



2021

프로그래밍 패러다임과 람다식

**PARADIGM & LAMBDA EXPRESSION**



# CONTENTS

---

## 01 프로그래밍 패러다임

명령형 프로그래밍  
논리형 프로그래밍  
패러다임 방식 비교

## 02 람다식

람다식이란?  
람다식 작성문법

## 03 작성예제

계산기



01

# 프로그래밍 패러다임

---

명령형 프로그래밍  
선언형 프로그래밍  
패러다임 방식 비교

# 01 프로그래밍 패러다임

## 명령형 프로그래밍(Imperative Programming)

프로그램이 어떻게 작동할 것인지에 대한 관심을 가지는 관점에서 프로그래밍하는 것

1. 절차적, 객체지향적 언어(JAVA, PASCAL, C, C++, C# 등 대부분의 언어)
2. 메모리에 저장된 명령어들을 순차적으로 실행
3. 변수, 배정문, 반복문, 조건문 등의 명령어를 순차적, 절차적으로 수행
4. 컴퓨터(프로그램)를 어떻게 동작시킬지 알고리즘을 생성하는 것

```
const ARRAY = [10, 20, 30, 40, 50];
document.getElementById('button3').addEventListener("click", event=>sumImpl(ARRAY));
//배열을 파라미터로 받고 각 요소들을 합하는 함수
function sumImpl(arr){
  let result = 0;
  for (let i = 0; i < arr.length; i++){
    result += arr[i];
  }
  return result;
}
```

소스코드

# 01 프로그래밍 패러다임

## 선언형 프로그래밍(Declarative Programming)

프로그램이 무엇을 할 것인지에 대한 관심을 가지는 관점에서 프로그래밍 하는 것

1. 논리, 함수형, 도메인 특화 언어(HTML, CSS, XML, SQL, Prolog, Lisp 등)
2. 어떻게 흘러가는지에 대한 알고리즘이 아닌 무엇을 할 것인지에 관심
3. 약속된 정의를 사용해서 작성

```
const ARRAY = [10, 20, 30, 40, 50];
document.getElementById('button6').addEventListener("click", event=>sumDecl(ARRAY));
//배열을 파라미터로 받고 각 요소들을 합하는 함수
function sumDecl(arr){
  return arr.reduce((prev, current) => prev + current, 0);
}
```

```
<div>
  <label>선언형</label>
  <button id="button4" class="unclicked"> 토클 </button>
  <button id="button5"> 더블 </button><input/>
  <button id="button6"> 합계 </button><input/>
</div>
```

```
SELECT *
FROM USERMP

INSERT INTO USERMP
VALUES ('A', 'B', 'C')
```

소스코드

# 01 프로그래밍 패러다임

## 패러다임 방식 비교

천일아이엔씨에서 근무하는 천일이는 점심밥을 먹으러 티파니돈까스에 가야한다.

<div>명령형</div> <div>선언형</div>	<div>출발</div> <div>천일아이엔씨</div>	<div>유동커피까지</div> <div>62m 이동</div>	<div>부전도서관</div> <div>방면 244m 이동</div>	<div>정향우케익카페</div> <div>오른쪽 91m 이동</div>	<div>동일상회에서</div> <div>오른쪽 40m 이동</div>	<div>도착</div> <div>티파니돈까스</div>
	1분	30초	4분	2분	1분	30초
	<div>출발</div> <div>전포대로 217</div> <div>천일아이엔씨</div>					<div>도착</div> <div>서면로56</div> <div>티파니돈까스</div>

02

# 람다식

---

람다식이란  
람다식 작성문법

## 02 람다식

### 람다식이란(Lambda Expression)

함수형 프로그래밍의 한 기법

1. 메서드를 하나의 간결한 식으로 표현 하는 것
2. 메서드의 이름과 반환타입을 지정하지 않으므로 익명함수라고 부른다.
3. 그 외에 함수형프로그래밍, 람다식, 람다함수 라고도 부른다.
4. 람다식은 클래스, 객체 생성이나 메서드를 호출할 필요가 없다.
5. 람다식은 매개변수로 전달될 수 있고, 변수에 저장이 가능하다. (1급객체)
6. JAVA 기준 JDK 1.8 이상 지원 ( 객체지향 + 함수형패러다임 )
7. 불필요한 코드를 줄이고 가독성을 향상시키는 것을 목적으로 둔다.



## 02 랴다식

### 람다식 작성문법

1. 이름과 반환타입은 작성하지 않는다.
  2. 기본 형태는 (매개변수) -> { body }
  3. 함수형 인터페이스 선언 ( @FunctionalInterface )
    - 내부 인터페이스는 1개의 추상메서드만 구현
- \* (매개변수) 규칙
    - 추론이 가능한 매개변수의 타입은 생략 가능
    - 매개변수가 두 개 이상인 경우, 일부 타입만 생략 불가
    - 매개변수가 하나, 타입이 생략인 경우 괄호 ( ) 생략 가능
  - \* {body} 규칙
    - return문 대신 식으로 대체 가능
    - 식의 끝에는 세미콜론 생략
    - 괄호 { } 안 문장이 하나인 경우 괄호 { } 생략 가능
    - return문은 괄호 생략 불가

03

# 작성예제

---

계산기 1  
계산기 2

# 03 작성예제

## 계산기 1

두 숫자를 받아 더하기, 빼기, 곱하기, 나누기, 나머지값을 구할 수 있는 계산기 프로그래밍(기존 자바 방식)

```
interface Calculator{
    public int add(int num1, int num2);
    public int sub(int num1, int num2);
    public int mul(int num1, int num2);
    public int div(int num1, int num2);
    public int mod(int num1, int num2);
}

public class Calc implements Calculator{
    @Override
    public int add(int num1, int num2) { return num1 + num2; }
    @Override
    public int sub(int num1, int num2) { return num1 - num2; }
    @Override
    public int mul(int num1, int num2) { return num1 * num2; }
    @Override
    public int div(int num1, int num2) { return num1 / num2; }
    @Override
    public int mod(int num1, int num2) { return num1 % num2; }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Calc cal = new Calc();
        System.out.println("두 수의 합은 : " + cal.add(5, 2));
        System.out.println("두 수의 차는 : " + cal.sub(5, 2));
        System.out.println("두 수의 곱은 : " + cal.mul(5, 2));
        System.out.println("두 수의 몫은 : " + cal.div(5, 2));
        System.out.println("두 수의 남은 : " + cal.mod(5, 2));
    }
}
```

코딩결과

두 수의 합은	:	7
두 수의 차는	:	3
두 수의 곱은	:	10
두 수의 몫은	:	2
두 수의 남은	:	1

소스코드

## 03 작성예제

### 계산기 2

두 숫자를 받아 더하기, 빼기, 곱하기, 나누기, 나머지값을 구할 수 있는 계산기 프로그래밍(람다식 활용)

```
@FunctionalInterface
interface LambdaCalculator{
    public int cal(int num1, int num2);
}
```

```
public class CalcLambda{
    public static void main(String[] args) {
        // TODO Auto-generated method stub

        LambdaCalculator addCal = (int num1, int num2) -> { return num1 + num2; };
        LambdaCalculator subCal = (int num1, int num2) -> { return num1 - num2; };
        LambdaCalculator mulCal = (int num1, int num2) -> { return num1 * num2; };
        LambdaCalculator divCal = (int num1, int num2) -> { return num1 / num2; };
        LambdaCalculator modCal = (int num1, int num2) -> { return num1 % num2; };

        System.out.println("두 수의 합은 : " + addCal.cal(5, 2));
        System.out.println("두 수의 차는 : " + subCal.cal(5, 2));
        System.out.println("두 수의 곱은 : " + mulCal.cal(5, 2));
        System.out.println("두 수의 몫은 : " + divCal.cal(5, 2));
        System.out.println("두 수의 남은 : " + modCal.cal(5, 2));
    }
}
```



```
addCal = (num1, num2) -> { return num1 + num2; };
subCal = (num1, num2) -> { return num1 - num2; };
mulCal = (num1, num2) -> { return num1 * num2; };
divCal = (num1, num2) -> { return num1 / num2; };
modCal = (num1, num2) -> { return num1 % num2; };
```

```
addCal = (num1, num2) -> num1 + num2;
subCal = (num1, num2) -> num1 - num2;
mulCal = (num1, num2) -> num1 * num2;
divCal = (num1, num2) -> num1 / num2;
modCal = (num1, num2) -> num1 % num2;
```

코딩결과

```
두 수의 합은 : 7
두 수의 차는 : 3
두 수의 곱은 : 10
두 수의 몫은 : 2
두 수의 남은 : 1
```

소스코드

# THANK YOU

SI 1팀 사원 이성진

# 출 처

[http://www.tcpschool.com/java/java\\_lambda\\_concept](http://www.tcpschool.com/java/java_lambda_concept)

<https://galid1.tistory.com/509>

<https://atoz-develop.tistory.com/entry/JAVA-%EB%9E%8C%EB%8B%A4%EC%8B%9DLambda-Expression>

<https://boxfoxs.tistory.com/430>