LAPORAN PRAKTIKUM TEKNIK PEMROGRAMAN: *JAVA COLLECTION FRAMEWORK*

Disusun Oleh :

Muhammad Arrafi Reva Razqana Arassy

NIM

241524017

POLITEKNIK NEGERI BANDUNG
PROGRAM STUDI D4 TEKNIK INFORMATIKA
2025

DAFTAR ISI

List, Set, Map

Dalam pemrograman, kita sering kali harus menyimpan banyak data dalam satu tempat, misalnya, kita ingin menyimpan daftar nama mahasiswa, daftar nomor telepon seseorang, atau daftar harga barang di supermarket. Untuk melakukan ini, kita menggunakan struktur data yang disebut Collection di Java. Ada tiga jenis utama dari koleksi ini adalah List, Set, dan Map

List

List adalah struktur data yang menyimpan elemen secara berurutan, elemen bisa juga berisi elemen yang sama (duplikat). Kita bisa membayangkan List seperti daftar tugas atau daftar belanja, di mana setiap elemen memiliki urutan tertentu. Contoh implementasi List di Java adalah ArrayList dan LinkedList.

Alasan Mengapa List Diperlukan

List cocok untuk menyimpan data dengan urutan tertentu dan mungkin mangandung duplikat seperti antrian tugas atau riwayat pemesanan makanan.

Set

Set adalah struktur data yang hanya menyimpan elemen unik, artinya tidak boleh ada elemen yang sama. Urutan elemen dalam Set juga biasanya tidak tetap atau tidak terjamin. Kita bisa membayangkan Set seperti daftar nomor KTP atau nomor registrasi mahasiswa, di mana setiap orang memiliki nomor yang unik. Contoh implementasi Set di Java adalah HashSet, LinkedHashSet, dan TreeSet.

Alasan Mengapa Set Diperlukan

Set dipakai jika kita ingin menyimpan data yang unik, seperti daftar username yang sudah terdaftar.

Map

Map adalah struktur data yang menyimpan data dalam bentuk pasangan kunci (key) dan nilai (value). Kunci (key) harus unik, tetapi nilai (value) bisa sama. Kita bisa membayangkan Map seperti daftar kontak di ponsel kita, di mana nama orang adalah key, dan nomor telepon adalah value. Contoh implementasi Map di Java adalah HashMap, LinkedHashMap, dan TreeMap.

Alasan Mengapa Map Diperlukan

Map diperlukan jika kita ingin mencari sesuatu berdasarkan "kata kunci", misalnya menyimpan username dan password di sistem login, ini memungkinkan pencarian yang lebih cepat.

Record

Saat kita membuat sebuah class di Java, kita biasanya harus menulis banyak kode untuk getter, setter, constructor, dan sebagainya. Hal ini sering membuat kode menjadi panjang dan sulit dibaca. Record adalah fitur baru di Java yang membantu kita membuat class dengan lebih ringkas dan otomatis. Record digunakan untuk menyimpan data yang tidak berubah (immutable). Dengan Record, kita tidak perlu menulis getter, setter, constructor, karena semuanya dibuat otomatis.

Alasan Mengapa Record Diperlukan

Record Berguna untuk membuat kelas yang hanya berisi data tanpa logika tambahan, sehingga mengurangi boilerplate code, Mengurangi jumlah kode yang harus ditulis, dan Data yang dibuat dengan record bersifat immutable (tidak bisa diubah setelah dibuat).

Optional

Di Java, jika kita mencari sesuatu yang tidak ada dalam data, biasanya hasilnya adalah null. Jika kita mencoba menggunakan nilai null, program bisa mengalami error yang disebut NullPointerException. Untuk mencegah error ini, kita bisa menggunakan Optional. Optional adalah cara yang lebih aman untuk menangani nilai yang mungkin kosong (null). Dengan Optional, kita bisa menentukan apa yang harus dilakukan jika nilai ada atau tidak ada.

Alasan Mengapa Optional Diperlukan

Optional Berguna untuk menghindari NullPointerException dan membuat kode lebih aman dan mudah dibaca dan Memaksa kita untuk menangani kemungkinan nilai kosong secara eksplisit.

Concurrent Collections

Biasanya, dalam program yang berjalan dengan banyak proses secara bersamaan (multithreading), ada risiko data berubah di tengah-tengah proses. Jika dua atau lebih proses mengakses dan mengubah data pada saat yang sama, bisa terjadi race condition atau data corruption.Concurrent Collections adalah koleksi data yang dirancang untuk digunakan dalam multithreading agar data tetap aman dan konsisten meskipun diakses oleh banyak proses.

Alasan Mengapa Concurrent Collections Diperlukan

Concurrent Collenction Berguna untuk memastikan keamanan thread (thread safety) dan menghindari race conditions, Untuk mencegah error saat beberapa proses mencoba mengubah data pada saat yang bersamaan, dan Untuk meningkatkan performa aplikasi dengan membiarkan beberapa proses berjalan secara bersamaan tanpa konflik.

Queue dan Dequeue

Dalam kehidupan sehari-hari, kita sering menemui konsep antrian (Queue). Misalnya, di kasir supermarket atau saat mengantri tiket bioskop. Queue adalah struktur data yang mengikuti prinsip FIFO (First In First Out), di mana elemen pertama yang masuk akan diproses lebih dulu. Sedangkan Deque (Double-Ended Queue) adalah variasi dari Queue yang memungkinkan kita menambahkan dan menghapus elemen baik dari depan maupun belakang.

Alasan Mengapa Queue dan Dequeue Diperlukan

Queue: Berguna untuk mengelola tugas-tugas yang perlu diproses dalam urutan tertentu.

Deque: berguna untuk kasus seperti browser history, di mana kita bisa menelusuri halaman sebelumnya dan berikutnya.

Immutable collection List.of, Set.of, Map.of

Biasanya, saat kita membuat List, Set, atau Map, kita bisa menambahkan atau mengubah elemen kapan saja. Tetapi, dalam beberapa kasus, kita ingin membuat koleksi yang tidak bisa diubah setelah dibuat. Immutable Collection adalah koleksi yang tidak bisa dimodifikasi. Java menyediakan cara mudah untuk membuat koleksi immutable menggunakan List.of(), Set.of(), dan Map.of().

Alasan Mengapa Immutable collection List.of, Set.of, Map.of Diperlukan

1. Untuk mencegah perubahan data yang tidak disengaja dalam program.
2. Untuk meningkatkan keamanan dan stabilitas aplikasi.
3. Untuk meningkatkan efisiensi memori karena koleksi immutable tidak perlu diubah.

Kasus Penerapan Dunia Nyata

Saya membuat kasus yaitu Sistem Manajemen Penyelenggaraan Festival Seni Budaya di Dunia Virtual yang memuat ke-6 konsep dari List, Set, Map, Record, Optional, Concurrent Collections, Queue dan Dequeue, dan Immutable Collection. Saya akan memberikan potongan Source code dari program saya terkait dengan ke-6 konsep List, Set, Map, Record, Optional, Concurrent Collections, Queue dan Dequeue, dan Immutable Collection serta alasan penggunaannya.

List

```
// List: Menyimpan daftar seniman yang berpartisipasi.
private final List<Artist> artists = new ArrayList<>();
```

```
public void addArtist(Artist artist) {
    artists.add(artist); // Tambahkan seniman ke List
    categories.add(artist.kategori()); // Tambahkan kategori ke Set
}
```

Alasan Penggunaan

Kasus: Dalam festival seni digital, banyak seniman yang akan mendaftar dan berpartisipasi.

Kebutuhan: Kita perlu menyimpan daftar seniman secara berurutan dan memungkinkan adanya duplikat (jika ada seniman dengan nama yang sama).

Solusi: List (khususnya ArrayList) cocok untuk menyimpan data berurutan dan memungkinkan penambahan, penghapusan, dan pengaksesan data dengan cepat.

Set

```
// Set: Menyimpan daftar kategori seni unik.
private final Set<String> categories = ConcurrentHashMap.newKeySet();
```

```
public void addArtist(Artist artist) {
    artists.add(artist); // Tambahkan seniman ke List
    categories.add(artist.kategori()); // Tambahkan kategori ke Set
}
```

Alasan Penggunaan

Kasus: Setiap seniman memiliki kategori seni yang berbeda, dan kita perlu memastikan tidak ada duplikat dalam daftar kategori.

Kebutuhan: Kita perlu menyimpan data unik tanpa duplikat.

Solusi: Set (khususnya ConcurrentHashMap.newKeySet()) cocok untuk menyimpan data unik dan memastikan tidak ada duplikat.

Map

```
// Map: Menyimpan informasi karya seni yang dipamerkan.
private final ConcurrentHashMap<String, Artwork> artworks = new ConcurrentHashMap<>();
```

```
public Optional<Artwork> addArtwork(Artwork artwork) {
    return Optional.ofNullable(artworks.putIfAbsent(artwork.id(), artwork));
}
```

Alasan Penggunaan

Konsep: Map digunakan untuk menyimpan informasi karya seni yang dipamerkan, dengan ID karya sebagai kunci dan deskripsi karya sebagai nilai.

Kasus: Setiap karya seni memiliki ID unik, dan kita perlu menyimpan informasi karya seni tersebut.

Kebutuhan: Kita perlu menyimpan data dengan kunci unik (ID karya) dan nilai (deskripsi karya).

Solusi: Map (khususnya ConcurrentHashMap) cocok untuk menyimpan data dengan kunci unik dan memastikan keamanan thread.

Record

```
public record Artist(String id, String nama, String kategori) {
}
```

```
public record Artwork(String id, String deskripsi) {
}
```

Alasan Penggunaan

Konsep: Record digunakan untuk membuat kelas sederhana dan immutable yang menyimpan data seniman dan karya seni.

Kasus: Kita perlu menyimpan data seniman (ID, nama, kategori) dan karya seni (ID, deskripsi) secara efisien.

Kebutuhan: Kita perlu membuat kelas yang hanya menyimpan data tanpa logika tambahan.

Solusi: Record cocok untuk membuat kelas sederhana dan immutable yang hanya menyimpan data.

Optional

```
public Optional<Artwork> addArtwork(Artwork artwork) {
    return Optional.ofNullable(artworks.putIfAbsent(artwork.id(), artwork));
}
```

Alasan Penggunaan

Konsep: Optional digunakan untuk menangani kasus di mana karya seni mungkin tidak ditemukan atau sudah ada.

Kasus: Saat menambahkan karya seni, kita perlu memastikan bahwa karya seni tersebut belum ada dalam daftar.

Kebutuhan: Kita perlu menghindari NullPointerException dan membuat kode lebih aman.

Solusi: Optional cocok untuk menangani kasus di mana nilai mungkin tidak ada (null).

Concurrent Collections

```
// Map: Menyimpan informasi karya seni yang dipamerkan.
private final ConcurrentHashMap<String, Artwork> artworks = new ConcurrentHashMap<>();
```

```
// Deque: ConcurrentLinkedDeque digunakan untuk mengantre partisipasi secara thread-safe.
private final Deque<String> queue = new ConcurrentLinkedDeque<>();
```

Alasan Penggunaan

Konsep: Concurrent Collections digunakan untuk memastikan keamanan thread dalam lingkungan multi-thread.

Kasus: Dalam festival seni digital, banyak seniman yang mendaftar dan memamerkan karya secara bersamaan.

Kebutuhan: Kita perlu memastikan bahwa operasi pada koleksi aman dari race condition.

Solusi: ConcurrentHashMap dan ConcurrentLinkedDeque cocok untuk operasi yang aman dalam lingkungan multi-thread.

Queue dan Dequeue

```
private final Deque<String> queue = new ConcurrentLinkedDeque<>();

/**
 * Method untuk menambahkan partisipasi seniman ke antrian.
 * @param artistId ID seniman yang akan berpartisipasi.
 */
public void addParticipation(String artistId) {
    queue.addLast(artistId); // Tambahkan seniman ke belakang antrian
}
```

```
/**
 * Method untuk memproses antrian partisipasi seniman.
 */
public void processParticipations() {
    ExecutorService executor = Executors.newFixedThreadPool(nThreads:2);
    while (!participationQueue.isEmpty()) {
        String artistId = participationQueue.pollParticipation(); // Ambil dari ParticipationQueue
        executor.submit(() -> {
            System.out.println("Memproses partisipasi seniman: " + artistId);
            // Simulasikan proses partisipasi
            try {
                Thread.sleep(millis:1000); // Simulasi waktu proses
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("Partisipasi selesai: " + artistId);
        });
    }
    executor.shutdown();
}
```

Alasan Penggunaan

Konsep: Queue dan Dequeue digunakan untuk mengatur antrian partisipasi seniman.

Kasus: Seniman mendaftar secara berurutan, dan kita perlu memproses pendaftaran mereka sesuai urutan.

Kebutuhan: Kita perlu mengelola antrian tugas dengan prinsip FIFO (First In First Out).

Solusi: Deque (khususnya ConcurrentLinkedDeque) cocok untuk mengelola antrian tugas dengan prinsip FIFO.

Immutable Collection

```java
/**
 * Record untuk menyimpan data seniman.
 * Menggunakan konsep **Record** untuk membuat kelas sederhana dan immutable.
 * @param id        ID unik seniman.
 * @param nama      Nama seniman.
 * @param kategori  Kategori seni yang ditekuni.
 */
public record Artist(String id, String nama, String kategori) {
}
```

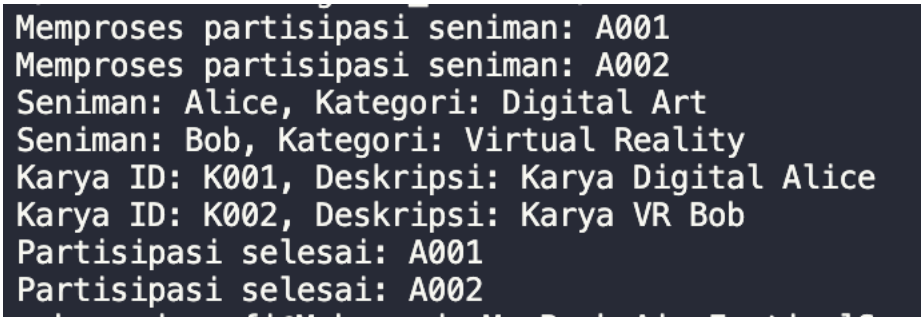
Konsep: Immutable Collection digunakan untuk menyimpan daftar data yang tidak boleh diubah setelah dibuat.

Kasus: Kategori seni yang ditampilkan dalam festival tidak boleh diubah setelah festival dimulai.

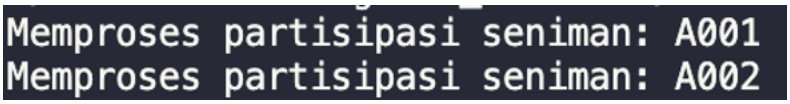
Kebutuhan: Kita perlu memastikan data tidak berubah setelah dibuat.

Solusi: Immutable Collection (seperti List.of()) cocok untuk menyimpan data yang tidak boleh diubah.

Hasil Output Program



Penjelasan Singkat Output



Apa yang terjadi?

Program memproses partisipasi seniman dengan ID A001 dan A002 secara berurutan.

Mengapa terjadi?

Seniman A001 dan A002 telah ditambahkan ke antrian partisipasi menggunakan method addParticipation().

Method processParticipations() mengambil seniman dari antrian (menggunakan prinsip FIFO) dan memprosesnya.

Konsep yang digunakan:

Queue dan Dequeue: Mengatur antrian partisipasi seniman dengan prinsip FIFO.

Concurrent Collections: Memastikan keamanan thread saat memproses partisipasi seniman.

```
Seniman: Alice, Kategori: Digital Art
Seniman: Bob, Kategori: Virtual Reality
```

Apa yang terjadi?

Program menampilkan daftar seniman yang telah mendaftar, yaitu:

Alice dengan kategori Digital Art.

Bob dengan kategori Virtual Reality.

Mengapa terjadi?

Seniman Alice dan Bob telah ditambahkan ke daftar seniman menggunakan method addArtist().

Method displayArtists() menampilkan informasi seniman yang tersimpan dalam List<Artist>.

Konsep yang digunakan:

List: Menyimpan daftar seniman secara berurutan.

Record: Menyimpan data seniman (Artist) secara efisien dan immutable.

```
Karya ID: K001, Deskripsi: Karya Digital Alice
Karya ID: K002, Deskripsi: Karya VR Bob
```

Apa yang terjadi?

Program menampilkan daftar karya seni yang telah dipamerkan, yaitu:

K001 dengan deskripsi Karya Digital Alice.

K002 dengan deskripsi Karya VR Bob.

Mengapa terjadi?

Karya seni K001 dan K002 telah ditambahkan ke daftar karya seni menggunakan method addArtwork().

Method displayArtworks() menampilkan informasi karya seni yang tersimpan dalam ConcurrentHashMap<String, Artwork>.

Konsep yang digunakan:

Map: Menyimpan informasi karya seni dengan ID unik sebagai kunci.

Record: Menyimpan data karya seni (Artwork) secara efisien dan immutable.

```
Partisipasi selesai: A001
Partisipasi selesai: A002
```

Apa yang terjadi?

Program menandai bahwa partisipasi seniman dengan ID A001 dan A002 telah selesai diproses.

Mengapa terjadi?

Setelah memproses partisipasi seniman, program menampilkan pesan bahwa partisipasi tersebut telah selesai.

Ini adalah bagian dari simulasi proses partisipasi yang dilakukan dalam method processParticipations().

Konsep yang digunakan:

Concurrent Collections: Memastikan keamanan thread saat memproses partisipasi seniman.

Queue dan Dequeue: Mengatur antrian partisipasi seniman dengan prinsip FIFO.

Link Zip Google Drive
https://drive.google.com/drive/folders/10dWkrkQor-GbJRytUAcOMTHQito1rNWG?usp=share_link