 result = grade(score);

 System.out.println(score + "는 학점 " + result + "입니다");
}
```

# 윤년

- 윤년을 판단하는 메소드 leafYear()를 정의하여라



# Anagram

- Anagram의 사전적 의미는 글자를 재배열하여 형성된 단어 또는 구문을 말함
- 두 개의 문자열은 문자열의 문자를 재정렬하거나 섞어 의미 있는 단어를 만드는 경우 Anagram이라고 함
- 즉, 두 문자열이 동일한 문자를 포함하지만 순서가 다른 경우 Anagram이라고 말할 수 있음
  - 예)
    - Race -> Care
    - LISTEN - > SILENT
    - HEART - > EARTH
    - LIVES - > ELVIS
    - KEEP -> PEEK
    - TABLE - >BLEAT

# 이상한 문자열

- 문자열 s는 한 개 이상의 단어로 구성되어 있다. 각 단어는 하나 이상의 공백 문자로 구분되어 있다. 각 단어의 짝수 번째 알파벳은 대문자로, 홀수 번째 알파벳은 소문자로 바꾼 문자열을 반환하는 메소드를 만들어보자
- 제한사항
  - 문자열 전체의 짝/홀수 인덱스가 아니라, 단어(공백을 기준)별로 짝/홀수 인덱스를 판단해야 함
  - 첫 번째 글자는 0번째 인덱스로 보아 짝수 번째 알파벳으로 처리해야 함
  - "try hello world" -> "TrY HeLlO WoRlD"

# 수열의 합 구하기

■  $1 + 1/2 + 1/3 + 1/4 + 1/5 + \dots + 1/n$  를 출력한다

```
import java.util.Scanner;
public class InfiniteSeries {
 public static void main(String[] args) {
 final double MAX = 1E8; // n을 얼마까지나 증가시킬 것인가?
 double sum;

 for (double n = 1; n <= MAX; n *= 10.0) {
 // 1 + 1/2 + 1/3 + 1/4 + 1/5 + ... + 1/n 를 출력한다.
 sum = 0.0;
 for (double i = 1; i <= n; i++) {
 sum += 1.0/i;
 }
 System.out.println("n = " + n + ": " + "sum = " + sum);
 }
 }
}
```

역수의 합을 구하는 부분

# 수열의 합 구하기

```
public class InfiniteSeries {
 public static void main(String[] args) {
 final double MAX = 1E8; // n을 얼마까지나 증가시킬 것인가?
 double sum;
 for (double n = 1; n <= MAX; n *= 10.0) {
 // 1 + 1/2 + 1/3 + 1/4 + 1/5 + ... + 1/n 를 출력한다.
 sum = addInverses(n); // n까지 역수들의 합을 구함
 System.out.println("n = " + n + ": " + "sum = " + sum);
 }
 }
 // to까지의 역수들의 합을 구하는 메소드 (1/1 + 1/2 + 1/3 + ... + 1/to)
 static double addInverses(double to) {
 double sum = 0.0;
 for (double i = 1; i <= to; i++)
 sum += 1.0/i;
 return sum;
 }
}
```

# Method Overloading

- Method를 구분하는 Signature
  - 메소드의 이름
  - 메소드의 전달 인자(매개 변수) Data Type
  - 메소드의 전달 인자(매개 변수) 개수
- 메소드를 구분하는 Signature중에서 메소드의 이름이 같아야 메소드의 오버로딩(Overloading)이므로 다음 두 가지 조건 중 하나를 만족해야 함
  - 메소드의 전달 인자 Data Type이 달라야 함
  - 메소드의 전달 인자 개수가 달라야 함



# Method Overloading

- 한 클래스 내에서 같은 이름의 메소드 사용
  - 같은 이름의 메소드는 매개변수의 타입, 개수가 달라야 함
  - 매개변수의 개수나 타입에 맞는 메소드 수행

| 구분       | Overloading | Overriding |
|----------|-------------|------------|
| 메소드 이름   | 동일          | 동일         |
| 매개변수, 타입 | 다름          | 동일         |
| 반환 타입    | 상관없음        | 동일         |

# Method Overloading 예제 1

## ■ 절대값 abs() 메소드 정의

```
private static int abs(int num){ //int형 데이터에 절대값 메소드
 if (num < 0)
 num = -num;
 return num;
}
```

```
private static long abs(long num){ //long형 데이터에 절대값 메소드
 if (num < 0)
 num = -num;
 return num;
}
```

```
private static double abs(double num){ //double 데이터에 절대값
 if (num < 0)
 num = -num;
 return num;
}
```

# Method Overloading 예제 1

```
public static void main(String[] args) {
 int var01 = -10, var02;
 var02 = abs(var01);
 System.out.println(var01 + "의 절대값은-> " + var02);

 long var03 = -20L, var04;
 var04 = abs(var03);
 System.out.println(var03 + "의 절대값은-> " + var04);

 double var05 = -3.4, var06;
 var06 = abs(var05);
 System.out.println(var05 + "의 절대값은-> " + var06);
}
```

# Method Overloading 예제2

- 매개변수의 개수가 다른 myPrint() 메소드를 Overloading으로 정의

```
private static void myPrint(int a, int b, int c) {
 System.out.println(a + "Wt" + b + "Wt" + c);
}
```

```
private static void myPrint(int a, int b) {
 System.out.println(a + "Wt" + b);
}
```

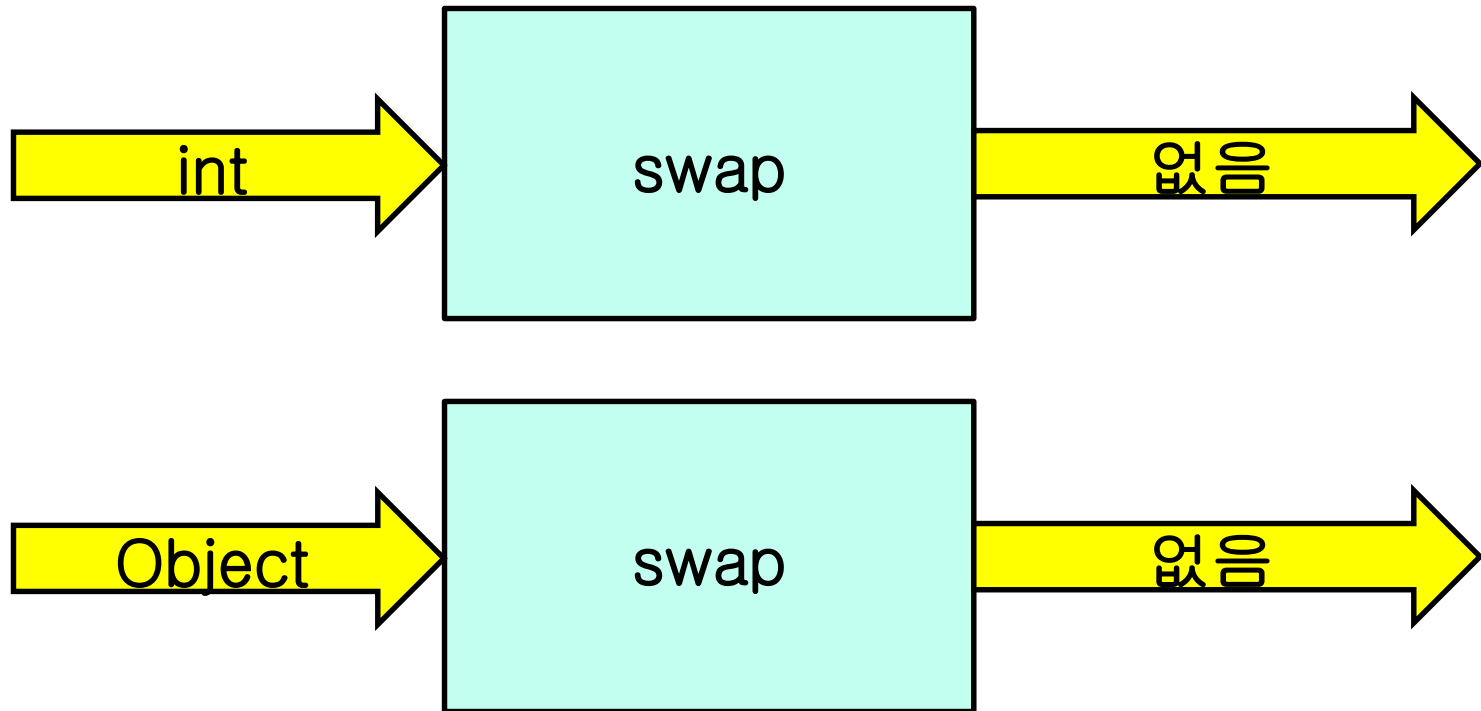
```
private static void myPrint(int a) {
 System.out.println(a);
}
```

# Method Overloading 예제2

```
public static void main(String[] args) {
 myPrint(10, 20, 30); //정수형 데이터 3개를 실매개변수로 지정
 myPrint(40, 50);
 myPrint(60);
}
```

# Method Overloading 예제3

- 두 정수를 서로 교환하는 swap() 메소드를 call by value와 call by reference 방식으로 만들어보자.



# Swapping(I)

```
public class Main {
 static int a = 10;
 static int b = 20;

 public static void main(String[] args) {
 System.out.println(">> Before Swapping <<");
 System.out.println("a = " + a + ", b = " + b);
 swap(a, b);
 System.out.println(">> After Swapping <<");
 System.out.println("a = " + a + ", b = " + b);
 }

 private static void swap(int a, int b) {
 int temp = a;
 a = b;
 b = temp;
 }
}
```

# Swapping(II)

```
public class Main {
 int a = 10;
 int b = 20;

 public static void main(String[] args) {
 Main object = new Main();
 System.out.println(">> Before Swapping <<");
 System.out.println("a = " + object.a + ", b = " + object.b);

 swap(object);

 System.out.println(">> After Swapping <<");
 System.out.println("a = " + object.a + ", b = " + object.b);
 }
}
```



# Swapping(II)

```
private static void swap(Main object) {
 int temp = object.a;
 object.a = object.b;
 object.b = temp;
}
}
```

# Swapping(III)

```
public static void main(String[] args) {
 int a = 10;
 int b = 20;
 int[] array = {a, b};

 System.out.println(">> Before Swapping <<");
 System.out.println("a = " + array[0] + ", b = " + array[1]);
 swap(array);
 System.out.println(">> After Swapping <<");
 System.out.println("a = " + array[0] + ", b = " + array[1]);
}
private static void swap(int[] array) {
 int temp = array[0];
 array[0] = array[1];
 array[1] = temp;
}
}
```

# Swapping(IV)

```
public class Main {
 public static void main(String[] args) {
 IntWrapper a = new IntWrapper(10);
 IntWrapper b = new IntWrapper(20);

 System.out.println(">> Before Swapping <<");
 System.out.println("a = " + a.value + ", b = " + b.value);

 swap(a, b);

 System.out.println(">> After Swapping <<");
 System.out.println("a = " + a.value + ", b = " + b.value);
 }
}
```

# Swapping(IV)

```
public static void swap(IntWrapper a, IntWrapper b) {
 IntWrapper c = new IntWrapper();
 c.value = a.value;
 a.value = b.value;
 b.value = c.value;
}
```

```
public class IntWrapper {
 public int value;
 public IntWrapper(){
 this.value = 0;
 }
 public IntWrapper(int a){
 this.value = a;
 }
}
```

# Swapping(V)

```
public class Main {
 static int a, b;

 public static void main(String[] args) {
 a = Integer.parseInt(args[0]);
 b = Integer.parseInt(args[1]);

 System.out.println(">> Before Swapping <<");
 System.out.println("a = " + a + ", b = " + b);

 swap(a, b);

 System.out.println(">> After Swapping <<");
 System.out.println("a = " + a + ", b = " + b);
 }
}
```

# Swapping(V)

```
private static void swap(int a, int b) {
 int temp;
 temp = a;
 a = b;
 b = temp;
}
}
```

# Swapping(VI)

```
public static void main(String[] args) {
 int[] array = new int[2];

 array[0] = Integer.parseInt(args[0]);
 array[1] = Integer.parseInt(args[1]);

 System.out.println(">> Before Swapping <<");
 System.out.println("a = " + array[0] + ", b = " + array[1]);

 swap(array);

 System.out.println(">> After Swapping <<");
 System.out.println("a = " + array[0] + ", b = " + array[1]);
}
```

# Swapping(VI)

```
private static void swap(int[] data) {
 int temp;
 temp = data[0];
 data[0] = data[1];
 data[1] = temp;
}
```



# String[] args

- JAVA의 main() 메소드가 static인 이유
  - Stack 영역에 스레드가 있어야 'new 연산자'를 통해 객체 생성이 가능하지만 Stack 영역에 가장 먼저 적재되는 스레드는 main() 메소드
  - 그러므로 main() 메소드의 객체 생성이 불가능하기에 main() 메소드는 static으로 설정
  - main()이 첫 스레드라면 사용자가 main()에게 데이터를 넘겨주고 싶은 경우도 있을 것임
  - main()에게 데이터를 넘겨주려면 main()이 시작되기 전에 데이터를 입력 받는 스레드가 필요
  - 하지만 Stack 영역의 적재되는 첫 스레드는 main()
  - 이를 위해 JDK는 JVM이 클래스 파일을 처리하기 전 커맨드 라인을 통해 데이터를 사용자로부터 입력 받음

# String[] args

- static 메소드 사용시 주의할 점
  - 메소드에서 인스턴스 변수를 사용하지 못함
  - static 메소드에서 인식할 수 있는 변수
    - static 변수
    - 지역 변수

# String[] args 예제

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

 if (args.length == 0) {
 System.err.println("옵션을 입력하세요");
 System.exit(1);
 }

 System.out.println("모두 " + args.length +
 "개의 옵션을 입력하였습니다.");
 System.out.println(); // 줄바꿈

 for (int i = 0; i < args.length; i++)
 System.out.format("args[%d] : %s%n", i, args[i]);
}
```

# String[] args 예제

## ■ [실행]-[구성 편집]

The screenshot shows the IntelliJ IDEA interface with the 'Run' menu open. The '구성 편집(B)...' (Edit Configuration) option is highlighted. The 'Run Configuration' dialog for 'Args01' is visible, showing the 'Main' configuration with the following settings:

- Working Directory: C:\Users\bae\.jdk\openjdk-21.0.1\
- Environment: 모두 2개의 옵션을 입력하였습니다.
- Arguments: args[0] : 경복대, args[1] : 홍길동
- VM Options: 종료 코드 0(으)로 완료된 프로세스

The background code editor shows a Java file 'Args01WMain.java' with the following code:

```
args.length; i++)
t("args[%d] : %s\n", i, args[i]);
```

The bottom status bar shows the current file is '실행/디버그 구성 편집 대화상자를 엽니다' (Opening the Run/Debug Configuration dialog).

# String[] args 예제

