Chapter 14 함수형 프로그래밍, 람다식, 스트림

14.1 함수형 프로그래밍 소개

프로그래밍 패러다임 분류

- 프로그래밍 패러다임을 크게 나누면 명령형 프로그래밍(Imperative programming)과 선언적 프로그래밍(declarative programming)으로 나눌 수 있다.
- 함수형 프로그래밍(functional programming): 선언적 프로그래밍의 한 형태로 프로그래밍을 순수함수의 적용으로 생각하는 방법론

명령형 프로그래밍 방법론

- 명령형 프로그래밍은 작업을 어떻게 수행하느냐를 중시한다.

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

public class Imperative {
    public static void main(String [] args) {
        List<Integer> list = List.of(12, 3, 16, 2, 1, 9, 7, 20);
        List<Integer> even = new ArrayList<>();

        for(Integer e : list) {
            if (e % 2 == 0)
                even.add(e);
        }

        for (Integer e : even) {
            System.out.println(e);
        }
    }
}
```

함수형 프로그래밍 방법론

- 선언적 프로그래밍은 해야 할 일에 집중한다.

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

public class Functional {
    public static void main(String [] args) {
        List<Integer> list = List.of(12, 3, 16, 2, 1, 9, 7, 20);
        list.stream() // 리스트 안의 원소를 하나씩 추출하는 메소드
            .filter(e -> e % 2 == 0) // 들어오는 정수 중에서 찍수만 추려내는 메소드
            .forEach(System.out::println); // 들어오는 각 정수에 대해 전달받은 함수
```

```
를 적용한다.
    }
}
```

왜 함수형 프로그래밍인가?

- Stream API: JAVA에서 지원하는 함수형 프로그래밍
- 함수형 프로그래밍은 병렬 처리를 쉽게 한다. 함수형 프로그래밍에서는 부작용 없는 순수 함수만을 사용하기 때문에 코어를 여러 개 사용하여도 서로 간에 복잡한 문제가 발생하지 않는다.

함수란 무엇인가?

- 함수에 부작용이 있다 : 함수가 실행되면서 외부의 변수를 변경한다.
- 순수 함수(pure function): 함수형 프로그래밍에서 함수. 부작용이 없다. 즉 외부 상태를 변경하지 않는다. 스레드에 안전하고 병렬적인 계산이 가능하다.

자바와 함수형 프로그래밍

- 자바에서 순수 함수로만 프로그램을 작성하는 것은 어렵다.
- 자바는 순수한 함수형 프로그래밍은 아니고 함수형 스타일을 지원한다.
- 순수 함수가 아니라면 여러 개의 스레드가 동시에 함수 코드를 실행할 수 없다.
- 자바에서 함수형 스타일을 하려면
- 지역 변수만을 변경할 수 있어야 한다.
- 참조하는 객체는 변경할 수 없어야 한다.

객체 지향 프로그래밍과 함수형 프로그래밍

- 자바 프로그래머들은 객체 지향 스타일과 함수형 스타일을 적절하게 구사할 수 있어야 한다.

Object-Oriented Programming	**Functional Programming**
객체에 기반을 둔다.	함수 호출이 기본 프로그래밍 블록이다.
명령형 프로그래밍 모델이다.	선언적 프로그래밍이다.
병령 처리를 지원하지 않는다.	병렬 처리를 지원한다.
객체와 메소드가 기본 요소다.	변수와 순수 함수가 기본 요소다.

14.2 람다식

함수의 1급 시민 승격

- 메소드: 클래스 안에서 정의된 함수
- 1급 시민(first-class citizen): 기초형의 값이나 객체, 배열처럼 모든 연산이 허용된 엔터티. 변수에 저장되거나 함수의 인자가 되거나 함수에서 반환될 수 있다.
- 2급 시민(second-class citizen): 값이 아니어서 위와 같은 것들이 불가능하다.
- Java 8 전에는 함수가 값이 아니었지만 Java 8에서 함수가 1급 시민으로 승격되며 다음과 같은 것이 가능해졌다.

1. 함수도 변수에 저장될 수 있다.
2. 함수를 매개 변수로 받을 수 있다. (-> 동작 매개 변수화를 가능하게 한다.)

3. 함수를 반환할 수 있다.

람다식의 필요성

람다식이란?

- Lamba expression(람다식): 나중에 실행딜 목적으로 다른 곳에 전달될 수 잇는 코드 블록
- 이름 없는 함수
- 간결함
- 함수가 필요한 곳에 간단히 함수를 보낼 수 있다.
- 함수가 딱 한 번만 사용되고 함수의 길이가 짧은 경우에 유용하다.

람다식의 정의

- 람다식 매개 변수, 람다식 연산자, 람다식 몸체

```
(int a, int b) -> { return a + b; }
```

- 람다식은 0개 이상의 매개 변수를 가질 수 있다.
- 화살표(->)는 람다식에서 매개 변수와 몸체를 구분한다.
- 문맥에 추정될 수 있을 경우에 매개 변수의 형식을 명시적으로 선언하지 않아도 된다.
- 단일 매개 변수이고 타입이 유추 가능한 경우에 괄호를 생략할 수 있다.
- 몸체에 하나 이상의 문장이 있으면 중괄호롷 묶어야 한다.

람다식의 활용

1. 자바 GUI에서 함수 몸체를 전달하고 싶을 때 익명 클래스를 사용한다.
2. 스레드를 작성할 때
3. 배열의 모든 요소를 출력할 때 forEach() 와 같은 함수형 프로그래밍을 사용할 수 있다.

```
// 이전의 방법
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerfromed(ActionEvent e) {
        System.out.println("버튼 클릭");
    }
})

// 람다식을 이용한 방법
button.addActionListener((e) -> {
    System.out.println("버튼 클릭!");
});
```

```
// 이전의 방법
new Thread(new Runnable() {
    @Override
```

```
    public void run() {
        System.out.println("스레드 실행");
    }
}).start();

// 람다식을 이용한 방법
new Thread(() -> System.out.println("스레드 실행")).start();
```

```
// 이전의 방법
List<Integer> list = Arrays.asList(1, 2, 3 ,4 ,5);
for (Integer n : list) {
    System.out.println(n);
}

// 람다식을 이용한 방법
List<Integer> list = Arrays.asList(1, 2, 3 ,4 ,5);
list.forEach(n -> System.out.println(n));
```

14.3 동작 매개 변수화

14.4 함수형 인터페이스

14.5 메소드 참조

14.6 스트림 API