3주차 과제

전공: 컴퓨터공학과 학년: 3학년 학번: 20211547 이름: 신지원

#1. Kotlin Iterator 상세 설명

Iterator는 말 그대로 반복하는 요소를 말한다. Kotlin 의 기반이 되는 자바의 collection framework 에서 collection에 저장되어 있는 요소들을 읽어오는 방법을 표준화한 것이다. Iterator은 collection 의 모든 요소를 하나하나 처리하는 데 유용하다. 하지만 Iterator 가 마지막 element 를 통과한 뒤엔 새로운 Iterator 를 생성하지 않는 한, 더 이상 element 를 검색하는 데 사용될 수 없으며 이전 위치로 재설정할 수도 없다. 이러한 특징으로 for 문, forEach 문, while 문 등 과 함께 사용되곤 한다.

Iterator#1.kt

```kotlin
1   val jiwonlikes = listof("coffee", "flower", "kuromi", "meat")
2   val jiwonlikesIterator = jiwonlikes.iterator()
3   while (jiwonlikesIterator.hasNext()) {
4       println(jiwonlikesIterator.next())
5   }
6
7   // output:
8   // coffee
9   // flower
10  // kuromi
11  // meat
12
```

1번 줄에서는 jiwonlikes 라는 이름의 리스트를 생성하고, "coffee", "flower", "kuromi", "meat" 라는 문자열 요소들로 초기화한다. 이는 listof 함수를 통해 불변 리스트로 반환된다. 2번 줄에서는 jiwonlikes.iterator() 를 호출하고 있다. 이는 jiwonlikes 리스트의 반복자를 생성한다. jiwonlikesIterator 는 jiwonlikes 를 순회할 수 있으며, 생성된 반복자는 jiwonlikesIterator 변수에 할당된다. 3-5번 줄에서는 while 문을 통한 순회가 이루어지고 있다. 'while(jiwonlikesIterator.hasNext())' 는 jiwonlikesIterator 가 다음 요소를 가지고 있는지를 확인하며 'hasNext()' 메소드는 더 이상 순회할 요소가 없을 때까지 'true'를 반환한다. 따라서 리스트의 모든 요소를 순회할 때까지 while 루프가 실행되는 것이다. 'println(jiwonlikesIterator.next())' 를 통해서는 현재 요소를 출력하고 'next()' 메소드 호출을 통해 반복자의 현재 요소를 반환하고, 반복자를 다음 요소로 이동시킨다.

위에서 사용된 Method 함수 hasNext(), next() 를 구체적으로 살펴보자.

type.kt

```kotlin
public interface Iterator<out T> {
    public operator fun hasNext(): Boolean
    public operator fun next(): T
}
```

- hasNext() 은 Boolean을 반환한다. 이 메서드는 Iterator가 다음 요소를 가지고 있는지 여부를 확인합니다. 만약 더 이상 반복할 요소가 없다면 false를 반환한다.
- next(): Type을 반환한다. Iterator의 다음 요소를 반환하며, 더 이상 반환할 요소가 없을 경우 NoSuchElementException을 발생시킨다. 만약 List에서 1번 째에 있는 요소를 가져오고 싶다면 next() 를 1번, 3번 째에 있는 요소를 가져오고 싶다면 3번을 호출해야 한다.

Kotlin 표준 라이브러리는 Iterator 인터페이스 외에도 MutableIterator, ListIterator, MutableListIterator 와 같은 여러 확장된 반복자 인터페이스를 제공하는데, 이들은 add(), set(), remove(), add(), hasPrevious(), previous() 와 같은 추가적인 method 함수를 제공한다.

#2. 아래 예제 코드를 Iterator를 사용하여 while loop로 변환하고, 1..10의 데이터 타입인 intRange에 대하여 조사하여 변환 후 코드의 동작 원리 설명

Iterator를 사용하여 "for (i in 1..10) print(i)" 과 같은 결과를 나타내는 while loop 를 보이자면 아래와 같다.

Iterator#2.kt

```kotlin
val intRange = 1..10
val iterator = intRange.iterator()

while (iterator.hasNext()) {
    val i = iterator.next()
    print(i)
}
```

1번 줄에서는 intRange 변수를 통해 1부터 10까지의 정수 범위를 선언하였고, 2번 줄에서는 1번에서 선언한 변수에 대한 반복자를 생성하였다. 4-7번 줄에서는 while 문을 통해 범위를 순회하고 있다. 이때, iterator.hasNext() 라는 method 를 통해 interator 가 끝에 도달했는지를 확인하며, iterator.next() 를 통해 다음 차례로 넘어가도록 구현하였다.

intRange 란 start..end 형태로 표현되며, start는 시작 값을, end는 끝 값을 나타낸다. 예시 코드에서 1...10 의 범위를 나타낸 것은 1부터 10 의 모든 정수를 포함하는 범위를 나타낸 것이다. 이때, 한 번 생성된 범위는 변경할 수 없기 때문에 시작 값과 끝 값은 고정되어 있다. Kotlin 공식문서에

서는 `IntRange(start: Int, endInclusive: Int)` 와 같이 정의하고 있는데, 'endInclusive' 라는 단어를 통해 끝 값을 포함하고 있다는 것을 한 번 더 확인할 수 있다. 주요 method 에는 first, last, step, isEmpty, contains, iterator 등이 있다. 이와 같이 intRange 는 반복문, 조건문 등을 간단하게 구현할 수 있도록 돕는다.

이를 통하여 for 문과 while 문의 차이점에 대해 알 수 있다. (i in 1..10) 과 같이 범위를 지정해주어야 하는 for 문과 달리, while 문은 반복의 횟수를 선언하지 않아도 실행할 수 있다. 하지만 무한 루프에 빠질 수 있다는 위험성이 있다.

#3. forEach 함수 source code를 제시하여 동작 상세 설명

Kotlin 에서 forEach 문은 순서가 있는 데이터에 대한 확장 함수로, 외부에서 반복하는 기존의 for 문에서 내부 반복으로 바꿔줄 수 있는 함수다. 이는 가독성을 향상시킬 수 있다. 아래는 forEach 문을 사용하여 작성한 예시 코드다.

Iterator#3.kt

```kotlin
val jiwonlikes = listOf("coffee", "flower", "kuromi", "meat")

jiwonlikes.forEach { like ->
    println(like)
}

// output:
// coffee
// flower
// kuromi
// meat

```

1번 줄에서는 jiwonlikes 라는 이름의 리스트를 생성하고, "coffee", "flower", "kuromi", "meat" 라는 문자열 요소들로 초기화한다. 이는 listOf 함수를 통해 불변 리스트로 반환된다. 3번 줄에서는 'like -> ' 의 표현을 통해 jiwonlikes 리스트의 각 요소에 대해 반복 실행될 람다 표현식을 받는다. 이때 forEach 문을 통해 내부의 모든 요소를 순회하며 차례로 가리키게 되고, 각 요소에 대해 인자로 전달된 람다 표현식을 실행한다. 람다 표현식을 간결하게 사용할 수도 있는데, 이는 아래와 같다.

```kotlin

jiwonlikes.forEach {
    println(it)
}

```

만약 람다 표현식이 하나의 매개변수만 받을 경우엔 그 매개변수의 이름 대신 'it'으로 대체할 수 있다. 이때 'it'은 현재 순회 중인 리스트의 요소를 가리킨다.

앞서 for 문은 외부 반복, forEach 문은 내부 반복이라고 언급하였다. 위 코드를 for 문으로 변경한 코드는 아래와 같다.

Iterator#3-1.kt

```kt
val jiwonlikes = listOf("coffee", "flower", "kuromi", "meat")

for (like in jiwonlikes) {
    println(like)
}

```

위처럼 직접 범위를 지정해주며 외부 반복을 하는 것을 알 수 있다. 뿐만 아니라 for 문과, forEach 문은 continue, break 문 사용에 차이가 있다. for 문과 달리 forEach 문은 이를 사용할 수 없다. 강제로 사용하는 방법도 있는데, continue 문은 run , break 문은 return@run 을 사용해야 한다.