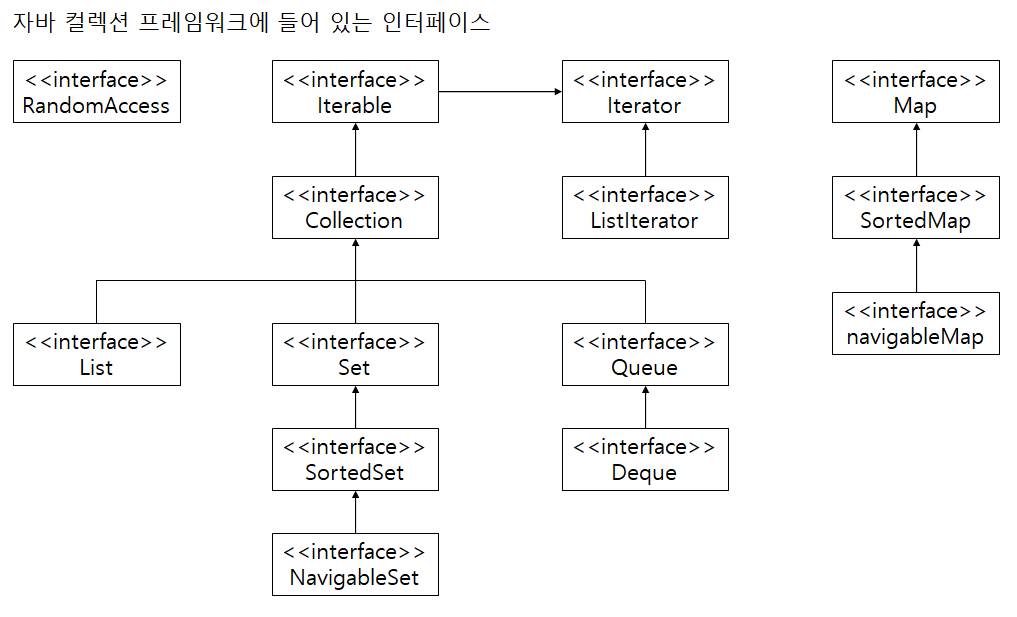
일자 : 2019. 01. 30 20:00 ~ 22:00 (120분)

참석자 : 김경민, 박지환, 임현철

1. 7장 컬렉션
   1. 컬렉션 프레임워크 개요
      * 컬렉션 프레임워크는 선택한 자료 구조에 독립적인 코드를 쉽게 작성할 수 있는 공통인터페이스를 제공한다.



* + - Collection<E> 인터페이스의 메서드

|  |  |
| --- | --- |
| 메서드 | 설명 |
| boolean add(E e)  boolean addAll(Collection<? extends E> c) | e를 추가하거나 c에 들어있는 요소를 추가한다. 컬렉션이 변경되면 true를 반환한다. |
| boolean remove(Object o)  boolean removeAll(Collection<?> c)  Boolean retainAll(Collection<?> c)  Boolean removeIf(Predicate<? super E> filter)  void clear() | 각각 o객체, c에 들어있는 요소, c에 없는 요소, 일치하는 요소, 모든 요소를 제거한다. 처음 메서드 네 개는 컬렉션이 변경되면 true를 반환한다. |
| int size() | 컬렉션에 들어있는 요소의 개수를 반환한다. |
| Boolean isEmpty()  Boolean contains(Object o)  Boolean containsAll(Collection<?> c) | 컬렉션이 비어있거나, o를 포함하거나, c에 들어있는 모든 요소를 포함하면 true를 반환한다. |
| Iterator<E> iterator()  Stream<E> stream()  Stream<E> parallelStream()  Spliterator<E> spliterator() | 컬렉션의 요소를 방문하는 반복자, 스트림 (아마도) 병렬 스트림, 분할반복자(spliterator)를 돌려준다. |
| Object[] toArray()  T[] toArray(T[] a) | 컬렉션의 요소를 담은 배열을 반환한다. 두번째 메서드는 a의 길이가 충분하면 a배열에 요소를 담아 반환한다. |

* + - List 인터페이스의 메서드

|  |  |
| --- | --- |
| 메서드 | 설명 |
| boolean add(int index, E e)  Boolean addAll(int index, Collection<?  extends E> c)  Boolean add(E e)  Boolean addAll(Collection<? extends E> c) | index의 앞이나 리스트의 끝에 e 또는 c에 들어 있는 요소를 추가한다. 리스트가 변겨되면 true를 반환한다. |
| E get(int index)  E set(int index, E element)  E remove(int index) | 지정한 인덱스에 있는 요소를 얻거나 설정하거나 제거한다. 마지막 두 메서드는 호출 이전에 해당 인덱스에 있던 요소를 반환한다. |
| int indexOf(Object o)  int lastIndexOf(Object o) | o와 동일한 처음 또는 마지막 요소의 인덱스를 반환한다. 일치하는 요소가 없으면 -1을 반환한다. |
| ListIterator<E> listIterator()  ListIterator<E> listIterator(int index) | 모든 요소 또는 index에서 시작하는 요소에 대응하는 리스트 반복자를 돌려준다. |
| void replaceAll(UnaryOperator<E> operator) | 각 요소에 연산자를 적용한 결과로 각 요소를 교체한다. |
| void sort(Comparator<? super E> c) | c로 지정한 순서 규칙에 따라 이 리스트를 정렬한다. |
| static List<E> of(E… elements) | 지정한 요소들을 담은 수정 불가 리스트를 돌려준다. |
| List<E> subList(int fromIndex, int toIndex) | fromIndex에서 시작해 toIndex 앞에서 끝나는 서브리스트의 뷰를 돌려준다. |

* + - Collections 클래스의 유용한 메서드

|  |  |
| --- | --- |
| 메서드 | 설명 |
| boolean disjoint(Collection<?> c1, Collection<?> c2) | 두 컬렉션에 공통된 요소가 없으면 true를 반환한다. |
| Boolean addAll(Collection<?super T> c, T...  elements) | 모든 요소를 c에 추가한다. |
| void copy(List<? super T> dest, List<?  extends T> src) | src의 모든 요소를 dest에 같은 인덱스로 저장한다(dest의 길이는 src 이상이어야 한다.) |
| Boolean replaceAll(List<T> list, T oldVal, T newVal) | oldVal 요소를 모든 newVal로 교체한다(둘중 하나가 null일 수도 있다). 일치하는 요소가 하나라도 있으면 true를 반환한다. |
| void fill(List<? super T> list, T obj) | 리스트의 모든 요소를 obj로 설정한다. |
| List<T> nCopies(int n, T o) | o의 사본 n개를 담은 불변 리스트를 돌려준다. |
| int frequency(Collection<?> c, Object o) | c에 들어있는 요소 중 o와 동일한 요소의 개수를 반환한다. |
| int indexOfSubList(List<?> source, List<?> target)  int lastIndexOfSubList(List<?> source,  List<?> target) | source 리스트에서 target 리스트가 처음 또는 마지막으로 나타난 시작 인덱스를 반환한다. target 리스트를 찾을 수 없으면 -1을 반환한다. |
| int binarySearch(List<? extends Comparable<>  super T>> list, T key)  int banarySearch(List<? extends T> list,  T key, Comparator<? super T> c) | 리스트가 지연 순서 또는 c로 정렬되어 있다고 가정한 상태에서 키 위치를 반환한다. 키가 없으면 –i -1을 반환한다. 이때 i는 해당 키를 삽입해야 하는 위치다. |
| sort(List<T> list)  sort(List<T> list, Comparator<? super T> c) | 지연 순서 또는 c를 이용해 리스트를 정렬한다. |
| void swap(list<? list, int I, int j) | 지정한 위치에 있는 요소들을 교환한다. |
| void rotate(List<?> ilist, int distance) | 인덱스 i에 있는 요소를 인덱스 (I + distance) % list.size()로 옮기는 방식으로 리스를 순환한다. |
| void reverse(List<?> list)  void shuffle(List<?> list)  void shuffle(List<?> list, Random rnd) | 리스트를 뒤집거나 무작위로 뒤섞는다 |
| synchronized(Collection|List|Set|SortedSet|  NavigableSet|Map|SortedMap|NavigableMap)() | 동기화 뷰를 돌려준다 |
| unmodifiable(Collection|List|Set|SortedSet|  NavigableSet|Map|SortedMap|NavigableMap)() | 수정 불가 뷰를 돌려준다 |
| checked(Collection|List|Set|SortedSet|  NavigableSet|Map|SortedMap|NavigableMap|Queue)() | 검사 뷰를 돌려준다 |

* 1. 반복자
     + 각 컬렉션은 특정 순서로 요소를 순환하는 메서드를 제공한다.
     + Collection의 슈퍼 인터페이스 Iterable<T>에는 다음 메서드가 선언되어 있다.  
         
       Iterator<T> iterator()  
         
       이 메서드는 모든 요소를 방문하는 데 사용할 수 있는 반복자를 돌려준다.

|  |
| --- |
| Collection<String> coll = ...;  Iterator<String> iter = coll.iterator();  while (iter.hasNext()) {  String element = iter.next();  element*를 처리한다.*  } |

* + - remove메서드는 방금 방문한 요소를 제거한다.

|  |
| --- |
| while (iter.hasNext()) {  String element = iter.next();  if (element*가 조건을 만족한다*)  iter.remove();  } |

* 1. 집합
     + 순서가 유용하지 않을 떄 유용
     + SortedSet(E)의 메서드

|  |  |
| --- | --- |
| 메서드 | 설명 |
| E first()  E last() | 집합의 처음과 마지막 요소를 반환한다. |
| SortedSet<E> headset(E toElement)  SortedSet<E> subset(E fromElement, E toElement)  SortedSet<E> tailSet(E fromElement) | fromElement에서 시작해 toElement앞에서 끝나는 요소들의 뷰를 반환한다. |

* + - NavigableSet(E)의 메서드

|  |  |
| --- | --- |
| 메서드 | 설명 |
| E higher(E e)  E ceiling(E e)  E floor(E e)  E lower(E e) | > | ≥ | ≤ | < e 인 요소 중 가장 가까운 요소를 반환한다. |
| E pollFirst()  E pollLast() | 처음 요소나 마지막 요소를 제거하고 반환한다. 집합이 비어 있으면 null을 반환한다. |
| NavigableSet<E> headset(E toElement, Boolean inclusive)  NavigableSet<E> subset(E fromElement, boolean fromInclusive,  E toElement, Boolean toExeclusive)  NavigableSet<E> tailSet(E fromElement, Boolean inclusive) | fromElement에서 toElement(포함 또는 미포함) 사이에 있는 요소들의 뷰를 반환한다. |

* 1. 맵
     + 맵(map)은 연관된 키(key)와 값(value)을 저장한다.
     + 연관된 키와 값을 새로 추가허가나 기존 키 값을 변경할 때는 put을 호출한다

|  |
| --- |
| Map<String, Integer> counts = new HashMap<>();  counts.put(“Alice”, 1); // 맵에 키/값 쌍으로 추가한다.  counts.put(“Alice”, 2); // 키에 대응하는 겂을 업데이트 한다. |

* + - 키와 연관된 값을 얻는 방법

|  |
| --- |
| int count = counts.getOrDefault(“Alice”, 0); |

* + - 다음 메서드를 호출하면 각각 맵의 키, 값, 엔트리의 뷰를 얻을 수 있다.

|  |
| --- |
| Set<K> keyset()  Set<Map.Entry<K, V>> entrySet()  Collection<V> values() |

* + - 맵에 들어있는 모든 키와 값을 순회하려면 다음과 같이 entrySet메서드가 반환하는 집합을 순회하면 된다.

|  |
| --- |
| for (Map.Entry<String, Integer> entry : counts.entrySet()) {  String k = entry.getKey();  Integer v = entry.getValue();  k, v*를 처리한다.*  } |

* 1. 기타 컬렉션
     + 주요 시스템 프로퍼티

|  |  |
| --- | --- |
| 프로퍼티 키 | 설명 |
| user.dir | 가상 머신의 ‘현재 작업 데리겨터리’ |
| user.home | 사용자 홈 디렉터리 |
| user.name | 사용자 계정 이름 |
| java.version | 가상 머신의 자바 실행 환경 버전 |
| java.home | 자바 설치본의 홈 디렉터리 |
| java.class.path | 가상 머신을 실행하는 데 사용한 클래스 패스 |
| java.io.tmpdir | 임시 파일 저장에 적합한 디렉터리(예를 들어 /tmp) |
| os.name | 운영 체제 이름 |
| os.arch | 운영체제의 아키텍쳐 |
| os.version | 운영체제 버전 |
| file.separator | 파일 분리자 |
| path.separator | 경로 분리자 |
| line.separator | 줄 분리자 |

* 1. 뷰
     + 컬렉션 뷰는 컬렉션 인터페이스를 구현하지만 요소를 저장하지 않는 경량 객체다.
  2. 핵심 내용 정리
     + Colletcion 인터페이스는 Map 인터페이스로 나타내는 맵을 제외하고 모든 컬렉션에 공통 메서드를 제공한다.
     + 리스트는 각 요소에 점수 인덱스가 붙은 순차 컬렉션이다.
     + 집합은 효율적인 포함 여부 검사에 최적화되어 있다. 자바에는 HashSet과 TreeSet 구현이 있다.
     + 맵은 HashMap과 TreeMap 구현 중에서 선택할 수 있다. LinkedHashMap은 삽입 순서를 유지한다.
     + Collection 인터페이스와 Collections 클래스에는 집합 연산, 검색, 정렬 뒤섞기(셔플링)처럼 다양하고 유용한 알고리즘이 있다.
     + 뷰는 표준 컬렉션 인터페이스를 이용해 어딘가 저장된 데이터에 접근할 수 있게 한다.

1. 8장 스트림
   1. 반복에서 스트림 연산으로 전환
      * 책에 나오는 긴 단어를 모두 센다.

|  |
| --- |
| String contents = new String(Files.readAllBytes(  Pahths.get(“alice.txt”)), StandardCharsets.UTF\_8); // 파일을 문자열로 읽어온다.  List<String> words = List.of(contents.split([\\PL+)](file:///\\PL+)));  // 단어로 분리한다(비문자(글자가 아닌 문자)를 구분자로 사용한다.  // 스트림 연산  long count = words.parallelStream()  .filter(w -> w.leangh() > 12)  .count(); |

* + - 컬렉션 vs 스트림  
      스트림은 요소를 저장하지 않는다.   
      스트림 연산은 원본을 변경하지 않는다.  
      스트림 연산은 가능하면 지연 방식으로 작동한다.
  1. 스트림 생성
     + 배열은 정적 메서드 Stream.of를 사용하여 스트림으로 변환해야 한다.

|  |
| --- |
| Stream<String> words = Stream.of(contents.split([\\PL+)](file:///\\PL+)));  // split은 String[] 배열을 반환한다. |

* + - 배열의 일부에서 스트림을 만들려면 Arrays.stream(array, from, to)를 사용해야 한다.
    - 요소가 없는 스트림을 만들려면 정적 메서드 Stream.empty를 사용한다.

|  |
| --- |
| Stream(String> silence = Stream.enpyu();  // 제네릭 타입 <String>이 추론되므로 Stream.<String>empty()와 같다. |

* + - generate메서드 : 인수 없는 함수를 받는다, 난수, 수열

|  |
| --- |
| // 상수 값의 스트림  Stream<String> echos = Stream.generate(() -> “Echo”);  // 난수의 스트림  Stream<Double> randoms = Stream.generate(Math::random);  // 수열  BigInteger limit = new BigInteger(“10000000”);  Stream<BigInteger> integers  = Stream.interate(BigInteger.ZERO,  n -> n.compareTo(limit) <0,  n -> n.add(BigInteger.ONE)); |

* 1. filter, map, flatMap 메서드
     + filter : 특정 조건과 일치하는 요소로 구성된 새 스트림을 돌려줌.
     + map : 스트림에 들어있는 값을 특정 방식으로 변환
     + flatMap : 단일 스트림으로 펼칠 때 사용
  2. 서브스트림 추출과 스트림 결합
     + stream.limit(n) 호출 : 요소 n개 이후 끝나는 새 스트림을 반환  
       Stream<Double> randoms = Stream.generate(Math::random).limit(100);
     + stream.skip(n) 호출 : limit와 반대로 처음 n개 요소를 버린다.  
       Stream<String> words = Stream.of(contents.split([\\PL+](file:///\\PL+))).skip(1);
     + stream.takeWhile(predicate) 호출 : predicate가 참인 동안 스트림에서 모든 요소를 가져온 후 중단  
       Stream<String> initialDigits = codePoints(str).takeWhile(  
        s -> “0123456789”.contains\*(s));
     + stream.dropWhile 메서드 : 조건이 참인 동안 요소를 버리고 조건이 처음으로 거짓으로 판명된 요수부터 시작해서 모든 요소의 스트림을 돌려준다.  
       Stream<String> wiuthoutInitialWhiteSpace = codePoints(str).dropWhile(  
        s -> s.trim().leangh() == 0);
     + concat : 두 스트림을 연결  
       Stream<String> combined = Stream.concat(  
        codePoints(“Hello”), codePoints(“World));
  3. 기타 스트림 변환
     + distinct : 원본스트림에 있는 요소의 중복을 제외하고 같은 순서로 돌려주는 스트림을 반환
     + sorted : 스트림 정렬요으로 사용
     + peek : 원본과 요소가 같은 또 다른 스트림을 돌려줌
  4. 단순리덕션
     + 리덕션은 종료 연산 이다.

|  |
| --- |
| //스트림의 최대값  Optional<String> largest = words.max(String||compareToIgnoreCase);  System.out.println(“largest: “ + largest.orElse(“”));  //비어있지 않은 컬렉션의 첫번쨰 값 반환  Optional<String> startWithQ  = words.filter(s -> s.startWith(“Q”)).findFirst();  // findAny  Optional<String> startWhthQ  = words.parallel().filter(s -> startsWith(“Q”)).findAny();  // 일치하는 요소확인(anyMatch)  boolean aWordStartsWithQ  = worlds.parallel().anyMatch(s -> s.startWith(“Q”)); |

* 1. 옵션 타입
     + Optional<T> 객체는 T 타입 객체 또는 객체가 없음을 나타내는 래퍼다.

|  |
| --- |
| // 옵션 값을 사용하는 방법  // 값이 없을 때 대체 값을 만들어 내는 방법  String result = optinalString.orElse(“”);  // 옵션 값으로 래핑된 문자열 또는 문자열이 없으면 “”  String result = optionalString.orElseGet(() -> System.getProperty(“myapp.default”));  // 필요할 때만 함수가 호출된다.  String result = optionalString.orElseThrow(IllegalStateException::new);  // 예외 객체를 돌려주는 메서드를 전달한다.  // 값이 있을 때만 소비하는 방법  optionalValue.ifPresent(v -> result.add(v));  /ifPresent 옵션값이 있을 때 해당 함수로 값이 전달된다.  optionalValue.ifPresentOrElse(  v -> v*를 처리한다*,  () -> *다른 일을 수행한다*);  // ifPresentOrElse 값이 있을 떄는 어떤 액션을 수행하고, 값이 없을때는 다른 액션을 수행  // 옵션 값 생성  public static Optional<Double> inverse(Double x) {  return x == 0 ? Optional.empty() : Optional.of(1 / x);  } |

* 1. 결과 모으기
     + forEach 메서드를 호출해 각 요소에 함수를 적용  
       stream.forEach(System.out::porintln);
     + 리스트로 모으기  
       List<String> result = stream.collect(Collectors.toList());
     + 집합으로 모으기  
       Set<String> result = stream.collect(Collectors.toSet());
     + 얻을 집합의 종류를 제어  
       TreeSet<String> result = stream.collect(Collectors.toCollection(TreeSet:new));
     + 스트림에 있는 모든 문자열을 연결해서 모으기  
       String result = stream.collect(Collectors.joining());
     + 요소 간에 구분자가 필요할 때   
       String result = stream.collect(Collectors.joining(“, “));
     + 스트림에 문자열 외의 객체가 있다면 해당 객체를 문자열로 변환해야 한다.  
       String result = stream.map(Object::toString).collect(Collectors.joining(“, “));
     + 합계, 카운트, 평균, 최댓값, 최소값으로 리듀스 summarizaing(Int|Long|Double)

|  |
| --- |
| IntSummaryStatistics summary = stream.collect(  Collectors.summarizingInt(String::leangh));  double averageWorldLeangh = summary.getAverage();  double maxWordLeangh = summary.getMax(); |

* 1. 맵으로 모으기

|  |
| --- |
| Map<Integer, String> idToName = people.collect(  Collecctors.toMap(Person::getId, Person::getName));  Map<Integer, String> idToPerson = people.collect(  Collecctors.toMap(Person::getId, Person::identity()));  Stream<Locale> locales = Stream.of(Locale.getAvcailableLacales());  Map<String, String> languageNames = locales.collect(  Collectors.toMap(  Locale::getDisplayLanguage,  loc -> loc.getDisplayLanguage(loc),  (existingValue, newValue) -> existingValue)); |

* 1. 그루핑과 파티셔닝

|  |
| --- |
| Map<Stgring, List<Locale>> contryToLocales = locales.collect(  Collectors.groupingBy(Locale::getContry));  List<locale> swissLocales = contryToLocales.get(“CH”);  // 로케일 [it\_CH, de\_CH, fr\_CH]를 돌려준다.  // 분류 함수가 프리디케이트 함수면 스트림요소가 두개로 나뉘며, 이 경우 partitioningBy를 사용하면 훨신 효율적이다.  Map<Boolean, List<Local>> enghlishAndOtherLocales = locales.collect(  Collectors.partitiongBy( l -> l.getLanguage().equals(“en”)));  List<Locale> englishLocales = englishAndOtherLocales.get(true);  // 영어와 그외 로케일로 분리한다. |

* 1. 다운스트림 컬렉터

|  |
| --- |
| // groupingBy메서드는 값을 리스트로 맵을 돌려주며, 리스트를 원하는 방식으로 처리하려면 다운 스트림 컬렉터를 작성하여아야 함.  Map<String, Set<Locale>> countryToLocaleSet = locales.collect(  groupingBy(Locale::getCountry, toSet()));  // 리스트 대신 집합이 필요할 때 Collectors.toSet 컬렉터를 사용  // Counting : 모든 요소의 개수를 센다  Map<String, Long> countryToLocaleCounts = locales.collect(  groupingBy(Locale::getCountry, counting()));  // summing : 함수의 인수를 받아서 해당 함수를 다운스트림 요소에 적용하고 합계를 구함  Map<String, Integer> stateToCityPopulation = cities.collect(  groupingBy(City::getState, summingInt(City::getPopulation)));  // 최대값과 최소값을 구한다  Map<String, Optional<City>> stateToLargestCity = cities.collect(  groupingBy(City::getState,  maxBy(Comparator.comparing(City::getPopulation)))); |

* 1. 리덕션 연산

|  |
| --- |
| // reduce 메서드  List<Integer> values = ...;  Optional<Integer> sum = values.stream().reduce((x, y) -> x + y);  // 스트림이 비어있으면 유효한 결과가 없는 것이므로 Optional을 반환한다.  List<Integer> values = ...;  Integer sum = values.stream().reduce(0, (x, y) -> x + y)  // 0 + v0 + v1 + v2 + ...을 계산한다.  // 스트림이 비어있으면 항등 값을 반환하므로 더는 Optional 클래스를 사용할 필요가 없다. |

* 1. 기본 타입 스트림
     + short, char, byte, boolean 타입을 저장할 때는 IntStream을 사용하고, float 타입을   
       저장할 때는 DoubleStream을 사용한다.
     + IntStream을 생성하려면 다음과 같이 IntStream.of와 Arrays.stream 메서드를 호출한다.

|  |
| --- |
| IntStream stream = IntStream.of(1, 1, 2, 3, 5); stream = Arrays.stream(values, from, to); // values는 int[ ] 배열이다. |

* + - 객체 스트림이 있다면 mapToInt, mapToLong, mapToDouble 메서드를 사용해 기본 타입 스트림으로 변환할 수 있다.

|  |
| --- |
| Stream<String> words = ...; IntStream lengths = words.mapToInt(String::length);  // 문자열 스트림에 있는 문자열 요소의 길이를 정수로 처리하고 싶을 때 |

* 기본 타입 스트림을 객체 스트림으로 변환하려면 boxed 메서드를 사용

|  |
| --- |
| Stream<Integer> integers = IntStream.range(0, 100).boxed(); |

* 1. 병렬 스트림
     + 스트림을 이용하면 벌크 연산(bulk operation)을 쉽게 병렬화할 수 있다. 처리 과정은 대부분 자동이지만, 몇 가지 규칙을 따라야 한다. 먼저 병렬 스트림(parallel stream)이 있어야 한다. 어떤 컬렉션에서든 Collection.parallelStream() 메서드로 병렬 스트림을 얻을 수 있다.

|  |
| --- |
| Stream<String> parallelWords = words.parallelStream();  Stream<String> parallelWords = Stream.of(wordArray).parallel();  // parallel 메서드는 어떤 순차 스트림이든 병렬 스트림으로 변환  Map<Integer, Long> shortWordCounts  = words.parallelStream()  .filter(s -> s.length() < 12)  .collect(groupingBy(  String::length,  counting()));  // 문자열을 길이로 묶어서 세면 안전하게 병렬화 가능 |

* 1. 핵심 내용 정리
     + 반복자는 특정 순화 전략을 내포하므로 효율적인 동시 실행을 방해한다.
     + 컬렉션, 배열, 발생기, 반복자에서 스트림을 생성할 수 있다.
     + filter요소를 선택하고, map으로 요소를 변환한다.
     + 다른 스트림 변환 연산으로 limit, distinct, sorted가 있다.
     + 스트림에서 결과를 얻으려면 count, max, min, findFirst, findAny 등 리덕션 연산자를 사용해야 한다. 이 메서드 중 일부는 Optional 값을 반환한다.
     + Optional 타입을 만든 목적은 null 값을 안전하게 다루는 것이다. null을 안전하게 사용하려면 ifPresent와 orElse 메서드를 써야 한다.
     + 스트림 결과를 컬렉션, 배열, 문자열, 맵으로 모을 수 있다.
     + Collectors 클래스와 groupingBy와 partitioningBy 메서드로 스트림의 콘텐츠를 그룹으로 나누고, 각 그룹의 결과를 얻을 수 있다.
     + 기본 타입 int, long, double용으로 특화된 스트림이 있다.
     + 병렬 스트림은 스트림 연산을 자동으로 병렬화한다.