

이펙티브 자바

Item 47 ~ 49

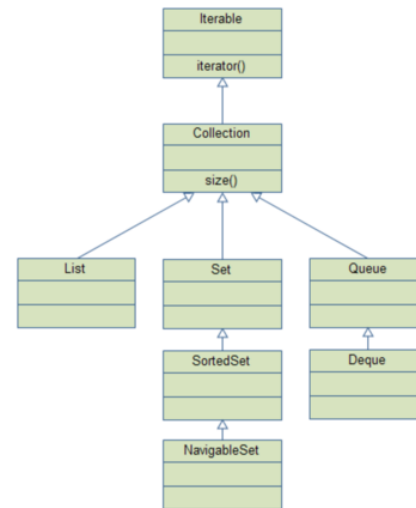
2024/04/30

Item 47 반환 타입으로는 스트림보다 컬렉션이 낫다.

자바8 이전의 시퀀스 반환: Collection, Iterable

자바8 stream 등장이후 시퀀스 반환 타입에 대한 논의

스트림으로 반환 해야 할까?



Item 47 반환 타입으로는 스트림보다 컬렉션이 낫다.

Stream은 반복을 지원하지 않는다.

한 값을 다른 값에 매핑하면 원래 값을 잃어 버린다.(item 45)



```
List<Integer> list=new ArrayList<>();  
IntStream.range(0,10).forEach(list::add);  
  
List<Integer> afterList=new ArrayList<>();  
list.forEach(num->afterList.add(num*2));
```

Item 47 반환 타입으로는 스트림보다 컬렉션이 낫다.

어댑터를 구현하여 iterator와 stream을 변환한다.



```
public static<E>Stream<E>streamOf(Iterable<E> iterable){  
    return StreamSupport.stream(iterable.spliterator(),false);  
}
```



```
public static<E>Iterable<E>iterableOf(Stream<E> stream){  
    return stream::iterator;  
}
```

Item 47 반환 타입으로는 스트림보다 컬렉션이 낫다.

객체 시퀀스를 반환해야하는 경우

오직 스트림 파이프라인에서 사용하는 경우 : Stream

반환한 객체를 반복문에 사용하는 경우 : Iterable

Collection은 Iterable의 자손 + Stream 함수 지원

Item 47 반환 타입으로는 스트림보다 컬렉션이 낫다.

배열도 Collection 으로 변환하여 Stream으로 사용

asList, forEach



```
Integer[] intArr={1,2,3,4,5};  
streamOf(Arrays.asList(intArr))  
    .forEach(System.out::println);
```

Arrays.asList() VS List.of()

Arrays.asList 변경가능, 추가 불가능

List.of 변경 불가능, 추가 불가능

```
private final E[] a;  
  
ArrayList(E[] array) {  
    a = Objects.requireNonNull(array);  
}
```

Arrays.asList

```
static <E> List<E> of(E e1) {  
    return new ImmutableList.List12<>(e1);  
}
```

List.of

Item 47 반환 타입으로는 스트림보다 컬렉션이 낫다.

리스트의 모든 부분 리스트를 생성

스트림은 어렵다..



```
public static <E> Stream<List<E>> of(List<E>list){  
    return Stream.concat(Stream.of(Collections.emptyList()),  
        prefixes(list).flatMap(SubLists::suffixes));  
}
```

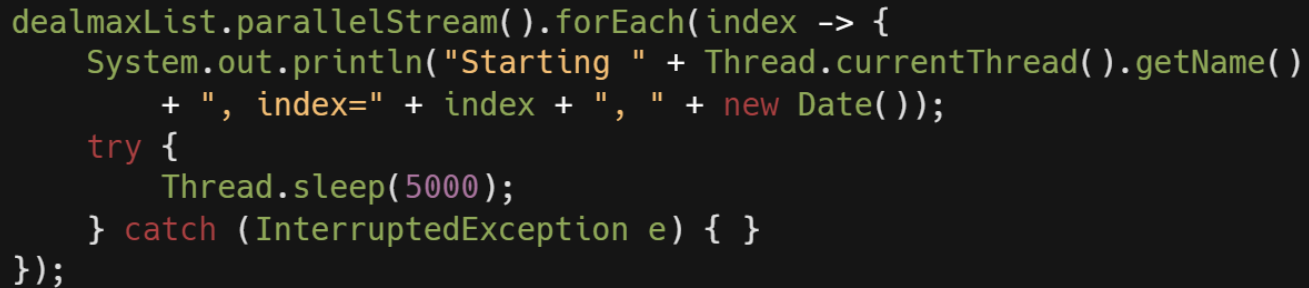

Item 47 반환 타입으로는 스트림보다 컬렉션이 낫다.

결론: Collection을 사용하자



Item 48 스트림 병렬화는 주의해서 적용하라

자바 8부터 등장한 `stream().parallel()`, `parallelStream()`



```
dealmaxList.parallelStream().forEach(index -> {  
    System.out.println("Starting " + Thread.currentThread().getName()  
        + ", index=" + index + ", " + new Date());  
    try {  
        Thread.sleep(5000);  
    } catch (InterruptedException e) { }  
});
```

Item 48 스트림 병렬화는 주의해서 적용하라

Stream.iterate , limit의 경우 병렬 연산으로 이점을 얻기 어렵다.

병렬연산은 생각할게 너무 많아요..

**ArrayList, HashMap, HashSet, ConcurrentHashMap, 배열, int
range, long range..**

Item 48 스트림 병렬화는 주의해서 적용하라

참조 지역성: 최근 접근한 리소스와 주변 리소스를 다시 접근할 확률이 높다.

벌크 연산에서 중요하다.

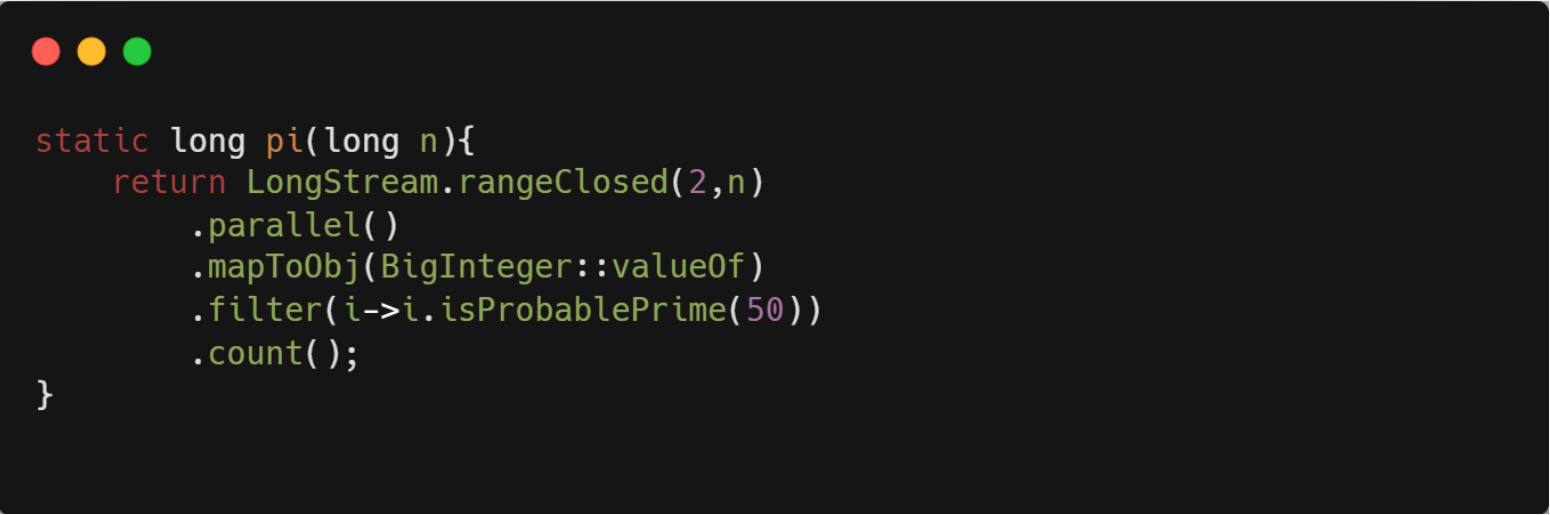
축소 연산(벌크 단위에서 개별로 진행 할 수 있는 연산)

(min, max, count...)

~~가변 축소연산~~

Item 48 반환 타입으로는 스트림보다 컬렉션이 낫다.

Flat한 연산의 경우 병렬처리를 고려할 만 하다.



```
static long pi(long n){  
    return LongStream.rangeClosed(2,n)  
        .parallel()  
        .mapToObj(BigInteger::valueOf)  
        .filter(i->i.isProbablePrime(50))  
        .count();  
}
```

Item 48 반환 타입으로는 스트림보다 컬렉션이 낫다.

결론: 병렬화는 확신할 수 없다면 사용하지 말자

만약 사용한다면 성능 향상을 반드시 검증해야 한다



Item 49 매개변수가 유효한지 검사하라

메서드와 생성자는 유효성 검사를 하지 않는다.

인덱스 값은 0또는 양수여야 한다. 객체 참조는 null이 아니어야 한다.

유효성을 빨리 검증할 수록 디버깅이 쉽다.

Item 49 매개변수가 유효한지 검사하라

Object.requireNonNull()

때때로 즉시 유효성을 검사하는 것이 비효율적일 수 있다.

검증 규칙을 위반하더라도 큰 손해가 없다면 유효성 검사를 미룰 수 있다.

실패원자성을 해칠 수 있음을 유의하자

Item 49 매개변수가 유효한지 검사하라

실패원자성 이란

실패(Error or Exception)했을 경우, 실패전 상태를 유지해야한다.

불변 객체는 실패 원자성을 보장한다.

Item 49 매개변수가 유효한지 검사하라

결론: 항상 메서드나 생성자는 코드 시작 전 유효성 검증을 해볼것

