JAVA 프로그래밍

Chapter 22 람다식

목차

- 자바 람다식(Lambda Expressions in Java)
- FunctionalInterface

Lambda Expressions

❖ Lambda Expressions

- 식별자 없이 실행 가능한 함수 표현식.
- 람다식은 익명 함수(anonymous function)를 생성하기 위한 식으로 객체 지향 언어보다는 함수 지향 언어에 가깝다.
- 람다식의 형태는 매개 변수를 가진 코드 블록이지만, 런타임 시에는 익명 구현 객체를 생성한다.
- 기존의 불필요한 코드를 줄이고 가독성을 향상시키는 것에 목적을 두고 있다.
- 자바 8에서 추가된 가장 특징적인 기능.

❖ Lambda Expressions Example

- 자바에서 람다식을 사용하려면 다음과 같은 방법으로 사용이 가능합니다.

```
( parameters ) -> expression body
( parameters ) -> { expression body }
( ) -> expression body
( ) -> { expression body }
```

Lambda Expressions

❖ 기존방식

- "Hello World."라는 단어를 출력하고 종료하는 쓰레드

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

❖ Lambda Expressions Code

- 인자가 없기 때문에 ()로 작성하고 실제로 동작할 코드를 ->{ ... }의 내부에 작성한다.
- () -> { expression body } 구조입니다.

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

@FunctionalInterface

❖ 함수적 인터페이스(@FunctionalInterface)

- 람다식이 하나의 메서드를 정의하기 때문에 두 개 이상의 추상 메서드를 가질 수 없다.
- 하나의 추상 메서드가 선언된 인터페이스를 함수적 인터페이스(functional interface)라고 한다.
- @FunctionalInterface 어노테이션을 붙여 사용한다.

```
@FunctionalInterface
public interface MyFunctionalInterface{
    public int method(int x, int y);
}
```