



# **Data Analysis**

## **(Stream and Parallel Processing 1)**

**Fall, 2020**

# Calendar

달력

양음력변환

날짜계산

전역일계산

만나이계산

오늘

<

2020.09

>

☐ 음력

☐ 손없는날

☒ 기념일

| 일   | 월             | 화            | 수           | 목                    | 금            | 토           |
|-----|---------------|--------------|-------------|----------------------|--------------|-------------|
| 30  | 31            | 1<br>소개      | 2<br>음 7.15 | 3<br>환경<br>세팅        | 4<br>지식재산... | 5           |
| 6   | 7<br>백로       | 8<br>복습<br>1 | 9           | 10<br>9.1<br>복습<br>2 | 11           | 12          |
| 13  | 14            | 15           | 16          | 17<br>음 8.1          | 18           | 19<br>청년의 날 |
| 3주차 |               |              |             |                      |              |             |
| 20  | 21<br>치매극복... | 22           | 23          | 24                   | 25           | 26          |
| 4주차 |               |              |             |                      |              |             |
| 27  | 28            | 29           | 30          | 1                    | 2            | 3           |
| 5주차 |               |              |             |                      |              |             |

# Calendar

| 달력  | 양음력변환        | 날짜계산        | 전역일계산       | 만나이계산                      |               |                      |
|---|--------------|-------------|-------------|----------------------------|---------------|----------------------|
| <div>오늘&lt;2020.10&gt;</div> <div><input type="checkbox"/> 음력<input type="checkbox"/> 손없는날<input checked="" type="checkbox"/> 기념일</div> |              |             |             |                            |               |                      |
| 일   | 월            | 화           | 수           | 목                          | 금             | 토                    |
| 27  | 28           | 29          | 30          | 1<br>음 8.15<br>추석<br>국군의 날 | 2<br>노인의 날    | 3<br>개천절             |
| 4   | 5<br>세계 한... | 6주차         |             |                            | 9<br>한글날      | 10                   |
| 11  | 12           | 13          | 14          | 15<br>체육의 날                | 16<br>부마민주... | 17<br>음 9.1<br>문화의 날 |
| 18  | 19           | 20          | 21          | 22                         | 23<br>상강      | 24<br>국제연합일          |
| 25<br>독도의날<br>중양절   | 26           | 27<br>금유의 날 | 28<br>교정의 날 | 29<br>지방자치...              | 30            | 31<br>음 9.15         |
| 9주차   |              |             |             |                            |               |                      |

# Calendar

| 달력  |    | 양음력변환 |      | 날짜계산 |    | 전역일계산 |  | 만나이계산 |  |
|---|----|-------|------|------|----|-------|--|-------|--|
| <div>오늘&lt;2020.11&gt;</div> <div><input type="checkbox"/> 음력<input type="checkbox"/> 손없는날<input checked="" type="checkbox"/> 기념일</div> |    |       |      |      |    |       |  |       |  |
| 일   | 월  | 화     | 수    | 목    | 금  | 토     |  |       |  |
| 1   | 2  | 3     | 4    | 5    | 6  | 7     |  |       |  |
|   |    | 10주차  |      |      |    | 입동    |  |       |  |
| 8   | 9  | 10    | 11   | 12   | 13 | 14    |  |       |  |
| 소방의 날   |    | 11주차  |      |      |    |       |  |       |  |
| 15  | 16 | 17    | 18   | 19   | 20 | 21    |  |       |  |
| 음 10.1  |    |       | 12주차 |      |    |       |  |       |  |
| 22  | 23 | 24    | 25   | 26   | 27 | 28    |  |       |  |
| 소설  |    |       | 13주차 |      |    |       |  |       |  |
| 29  | 30 | 1     | 2    | 3    | 4  | 5     |  |       |  |
| 음 10.15   |    |       |      |      |    |       |  |       |  |

# Calendar

달력

양음력변환

날짜계산

전역일계산

만나이계산

오늘

<

2020.12

>

☐ 음력
☐ 손없는날
☒ 기념일

| 일             | 월        | 화             | 수  | 목  | 금         | 토          |
|---------------|----------|---------------|----|----|-----------|------------|
| 29            | 30       | 1             | 2  | 3  | 4         | 5<br>무역의 날 |
| 14주차          |          |               |    |    |           |            |
| 6             | 7<br>대설  | 8             | 9  | 10 | 11        | 12         |
| 15주차          |          |               |    |    |           |            |
| 13            | 14       | 15<br>음 11.1  | 16 | 17 | 18        | 19         |
| 16주차: 기말고사 주간 |          |               |    |    |           |            |
| 20            | 21<br>동지 | 22            | 23 | 24 | 25<br>성탄절 | 26         |
| 27<br>원자력의... | 28       | 29<br>음 11.15 | 30 | 31 | 1         | 2          |

# Table of Contents

- Functional Programming and Lambda Expression
- Motivation for Parallel Stream

# Review the parallel friends of friends

```
new Thread(new Runnable() {  
    @Override  
    public void run() {  
        for (int i = ki.length * 3 / 4; i < ki.length; i++) {  
            Integer k = (Integer) ki[i];  
            HashSet<Integer> v = data.get(k);  
            for (Integer ve : v) {  
                HashSet<Integer> vv = data.get(ve);  
                if (vv != null) {  
                    for (Integer vve : vv) {  
                        HashSet<Integer> vvv = data.get(vve);  
                    }  
                }  
            }  
        }  
    }  
}).start();
```

Separate the scope of task yourself

Would like to only focus on the task

X4 or X6

# Introduction to Lambda Expression support in Java 8

- Since v8, Java supports Lambda Expression
  - 명령형 프로그래밍: 프로그래밍의 상태와 상태를 변경시키는 구문의 관점에서 연산을 설명하는 방식
    - 절차지향 프로그래밍: 수행되어야 할 연속적인 계산 과정을 포함하는 방식 (C, C++)
    - 객체지향 프로그래밍: 객체들의 집합으로 프로그램의 상호작용을 표현 (C++, Java, C#)
  - 선언형 프로그래밍: 어떤 방법으로 해야 하는지(How)를 나타내기보다 무엇(What)과 같은지를 설명하는 방식
    - 함수형 프로그래밍: 순수 함수를 조합하고 소프트웨어를 만드는 방식 (클로저, 하스켈, 리스프)

|             | 명령형 프로그래밍       | 함수형 프로그래밍                   |
|-------------|-----------------|-----------------------------|
| 프로그램이란?     | 프로그램은 명령의 수행이다  | 프로그램은 함수의 계산이다              |
| 중점적 시각      | 어떻게(how to)에 초점 | 무엇(what)에 초점                |
| 이론적 배경      | 튜링 머신           | 람다 계산식                      |
| 주요 프로그래밍 언어 | C, 자바 등 대부분의 언어 | Scheme, Haskell, ML, Erlang |

〈표 1〉 명령형 프로그래밍 대 함수형 프로그래밍

<https://velog.io/@kyusung/%ED%95%A8%EC%88%98%ED%98%95-%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D-%EC%9A%94%EC%95%BD>



# Introduction to Lambda Expression support in Java 8

- Benefits
  - Conciseness
  - Reduction in code bloat
  - Readability
  - Elimination of shadow variables
  - Encouragement of functional programming
  - Code reuse
  - Enhanced iterative syntax
  - Simplified variable scope
  - Less boilerplate code
  - JAR file size reductions
  - Parallel processing opportunities

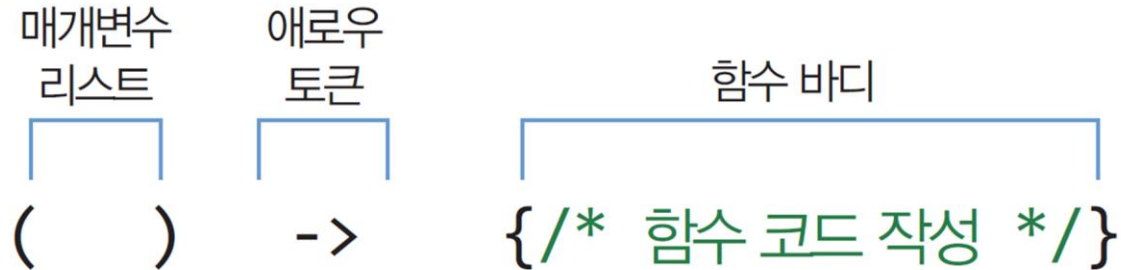
<https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Benefits-of-lambda-expressions-in-Java-makes-the-move-to-a-newer-JDK-worthwhile>

# Lambda Expression

- Lambda Expression
  - Is a way to express mathematical functions
  - E.g., define a function , which computes  $x + y$ 
    - $(x,y) \rightarrow x + y$ ;
  - E.g., use the function
    - $((x,y) \rightarrow x + y)(2,3)$
    - $= 2 + 3$
    - $= 5$

# Lambda Expression

- Syntax



[그림 1] 자바의 람다식 구조

- Parameters
  - Type of parameters can be omitted
  - () can be omitted if one parameter
- Function Body
  - {} can be omitted if only one statement

# Lambda Expression

- Example
  - Print the given age

```
(int age) -> { System.out.println("나이는 " + age); }
```

- Type of a parameter can be omitted (inferred by compiler)

```
(age) -> { System.out.println("나이는 " + age); } // int 타입 생략
```


- () and {} can be omitted

```
age -> System.out.println("나이는 " + age); // ()와 {} 생략
```

# Lambda Expression

- Example
  - Return the sum of the given x and y

```
(x, y) -> { return x + y; } // 합을 리턴하는 람다식
```

 (x, y) -> return x + y; // 오류. {} 생략 안 됨

```
(x, y) -> x + y; // return 생략 가능. x+y의 합을 리턴하는 람다식
```

# Lambda Expression

- How to write and use Lambda Expression!
- Functional Interface
  - An interface contains only one method

```
interface MyFunction { // 함수형 인터페이스  
    int calc(int x, int y); // 추상 메소드  
}
```

- Implementing the interface using Lambda Expression

```
MyFunction f = (x, y) -> { return x + y; }
```



Inferred Type

- Using the method

```
System.out.println(f.calc(1, 2));
```

# Practice #1

- 1. Implementing HelloWorld in a learnt way

```
new Thread(new MyRunnable()).start();
```

- 2. Anonymous class instance

```
new Thread(new Runnable() {  
    @Override  
    public void run() {  
        System.out.println("Hello Thread World");  
    }  
}).start();
```

# Practice #1

- 3. Using Lambda expression

```
Runnable r = () -> { System.out.println("Hello Thread World");};  
  
new Thread(r).start();
```

- 4. Using Lambda expression

```
new Thread(() -> {  
    System.out.println("Hello Thread World");  
}).start();
```



## Practice #2

- 1. Try to create MyHashSet using Lambda expression
  - Is it possible?

## Practice #3

- Create MyFunction interface
  - The interface has one 'calc' method
    - Receives two integer parameters
    - Return integer
- Create and use instance of myFunction
  - 1. add values
  - 2. subtract arg 1 by arg 2

# Lambda Expression

- Practice



## 예제 1

매개변수 x, y의 합과 차를 출력하는 2개의 람다식 만들기

이 예제에서 람다식이 한 문장이므로 중괄호({})를 생략하였다. 예제를 실행하면 람다식은 x + y의 결과 값을 리턴한다.

```
interface MyFunction { // 함수형 인터페이스
    int calc(int x, int y); // 람다식으로 구현할 추상 메소드
}

public class LambdaEx1 {
    public static void main(String[] args) {
        MyFunction add = (x, y) -> { return x + y; }; // 람다식
        MyFunction minus = (x, y) -> x - y; // 람다식. {}와 return 생략

        System.out.println(add.calc(1, 2)); // 합 구하기
        System.out.println(minus.calc(1, 2)); // 차 구하기
    }
}
```

→ 실행 결과

3  
-1

# Lambda Expression

- Lambda Expression can be delivered as a parameter



## 예제 4

### 람다식을 매개변수로 전달하기

이 예제에서는 곱을 리턴하는 람다식을 만들어, printMultiply() 메소드의 매개변수로 전달하는 사례를 보인다.

```
interface MyFunction {  
    int calc(int x, int y);  
}  
  
public final class LambdaEx4 {  
    public static void main(String[] args) {  
        printMultiply(3, 4, (x,y)->x*y); // 람다식((x,y)->x*y)을 매개변수로 전달  
    }  
  
    static void printMultiply(int x, int y, MyFunction f) { // f로 (x,y)->x*y  
                                                람다식 전달받음  
        System.out.println(f.calc(x, y));  
    }  
}
```

⇒ 실행 결과

12

# Lambda Expression

- Using Generics



## 예제 5

매개변수로 주어진 객체를 문자열로 출력하는 람다식 만들기

함수형 인터페이스를 제네릭 타입으로 만들고, 람다식에 String 타입과 Integer 타입 객체를 각각 매개변수로 넘겨주는 사례이다.

```
@FunctionalInterface
interface MyFunction<T> { // 제네릭 타입 T를 가진 함수형 인터페이스
    void print(T x); // 람다식으로 구현할 추상 메소드
}

public class LambdaEx5 {
    public static void main(String[] args) {
        MyFunction<String> f1 = (x) -> System.out.println(x.toString());
        f1.print("ABC"); // String 객체를 람다식에 넘겨준다.

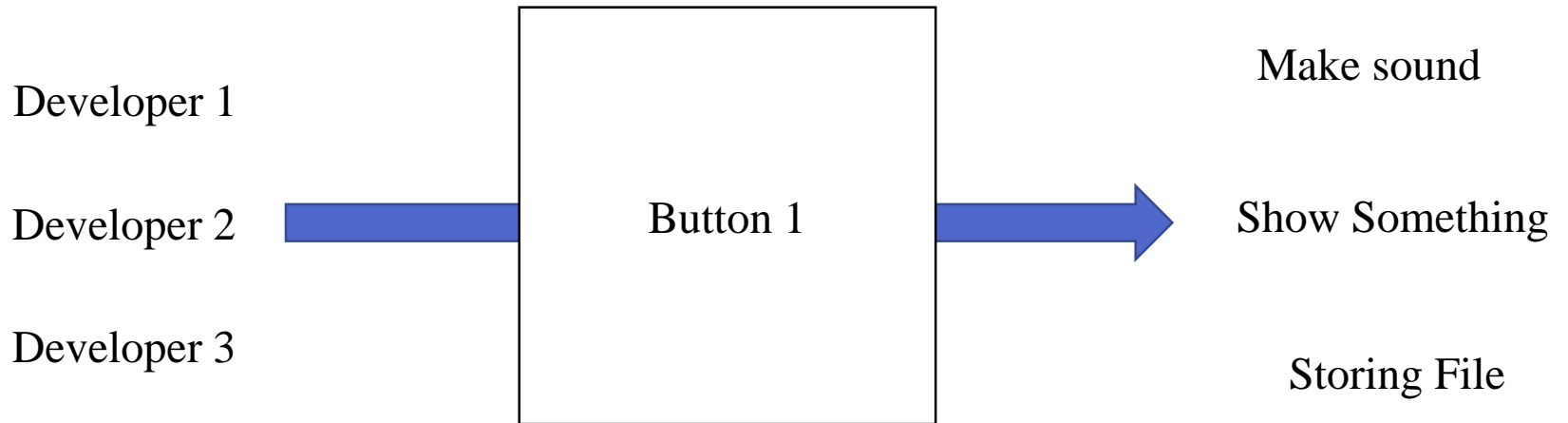
        MyFunction<Integer> f2 = (x) -> System.out.println(x.toString());
        f2.print(Integer.valueOf(100)); // Integer 객체를 람다식에 넘겨준다.
    }
}
```

→ 실행 결과

```
ABC
100
```

# Lambda Expression

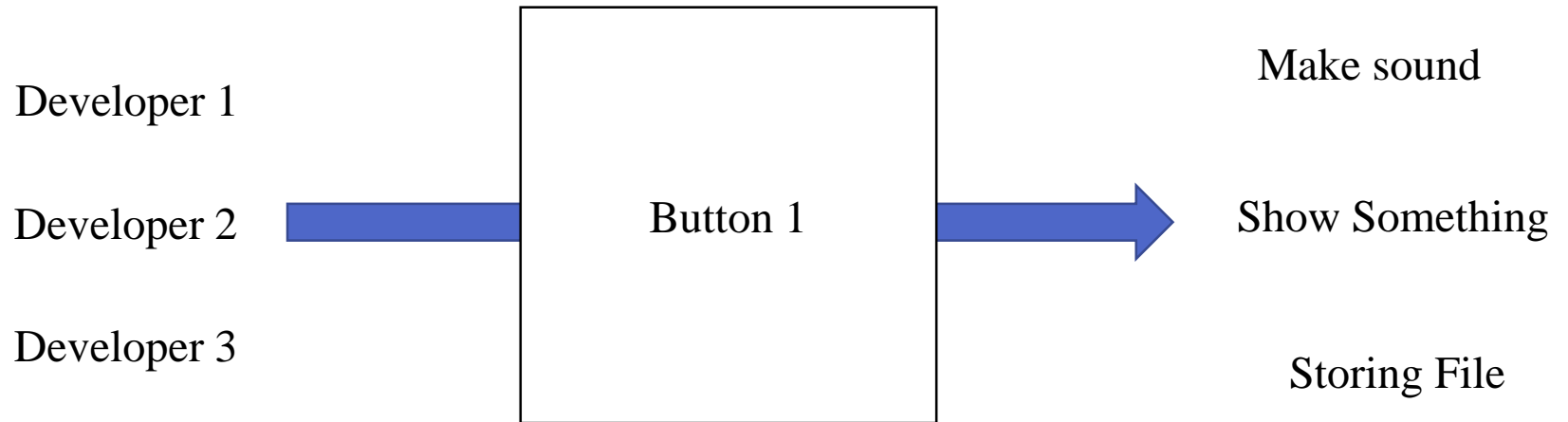
- Java Swing?



```
1 JButton button = new JButton("Click Me!");
2
3 button.addActionListener(new ActionListener() {
4     public void actionPerformed(ActionEvent evt) {
5         System.out.println("Handled by anonymous class listener");
6     }
7 });
```

# Lambda Expression

- Java Swing?



**How about it?**

```
1 button.addActionListener(e -> {  
2     System.out.println("Handled Lambda listener");  
3     System.out.println("Have fun!");  
4 });
```

# Thread-based parallel processing

```
new Thread(new Runnable() {
    @Override
    public void run() {
        for (int i = 0; i < ki.length / 4; i++) {
            Integer k = (Integer) ki[i];
            HashSet<Integer> v = data.get(k);
            for (Integer ve : v) {
                HashSet<Integer> vv = data.get(ve);
                if (vv != null) {
                    for (Integer vve : vv) {
                        HashSet<Integer> vvv = data.get(vve);
                    }
                }
            }
        }
        System.out.println("EF: " + (System.currentTimeMillis() - pre));
    }
}).start();

new Thread(new Runnable() {
    @Override
    public void run() {
        for (int i = ki.length / 4; i < ki.length / 2; i++) {
            Integer k = (Integer) ki[i];
            HashSet<Integer> v = data.get(k);
            for (Integer ve : v) {
                HashSet<Integer> vv = data.get(ve);
                if (vv != null) {
                    for (Integer vve : vv) {
                        HashSet<Integer> vvv = data.get(vve);
                    }
                }
            }
        }
        System.out.println("EF: " + (System.currentTimeMillis() - pre));
    }
}).start();

new Thread(new Runnable() {
    @Override
    public void run() {
        for (int i = ki.length / 2; i < ki.length * 3 / 4; i++) {
            Integer k = (Integer) ki[i];
            HashSet<Integer> v = data.get(k);
            for (Integer ve : v) {
                HashSet<Integer> vv = data.get(ve);
                if (vv != null) {
                    for (Integer vve : vv) {
                        HashSet<Integer> vvv = data.get(vve);
                    }
                }
            }
        }
        System.out.println("EF: " + (System.currentTimeMillis() - pre));
    }
}).start();

new Thread(new Runnable() {
    @Override
    public void run() {
        for (int i = ki.length * 3 / 4; i < ki.length; i++) {
            Integer k = (Integer) ki[i];
            HashSet<Integer> v = data.get(k);
            for (Integer ve : v) {
                HashSet<Integer> vv = data.get(ve);
                if (vv != null) {
                    for (Integer vve : vv) {
                        HashSet<Integer> vvv = data.get(vve);
                    }
                }
            }
        }
        System.out.println("EF: " + (System.currentTimeMillis() - pre));
    }
}).start();
```

VS.

```
data.entrySet().parallelStream().forEach(e -> {
    Integer k = e.getKey();
    Set<Integer> v =
e.getValue().parallelStream().flatMap(e1 -> {
    if (data.containsKey(e1))
return data.get(e1).parallelStream();
else
return null;
}).filter(e2 -> e2 != null).flatMap(e3 -> {
    if (data.containsKey(e3))
return data.get(e3).parallelStream();
else
return null;
}).collect(Collectors.toSet());
System.out.println(k + " " + v);
});
```



# Wrap-up

- Functional Programming and Lambda Expression
- Motivation for Parallel Stream