LAPORAN TUGAS TEKNIK PEMRORGAMAN

PERTEMUAN 5

NAMA: FAUZAN JUNIAR MULYANA

NIM: 241524011

POLITEKNIK NEGERI BANDUNG

PROGRAM STUDI D4-TEKNIK INFORMATIKA

2024

DAFTAR ISI

Definisi

1. List, Set, dan Map

List

- Definisi:
  List adalah struktur data yang menyimpan elemen dalam urutan tertentu dan memungkinkan duplikasi. Elemen dalam List dapat diakses menggunakan indeks, yang dimulai dari nol. Implementasi umum dari List di Java termasuk:

  - **ArrayList:** Menggunakan array dinamis untuk menyimpan elemen. Memungkinkan akses cepat ke elemen berdasarkan indeks, tetapi operasi penyisipan dan penghapusan bisa lambat jika dilakukan di tengah-tengah daftar.
  - **LinkedList:** Menggunakan struktur data berbasis node, di mana setiap node berisi data dan referensi ke node sebelumnya dan berikutnya. Ini membuat operasi penyisipan dan penghapusan lebih efisien dibandingkan ArrayList, terutama pada posisi awal atau tengah.

- Alasan Penggunaan:
  List sangat berguna ketika urutan elemen penting atau ketika Anda perlu mengizinkan duplikasi. Misalnya, dalam aplikasi pemesanan tiket, Anda mungkin ingin menyimpan daftar kursi yang sudah dipesan.

  Contoh Penerapan:

  - Mengelola daftar tugas (to-do list) di aplikasi manajemen waktu.
  - Menyimpan riwayat pencarian pengguna dalam aplikasi e-commerce untuk memberikan rekomendasi produk.

Set

- Definisi:
  Set adalah koleksi yang tidak mengizinkan elemen duplikat dan tidak menjamin urutan penyimpanan. Implementasi umum dari Set termasuk:

  - **HashSet:** Menggunakan tabel hash untuk menyimpan elemen. Ini memberikan waktu akses yang sangat cepat (O(1) untuk operasi dasar) tetapi tidak mempertahankan urutan.
  - **TreeSet:** Mengimplementasikan antarmuka NavigableSet dan menyimpan elemen dalam urutan alami atau berdasarkan komparator yang diberikan. Operasi pencarian, penyisipan, dan penghapusan memiliki kompleksitas waktu O(log n).

- **LinkedHashSet:** Menggabungkan fitur HashSet dan LinkedList, menjaga urutan penyisipan sambil tetap memberikan performa cepat.

- Alasan Penggunaan:
  Set sangat berguna ketika Anda perlu memastikan bahwa tidak ada duplikasi dalam kumpulan data. Misalnya, dalam aplikasi registrasi pengguna, Anda ingin memastikan bahwa setiap alamat email hanya terdaftar sekali.

Contoh Penerapan:

- Mengelola daftar ID unik pengguna dalam sistem pendaftaran.
- Mengumpulkan item-item unik yang telah dibeli oleh pengguna dalam aplikasi belanja.

Map

- Definisi:
  Map adalah koleksi yang menyimpan pasangan kunci-nilai, di mana setiap kunci bersifat unik. Implementasi umum dari Map termasuk:

  - **HashMap:** Menggunakan tabel hash untuk menyimpan pasangan kunci-nilai. Ini memberikan waktu akses yang sangat cepat (O(1) untuk operasi dasar) tetapi tidak mempertahankan urutan.
  - **TreeMap:** Menyimpan pasangan kunci-nilai dalam urutan terurut berdasarkan kunci. Operasi pencarian, penyisipan, dan penghapusan memiliki kompleksitas waktu O(log n).
  - **LinkedHashMap:** Menyimpan pasangan kunci-nilai dengan mempertahankan urutan penyisipan.

- Alasan Penggunaan:
  Map sangat berguna ketika Anda perlu mengasosiasikan nilai dengan kunci tertentu dan melakukan pencarian berdasarkan kunci tersebut. Misalnya, dalam aplikasi kamus, Anda dapat menggunakan kata sebagai kunci dan definisinya sebagai nilai.

Contoh Penerapan:

- Menyimpan konfigurasi aplikasi dalam format key-value (misal: config.put("timeout", 30)).
- Membangun indeks kata untuk dokumen teks agar pencarian lebih cepat.

2. Record

Definisi:

Record adalah fitur baru yang diperkenalkan di Java 14 sebagai cara untuk mendefinisikan kelas data sederhana dengan sintaksis yang lebih ringkas. Record secara otomatis menghasilkan metode equals(), hashCode(), dan toString(), serta konstruktor untuk semua field-nya. Dengan mendeklarasikan kelas sebagai record, dapat menghindari boilerplate code yang biasanya diperlukan untuk kelas data biasa.

Alasan Penggunaan:

Record membantu mempercepat pengembangan dengan mengurangi jumlah kode yang harus ditulis untuk kelas data sederhana. Dengan menjadikan record immutable secara default, Anda juga meningkatkan keamanan data karena objek tidak dapat dimodifikasi setelah dibuat.

Contoh Penerapan:

- Menyimpan hasil query database atau respons API dengan atribut tetap (misal: data produk dengan atribut id, nama, harga).
- Menggunakan record untuk mendefinisikan entitas dalam aplikasi seperti pengguna atau produk tanpa menulis banyak kode boilerplate.

3. Optional

Definisi:

Optional adalah kontainer objek yang mungkin berisi nilai atau tidak ada nilai sama sekali (null). Ini dirancang untuk menghindari masalah NullPointerException dengan memaksa pengembang untuk menangani kemungkinan nilai null secara eksplisit.

Alasan Penggunaan:

Menggunakan Optional meningkatkan keamanan kode dengan memaksa pengembang untuk berpikir tentang kasus di mana nilai mungkin tidak ada. Ini juga membuat kode lebih bersih dan mudah dibaca dibandingkan dengan menggunakan pemeriksaan null manual.

Contoh Penerapan:

- Dalam sistem pencarian pengguna, saat mencari berdasarkan ID pengguna, Anda dapat mengembalikan Optional<User> daripada User langsung untuk menunjukkan bahwa pengguna mungkin tidak ditemukan.
- Dalam API RESTful, Anda dapat menggunakan Optional sebagai respons ketika data mungkin tidak ada (misal: Optional<Product>).

4. Concurrent Collections

Definisi:

Concurrent Collections adalah koleksi yang dirancang khusus untuk digunakan dalam lingkungan multithreading tanpa memerlukan sinkronisasi manual dari pengembang. Beberapa implementasi utama termasuk:

- **ConcurrentHashMap:** Versi thread-safe dari HashMap yang memungkinkan beberapa thread membaca dan menulis ke peta secara bersamaan tanpa memerlukan penguncian penuh.
- **CopyOnWriteArrayList:** Daftar thread-safe di mana setiap kali ada perubahan pada daftar (penyisipan atau penghapusan), salinan baru dari array dibuat.
- **BlockingQueue:** Antarmuka Queue yang mendukung operasi blokir saat antrian kosong atau penuh (misal: ArrayBlockingQueue, LinkedBlockingQueue).

Alasan Penggunaan:

Concurrent Collections memungkinkan akses aman ke koleksi tanpa harus menangani sinkronisasi secara manual, sehingga mengurangi kompleksitas kode dan risiko kesalahan terkait multithreading.

Contoh Penerapan:

- Dalam aplikasi web yang menangani permintaan simultan dari banyak pengguna, ConcurrentHashMap dapat digunakan untuk menyimpan sesi pengguna.
- Dalam sistem pemrosesan pesan seperti RabbitMQ, BlockingQueue dapat digunakan untuk mengelola antrian pesan antara produsen dan konsumen secara aman.

5. Queue dan Deque

Definisi:

Queue adalah struktur data berbasis FIFO (*First-In-First-Out*), sedangkan Deque (*Double Ended Queue*) memungkinkan penambahan dan penghapusan elemen dari kedua ujungnya. Implementasi utama meliputi:

- Queue: Implementasi umum termasuk LinkedList (FIFO) dan PriorityQueue (elemen diproses berdasarkan prioritas).
- Deque: Implementasi seperti ArrayDeque memungkinkan akses cepat ke kedua ujung antrian.

Alasan Penggunaan:

Queue sangat berguna ketika Anda perlu memproses elemen dalam urutan kedatangan mereka (misal: antrian pelanggan). Deque memberikan fleksibilitas tambahan dengan memungkinkan operasi LIFO (*Last-In-First-Out*) jika diperlukan.

Contoh Penerapan:

- Dalam sistem antrian layanan pelanggan, Queue dapat digunakan untuk mengelola pelanggan yang menunggu giliran.
- Dalam aplikasi permainan, Deque dapat digunakan untuk menyimpan riwayat langkah pemain sehingga pemain dapat melakukan *undo/redo* pada langkah-langkah mereka.

6. Immutable Collection (List.of, Set.of, Map.of)

Definisi:

Immutable Collections adalah koleksi tak bisa diubah yang diperkenalkan di Java 9 melalui metode factory seperti List.of(), Set.of(), dan Map.of(). Koleksi ini tidak dapat dimodifikasi setelah dibuat; setiap upaya untuk menambah atau menghapus elemen akan menghasilkan pengecualian (UnsupportedOperationException).

Alasan Penggunaan:

Koleksi immutable membantu menjaga integritas data karena objek tidak bisa diubah setelah inisialisasi. Ini juga meningkatkan keamanan di lingkungan multithreading karena tidak ada risiko kondisi balapan pada koleksi tersebut.

Contoh Penerapan:

- Dalam aplikasi konfigurasi, Anda mungkin ingin membuat daftar opsi tetap agar tidak bisa dimodifikasi setelah aplikasi berjalan.
- Dalam sistem rekomendasi produk, Anda bisa menggunakan Immutable Collections untuk menyimpan kategori produk tetap agar tidak berubah selama runtime aplikasi.

Dengan penjelasan ini, Anda sekarang memiliki pemahaman yang lebih mendalam tentang masing-masing komponen dalam Java Collection Framework beserta alasan penggunaannya serta contoh penerapannya di dunia nyata! Jika ada pertanyaan lebih lanjut atau topik lain yang ingin dibahas, silakan beri tahu!

CONTOH KASUS PENERAPAN

1. List

```java
// File 2: InventoryService.java
private final List<Order> orderHistory = Collections.synchronizedList(new ArrayList<>());
```

2. Set

```java
// File 2: InventoryService.java
private final Set<String> categories = ConcurrentHashMap.newKeySet();
```

3. Map

```java
// File 2: InventoryService.java
private final ConcurrentHashMap<String, Integer> stock = new ConcurrentHashMap<>();
```

4. Record

```java
// File 1: Product.java
public record Product(
    String id,
    String name,
    double price,
    List<String> tags
) {}
```

5. Optional

```java
// File 2: InventoryService.java
public Optional<String> placeOrder(Order order) {
    return Optional.ofNullable(
        stock.compute(order.product().id(), (k, v) -> {
            if (v == null || v < order.quantity()) return v;
            orderQueue.offer(order);
            return v - order.quantity();
        })
    ).filter(newStock -> newStock != null && newStock >= 0);
}
```

6. Concurrent

```java
// File 2: InventoryService.java
private final PriorityBlockingQueue<Order> orderQueue = new PriorityBlockingQueue<>();
```

7. Queue

```java
// File 3: Order.java (Implementasi antrian prioritas)
public record Order(
    String id,
    Product product,
    int quantity,
    Instant timestamp,
    boolean isPriority
) implements Comparable<Order> {
    @Override
    public int compareTo(Order other) {
        // Bandingkan prioritas (VIP lebih dahulu), lalu waktu
        if (this.isPriority() != other.isPriority()) {
            return other.isPriority() ? 1 : -1;
        }
        return this.timestamp().compareTo(other.timestamp());
    }
}
```

8. Immutable

```java
// File 1: Product.java (Contoh pembuatan Immutable List)
public static Product create(String id, String name, double price, String... tags) {
    return new Product(id, name, price, List.of(tags)); // Tags tidak bisa diubah
}
```

PENJELASAN

1. **List**:
   Pada baris private final List<Order> orderHistory = Collections.synchronizedList(new ArrayList<>());, bagian dalam kurung new ArrayList<>() diisi dengan **new ArrayList<Order>()**, yang berarti membuat instance ArrayList untuk menyimpan objek bertipe Order (misalnya: data pesanan). Ini di-wrap dengan synchronizedList agar aman diakses oleh banyak thread.
2. **Set**:
   Pada private final Set<String> categories = ConcurrentHashMap.newKeySet();, isi dalam kurung ConcurrentHashMap.newKeySet() adalah **new ConcurrentHashMap<String, Boolean>().keySet()**, yang menghasilkan Set thread-safe untuk menyimpan kategori produk (misalnya: "Elektronik", "Aksesoris") dengan nilai unik.
3. **Map**:
   Pada private final ConcurrentHashMap<String, Integer> stock = new ConcurrentHashMap<>();, isi dalam kurung new ConcurrentHashMap<>() adalah **new ConcurrentHashMap<String, Integer>()**, yang membuat map dengan key bertipe String (ID produk) dan value bertipe Integer (stok produk). Contoh: {"P1": 100, "P2": 50}.
4. **Record**:
   Pada public record Product(String id, String name, double price, List<String> tags) {}, isi dalam kurung (String id, ...) adalah parameter data imutabel yang disimpan di Product, seperti:
   - id (ID produk, contoh: "P1"),
   - name (nama produk, contoh: "Laptop"),
   - price (harga, contoh: 15000000),
   - tags (daftar tag, contoh: List.of("Elektronik", "Premium")).
5. **Optional**:
   Pada Optional.ofNullable(stock.compute(...)), isi dalam kurung compute adalah **lambda expression** yang memeriksa stok

6. **Concurrent:**
   Pada private final PriorityBlockingQueue<Order> orderQueue = new PriorityBlockingQueue<>();, isi dalam kurung new PriorityBlockingQueue<>() adalah new PriorityBlockingQueue<Order>(), yang membuat antrian pesanan dengan prioritas (contoh: pesanan VIP diproses lebih dulu).
7. **Queue**:
   Pada implementasi compareTo di Order, isi dalam kurung adalah logika prioritas. Contoh: Pesanan VIP (isPriority=true) akan selalu di depan pesanan biasa.
8. **Immutable:**
   Pada List.of(tags), isi dalam kurung adalah **data konkret** yang dimasukkan ke koleksi imutabel, seperti List.of("Elektronik", "Premium"). Setelah dibuat, daftar ini tidak bisa diubah (tidak bisa ditambah/dihapus).