컬렉션 API 개선 8장

By 박찬
역

8.1 컬렉션 팩토리

자바 9에서는 작은 컬렉션 객체를 쉽게 만들 수 있는 몇 가지 방법을 제공한다.

기존 방법

```
List<String> friends = new ArrayList<>();
friends.add("Raphael");
friends.add("Olivia");
friends.add("Thibaut");
```

asList() 메서드

```
List<String> friendsList = Arrays.asList("Raphael", "Olivia", "Thibaut");
```

- 고정 크기의 리스트를 만든다.
- 요소에 대해 갱신은 가능하지만 추가를 하면 Unsupported OperationException이 발생한다.

By 박찬

8.1 컬렉션 팩토리

```
friendsList.set(0, "Richard");
System.out.println("asList: " + friendsList);

friendsList.add("Thibaut");
System.out.println(friendsList);
```

추가가 가능하게 하고 싶다면?

```
Set<String> friendsSet = new HashSet<>(Arrays.asList("Raphael", "Olivia", "Thibaut"));
friendsSet.add("test1");
```

```
Set<String> friendsSetSteam = Stream.of("Raphael", "Olivia", "Thibaut")
    .collect(Collectors.toSet());
friendsSetSteam.add("test");
```

8.1.1 리스트 팩토리

List.of 메서드 활용

```
List<String> friendsListOf = List.of("Raphael", "Olivia", "Thibaut");
System.out.println("ListOf: " + friendsListOf);

friendsListOf.set(0, "Chih-Chun");
friendsListOf.add("Chig-Chun")
```

- 고정 크기의 리스트를 만든다.
- 요소에 대한 갱신 및 추가 전부 불가능 하다.

⇒ 의도치 않은 변화를 막을 수 있다.

⇒ 데이터 처리 형식을 설정하거나 데이터를 변환할 필요가 없을 경우 이용

8.1.2 집합 팩토리

Set.of 메서드 활용

```
Set<String> friendsSetOf = Set.of("Raphael", "Olivia", "Thibaut");
System.out.println("friendsSetOf: " + friendsSetOf);

Set<String> friendsSe = Set.of("Raphael", "Olivia", "Olivia");
//오직 고유의 요소만 포함할 수 있기 때문에 에러 발생
```

- 변경할 수 없는 집합을 만든다.
- 집합이기 때문에 중복을 허용하지 않는다.

8.1.3 맵 팩토리

두 가지 방법으로 바꿀 수 없는 맵을 초기화 할 수 있다.

Map.of 메서드 활용

```
Map<String, Integer> ageOfFriends = Map.of("Raphael", 30, "Olivia", 25);
//열 개 이하의 키와 값 쌍을 가진 작은 맵을 만들때 유용
```

Map.ofEntries 메서드 활용

```
Map<String, Integer> ageOfFriendsMapOfEntries = Map.ofEntries(
    entry("Rapha", 30),
    entry("Olivia", 25),
    entry("Thibaut", 26)
);
```

- 10개 이상의 맵을 만들 때 유용하다.
- 이 메서드는 키와 값을 감쌀 추가 객체 할당을 필요로 한다.

8.1.3 맵 팩토리

퀴즈 8-1

다음 코드를 실행한 결과는?

```
List<String> actors = List.of("Keanu", "Jessica");
actors.set(0, "Brad");
System.out.println(actors);
```

정답 : UnsupportedOperationException 발생

8.2 리스트와 집합 처리

- removeIf: 프레디케이트를 만족하는 요소를 제거한다. List나 Set을 구현하거나 그 구현을 상속받은 모든 클래스에서 이용할 수 있다.
- replaceAll: 리스트에서 이용할 수 있는 기능으로 UnaryOperator 함수를 이용해 요소를 바꾼다.
- Sort: List 인터페이스에서 제공하는 기능으로 리스트를 정렬한다.

⇒ 호출한 컬렉션 자체를 바꾼다.

⇒ 새로운 결과를 만드는 스트림 동작과 달리 이들 메서드는 기존 컬렉션을 바꾼다.

8.2.1 removeIf 메서드

기존 방법

> ConcurrentModificationException

```
Iterator<Integer> iterator = integers.iterator();
while (iterator.hasNext()){
    int number = iterator.next();
    if (number == 1) {
        integers.remove(number);
        //참조 객체에서 값을 제거하기 때문에 동기화되지 않는다.
        //즉 integers에서 값이 빠지게 되면 그 객체를 참조하는 반복자인 iterator와 동기화 되지 않는다
    }
}
```

8.2.1 removeIf 메서드

기존 방법

```
Iterator<Integer> iterator = integers.iterator();
while (iterator.hasNext()){
    int number = iterator.next();
    if (number == 1) {
        iterator.remove(); //이터레이터 객체에서 값을 제거한다.
    }
} //코드가 복잡해진다는 단점이 있다.
```

removeIf

```
integers.removeIf(number -> number == 1);
```

8.2.2 replaceAll 메서드

replaceAll

```
 integers.stream()
    .map(number -> number + 1)
    .collect(Collectors.toList())
    .forEach(System.out::println); //새로운 컬렉션을 만든다는 단점이 있다.
// 우리는 기존의 컬렉션을 바꾸기를 원한다.
```

```
ListIterator<Integer> integerIterator = integers.listIterator();
while (integerIterator.hasNext()) {
    int number = integerIterator.next();
    integerIterator.set(number + 2);
}
// 코드가 굉장히 복잡해 지며
```

```
integers.replaceAll(number -> number + 2);
```

By 박찬

Map 인터페이스에 추가된 디폴트 메서드

8.3.1 forEach 메서드

이전 코드

```
Set<Map.Entry<String, Integer>> entrySet = ageOfFriends.entrySet();
Iterator<Map.Entry<String, Integer>> entryIterator = entrySet.iterator();

while (entryIterator.hasNext()) {
    Map.Entry<String, Integer> entry = entryIterator.next();
    String key = entry.getKey();
    Integer value = entry.getValue();
    System.out.println(key + " is " + value + " years old");
}
```

forEach 적용

```
ageOfFriends.forEach((friend, age) ->
                    System.out.println(friend + " is " + age + " years old"));
```

8.3.2 정렬 메서드

Entry.comparingByKey

맵 생성

```java
Map<String, String> favouriteMovies = Map.ofEntries(entry("Raphael", "Start Wars"),
                                                    entry("Cristina", "Matrix"),
                                                    entry("Olivia", "James Bond"));
```

Entry.comparingByKey

```java
favouriteMovies
    .entrySet()
    .stream()
    .sorted(Map.Entry.comparingByKey())
    .forEachOrdered(System.out::println); //알파벳 순으로 요소를 처리
```

결과

```
Cristina=Matrix
Olivia=James Bond
Raphael=Start Wars
```

8.3.2 정렬 메서드

Entry.comparingByValue

> Entry.compoaringByValue

```
favouriteMovies
    .entrySet()
    .stream()
    .sorted(Map.Entry.comparingByValue())
    .forEachOrdered(System.out::println);
```

```
Olivia=James Bond
Cristina=Matrix
Raphael=Start Wars
```

By 박찬

8.3.3 getOrDefault 메서드

- NullPointerException 방지(키가 존재하지 않으면 기본값 반환)
- 키가 존재하나 값이 존재하지 않아도 기본값 반환

맵 생성

```
Map<String, String> favouriteMovies = Map.ofEntries(entry("Raphael", "Start Wars"),
                                                    entry("Cristina", "Matrix"),
                                                    entry("Olivia", "James Bond"));
```

getOrDefault

```
System.out.println(favouriteMovies.getOrDefault("Olivia", "Matrix"));
System.out.println(favouriteMovies.getOrDefault("Thibaut", "Matrix"));
```

결과

```
James Bond
Matrix
```

8.3.4 계산 패턴

•computeIfAbsent: 제공된 키에 해당하는 값이 없으면(값이 없거나 널), 키를 이용해 새 값을 계산하고 맵에 추가한다.

```
Map<String, List<String>> friendsToMovies = new HashMap<>();
List<String> movieList = new ArrayList<>();
movieList.add("movie1");
movieList.add("movie2");
movieList.add("movie3");
friendsToMovies.put("Raphael", movieList);

String friend = "Raphael";
List<String> movies = friendsToMovies.get(friend);
if (movies == null) {
    movies = new ArrayList<>();
    friendsToMovies.put(friend, movies);
}
```

computeIfAbsent 적용

```
friendsToMovies.computeIfAbsent("Raphael", name -> new ArrayList<>())
    .add("Star Wars");
```

By 박찬

8.3.5 삭제 패턴

키가 특정한 값과 연관되었을 때만 항목을 제거하는 메서드 구현

```
if (friendsToMovies.containsKey(key) &&
    Objects.equals(friendsToMovies.get(key), movieList)) {
    friendsToMovies.remove(key);
}
```

remove(key, value) 메서드

```
friendsToMovies.remove(key, movieList);
```

- Key와 value를 가지고 있으면 key에 들어있는 요소들 제거

8.3.6 교체 패턴

replaceAll: BiFunction을 적용한 결과로 각 항목의 값을 교체.

List의 replaceAll과 비슷한 동작을 수행한다.

```
Map<String, String> favouriteMovie = new HashMap<>();
favouriteMovie.put("Raphael", "Star Wars");
favouriteMovie.put("Olivia", "james bond");
favouriteMovie.replaceAll((name, movie) -> movie.toUpperCase());
```

> 결과

```
{Olivia=JAMES BOND, Raphael=STAR WARS}
```

8.3.7 합침

putAll() 메서드

> putAll (키 중복시 덮어쓴다.)

```java
Map<String, String> family = Map.ofEntries(
    entry("Teo", "Start Wars"), entry("Cristina", "James Bond"));
Map<String, String> friends = Map.ofEntries(
    entry("Raphael", "Star Wars"), entry("Cristina", "Matrix"));

Map<String, String> everyone = new HashMap<>(family);
everyone.putAll(friends);
```

> 결과

```
{Cristina=Matrix, Raphael=Star Wars, Teo=Start Wars}
```

- 키가 중복될 경우 나중에 들어온 값으로 덮어 쓴다.

8.3.7 합침

Merge 메서드

```
friends.forEach((k, v) ->
                everyone.merge(k, v, (movie1, movie2) -> movie1 + " & " + movie2));
System.out.println(everyone);
```

> 결과

```
{Raphael=Star Wars, Cristina=James Bond & Matrix, Teo=Start Wars}
```

- Merge는 중복된 키를 어떻게 합칠지 결정하는 BiFunction을 인수로 받는다.
- 중복이 될 경우 두 영화의 문자열을 합치는 방법으로 문제를 해결

8.3.7 합침

퀴즈 8-2

다음 코드가 어떤 작업을 수행하는지 파악한 다음 코드를 단순화할 수 있는 방법을 설명하시오.

```
Map<String, Integer> movies = new HashMap<>();
movies.put("JamesBond", 20);
movies.put("Matrix", 15);
movies.put("Harry Potter", 5);
Iterator<Map.Entry<String, Integer>> iterator =
    movies.entrySet().iterator();
```

> 정답

```
movies.entrySet().removeIf(entry -> entry.getValue() < 10);
```

By 박찬

8.4 개선된 ConcurrentHashMap

- 내부 자료구조의 특정 부분만 잠굼으로 동시 추가, 갱신 작업을 허용한다.
- 동기화된 Hashtable 버전에 비해 읽기 쓰기 연산 성능이 월등하다.

reduceValues()메서드를 활용한 리듀스 연산

```
ConcurrentHashMap<String, Long> map = new ConcurrentHashMap<>();
map.put("name1", 10L);
map.put("name2", 30L);
map.put("name3", 20L);
long parallelismThreshold = 1; //공통 스레드 풀을 이용해 병렬성을 극대화
Optional<Long> maxValue =
    Optional.ofNullable(map.reduceValues(parallelismThreshold, Long::max));
```

> 결과

```
Optional[30]
```

By 박찬

8.4.2 계수

- 맵의 개수를 반환하는 mappingCount 메서드를 제공한다.

> mappingCount

```
long counts = map.mappingCount();
System.out.println(counts);
```

> 결과

```
3
```