LAPORAN PRAKTIKUM TEKNIK PEMROGRAMAN
PERTEMUAN KELIMA: JAVA COLLECTION FRAMEWORK
PRAKTIKUM

Oleh:

Ihsan Fauzi 241524048

PROGRAM STUDI D4-TEKNIK INFORMATIKA
JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA
POLITEKNIK NEGERI BANDUNG
2025

DAFTAR ISI

BAB I
PENDAHULUAN

1. Latar belakang
   Dalam pengembangan perangkat lunak, pemilihan struktur data yang tepat sangat mempengaruhi efisiensi dan performa aplikasi. Java menyediakan Java Collections Framework (JCF) yang mencakup berbagai struktur data untuk menangani pengelolaan data secara optimal. Pemahaman mengenai List, Set, Map, Record, Optional, Concurrent Collection, Queue dan Dequeue, serta Immutable Collection sangat penting dalam membangun aplikasi yang efisien dan mudah dikelola.
2. Tujuan
   Laporan ini bertujuan untuk:
   - Memahami berbagai koleksi dalam Java dan penggunaannya dalam pemrograman.
   - Menganalisis kapan dan bagaimana menggunakan koleksi Java yang tepat sesuai kebutuhan.
   - Mempelajari implementasi koleksi Java dalam studi kasus nyata.

BAB II
LIST, SET, DAN MAP

1. Definisi

   Java Collections Framework menyediakan berbagai struktur data untuk mengelola kimpulan data secara efisien. Collection dibagi menjadi 3 yaitu
   - List adalah koleksi yang dapat menyimpan elemen dalam urutan tertentu dan memungkinkan duplikasi. Contoh implementasi: ArrayList, LinkedList.
   - Set adalah koleksi yang tidak mengizinkan elemen duplikat dan tidak menjamin urutan penyimpanan. Contoh implementasi: HashSet, TreeSet.
   - Map adalah koleksi yang menyimpan data dalam bentuk pasangan key-value atau kalau di python disebut dictionary, dimana setiap key bersifat unik. Biasanya setiap key memiliki perbedaan isi. Contoh implementasi: HashMap, TreeMap.
2. Kegunaan
   - List berguna untuk menyimpan dan mengakses data secara berurutan, misalnya daftar siswa dalam kelas
   - Set digunakan ketika kita ingin memastikan bahwa tidak ada elemen duplikat, seperti daftar ID user.
   - Map cocok untuk pencarian cepat berdasarkan kunci, misalnya menyimpan informasi pengguna berdasarkan username
3. Kapan harus digunakan
   - List ketika urutan data penting dan elemen dapat diduplikasi
   - Set ketika data harus unik dan tidak memerlukan pengurutan spesifik
   - Map ketika setiap elemen dikaitkan dengan suatu kunci dan harus diakses berdasarkan kunci tersebut.
4. Contoh penerapan

```
// 1. List digunakan untuk menyimpan riwayat transaksi secara thread-safe
//    Alasan: Menyimpan elemen berurutan, memungkinkan duplikat, cocok untuk riwayat transaksi.
private final List<CryptoTrade> tradeHistory = new CopyOnWriteArrayList<>();
```

   - CopyOnWriteArrayList digunakan untuk menyimpan riwayat transaksi dalam multi-threading.
   - Dengan struktur ini, setiap kali ada perubahan (penambahan atau penghapusan transaksi), Java akan membuat salinan baru dari daftar.
   - Ini menghindari kondisi balapan (race condition) di mana beberapa thread mengakses daftar transaksi yang sama.

```
// 1. Set digunakan untuk menyimpan daftar aset kripto yang dimiliki pengguna tanpa duplikasi
//    Alasan: Menggunakan Set memastikan bahwa seorang pengguna tidak memiliki aset yang sama lebih dari sekali.
private final Map<String, Set<String>> userAssets = new ConcurrentHashMap<>();
```

   - ConcurrentHashMap digunakan untuk memastikan akses yang aman dalam lingkungan multi-threading.
   - Set<String> digunakan agar setiap pengguna tidak memiliki aset yang sama lebih dari satu kali.

- Misalnya, jika seorang pengguna memiliki Bitcoin dan Ethereum, Set<String> akan mencegah pengguna memiliki Bitcoin dua kali.

```
// 1. Map digunakan untuk menyimpan harga aset kripto secara real-time
//    Alasan: Menyimpan pasangan key-value, cocok untuk harga aset atau mapping pengguna ke aset.
private final Map<String, Double> cryptoPrices = new ConcurrentHashMap<>();
```

- Map digunakan karena setiap aset kripto memiliki harga yang unik dan terus diperbarui.
- ConcurrentHashMap memungkinkan update harga dilakukan oleh beberapa thread secara aman, misalnya saat aplikasi menerima harga terbaru dari API pasar kripto.
- Contohnya, jika harga Bitcoin (BTC) berubah dari $65,000 menjadi $66,000, nilai dalam cryptoPrices akan diperbarui tanpa mengganggu thread lain yang membaca harga lama.

BAB III
RECORD

1. Definisi

   Dengan record, kita dapat membuat objek bahkan tipe data yang berbeda hanya dengan mendeklarasikan atributnya tanpa perlu menuliskan getter, setter , equals(), atau hashcode()

2. Kegunaan
   - Record digunakan untuk merepresentasikan data yang bersifat tetap dan tidak berubah walaupun punya tipe data yang berbeda, seperti hasil transaksi atau konfigurasi sistem, yang biasanya memiliki tipe data id integer dan user string
   - Mengurangi boilerplate code (kode yang dapat digunakan kembali dalam berbagai proyek perangkat lunak dengan sedikit atau tanpa perubahan)
3. Kapan harus digunakan
   - Ketika membutuhkan objek yang bersifat immutable dan hanya menyimpan data tanpa logika kompleks
   - Ketika ingin membuat representasi data dengan kode yang lebih ringkas dan mudah dibaca
4. Contoh penerapan

```
// 2. Record digunakan untuk merepresentasikan transaksi kripto secara immutable
//    Alasan: Record membuat objek transaksi bersifat immutable, sehingga aman untuk pemrograman konkurensi.
record CryptoTrade(String user, String asset, double amount, boolean isBuyOrder, double price) {}
```

   - Record memungkinkan pembuatan objek immutable tanpa perlu menuliskan banyak kode boilerplate seperti getter, setter, equals(), dan hashCode().
   - Dalam konteks aplikasi trading, transaksi yang telah dicatat tidak boleh diubah. Oleh karena itu, CryptoTrade dibuat sebagai record untuk memastikan bahwa nilai transaksi tetap sama setelah dibuat.
   - Misalnya, transaksi "User Alice membeli 0.5 BTC seharga $32,000" akan direpresentasikan sebagai objek CryptoTrade yang tidak bisa diubah setelah dibuat.

BAB IV
OPTIONAL

1. Definisi

   Optional digunakan untuk menangani nilai yang mungkin null dengan cara yang lebih aman. Daripada menggunakan null, Optional memberikan metode untuk menangani nilai yang ada (isPresent()) atau memberikan nilai default jika tidak ada (orElse)).

2. Kegunaan
   - Menghindari NullPointerException yang sering terjadi saat mengakses referensi null.
   - Agar dapat menangani kemungkinan nilai kosong
3. Kapan harus digunakan
   - Saat bekerja dengan data yang mungkin tidak tersedia, seperti hasil pencarian di database
   - Saat ingin menghindari pengolahan nilai null yang tidak terduga dalam aplikasi
4. Contoh penerapan

```
// 3. Menggunakan Optional untuk validasi aset
//    Alasan: Optional membantu menghindari NullPointerException
dan memberikan cara yang lebih aman untuk menangani data yang
mungkin tidak ada.
Optional<String> validAsset =
Optional.ofNullable(supportedCryptos.contains(asset) ? asset :
null);
```

   - Optional digunakan untuk menghindari NullPointerException saat memeriksa apakah aset yang dimasukkan pengguna valid atau tidak.
   - Jika aset tersebut termasuk dalam supportedCryptos, maka akan dikembalikan, tetapi jika tidak, akan menghasilkan nilai Optional.empty().
   - Contohnya, jika seorang pengguna mencoba membeli "DOGE", tetapi sistem hanya mendukung "BTC", "ETH", dan "ADA", maka validAsset akan kosong dan program dapat memberikan peringatan tanpa error.

BAB V
CONCURRENT COLLECTION

1. Definisi

   Concurrent Collection dirancang untuk bekerja dengan aman dalam lingkungan multi-threading. Koleksi ini termasuk ConcurrentHashMap, CopyOnWriteArrayList, dan ConcurrentLinkedQueue.

2. Kegunaan
   - ConcurrentHashMap: Struktur data berbasis key-value yang aman untuk beberapa thread
   - CopyOnWriteArrayList: List yang memungkinkan pembacaan tanpa konflik di antara beberap thread
   - ConcurrentLinkedQueue: Implementasi queue yang efisien untuk pemrograman konkurensi.
3. Kapan harus digunakan
   - Ketika beberapa thread harus mengakses koleksi data secara bersamaan tanpa risiko kondisi balapan (race condition)
   - Ketika performa lebih penting daripada synchronized
4. Contoh penerapan

```
private final Map<String, Set<String>> userAssets = new
ConcurrentHashMap<>();

private final Map<String, Double> cryptoPrices = new
ConcurrentHashMap<>();

private final Queue<CryptoTrade> orderQueue = new
ConcurrentLinkedQueue<>();

        userAssets.computeIfAbsent(trade.user(), k ->
ConcurrentHashMap.newKeySet()).add(trade.asset());
```

   - userAssets menyimpan daftar aset pengguna, sementara cryptoPrices menyimpan harga aset yang terus berubah secara real-time.
   - Contohnya, jika dua pengguna membeli Ethereum (ETH) secara bersamaan, maka sistem tetap bisa menangani transaksi dengan aman tanpa error.

BAB VI
QUEUE DAN DEQUEUE

1. Definisi
    - Queue adalah antrian yang menerapkan prinsip FIFO (First In, First Out)
    - Dequeue (Double-Ended Queue) memungkinkan penambahan dan penghapusan elemen di kedua ujungnya
2. Kegunaan
    - Queue berguna dalam antrean tugas yang dieksekusi dalam urutan tertentu, seperti pemrosesan transaksi atau antrian dalam marketplace online
    - Dequeue berguna dalam skenario yang membutuhkan fleksibilitas dalam penyisipan dan penghapusan elemen, seperti implementasi undo-redo
3. Kapan harus digunakan
    - Gunakan Queue untuk antrean tugas atau antrian pelanggan
    - Gunakan Dequeue ketika memerlukan operasi antrian yang fleksibel
4. Contoh penerapan

```
// 5. Queue digunakan untuk antrean order yang belum dieksekusi
//    Alasan: ConcurrentLinkedQueue memastikan FIFO (First-In, First-Out)
dan aman digunakan dalam lingkungan multithreading.
private final Queue<CryptoTrade> orderQueue = new
ConcurrentLinkedQueue<>();
```

- Queue digunakan untuk menyimpan order jual/beli sebelum dieksekusi.
- Contohnya, ketika pengguna memasang order beli Bitcoin, transaksi ini masuk ke orderQueue dan akan diproses sesuai urutan (FIFO - First In, First Out).

```
orderQueue.offer(trade); // 5. Menambahkan order ke antrean
```

- removeIf() digunakan untuk menghapus order tertentu tanpa perlu iterasi manual, yang lebih efisien dalam multi-threading.
- Contohnya, jika pengguna ingin membatalkan order pembelian Ethereum, sistem dapat langsung menghapus ordernya dari antrean tanpa mengganggu order lain.

```
public void dequeueOrder(String user, String asset) {
    // 7. Menggunakan removeIf untuk menghapus order tertentu dari antrean
    //    Alasan: removeIf lebih efisien dibandingkan iterasi manual,
sehingga menghindari kondisi balapan (race condition).
    boolean removed = orderQueue.removeIf(trade ->
trade.user().equals(user) && trade.asset().equals(asset));
```

```
while (!orderQueue.isEmpty()) {
    CryptoTrade trade = orderQueue.poll(); // 5. FIFO
    futures.add(executor.submit(() -> executeTrade(trade)));
}
```

- poll() mengambil dan menghapus order pertama dalam antrean.
- executor.submit() menjalankan order tersebut di thread terpisah untuk meningkatkan efisiensi pemrosesan.
- Contohnya, jika antrean order berisi 10 transaksi, maka transaksi yang masuk lebih dulu akan dieksekusi lebih dulu.

BAB VII
IMMUTABLE COLLECTION (LIST.OF, SET.OF, MAP.OF)

1. Definisi

   Java menyediakan metode List.of(), Set.of(), dan Map.of() untuk membuat koleksi yang immutable (tidak dapat diubah setelah dibuat).

2. Kegunaan
   - Mencegah modifikasi data yang tidak disengaja.
   - Menghemat penggunaan memori dengan menghindari perubahan data yang tidak perlu.

3. Kapan harus digunakan
   - Saat bekerja dengan data konstan, seperti daftar mata uang kripto yang didukung dalam aplikasi perdagangan.
   - Saat ingin mencegah perubahan data dalam koleksi yang tidak seharusnya dimodifikasi oleh pengguna lain.

4. Contoh penerapan

```
// 6. Immutable Collection untuk daftar mata uang kripto yang
tersedia
//    Alasan: Menggunakan Set.of memastikan daftar aset ini tetap
konstan dan tidak bisa dimodifikasi.
private final Set<String> supportedCryptos = Set.of("BTC", "ETH",
"ADA", "XRP", "SOL");
//keunikan jadi pertimbangan
```

   - Set.of() membuat daftar mata uang kripto yang tidak dapat diubah setelah didefinisikan.
   - Ini memastikan bahwa daftar aset yang didukung tetap konstan sepanjang masa pakai aplikasi.
   - Contohnya, jika developer ingin menambah koin baru seperti "DOGE", perubahan hanya bisa dilakukan dengan memodifikasi kode sumber, bukan secara langsung dalam runtime.

```
// 6. Immutable Collection List.of untuk daftar pengguna awal
//    Alasan: Menggunakan List.of memastikan daftar ini tetap
konstan dan tidak dapat dimodifikasi.
private final List<String> initialUsers = List.of("Alice", "Bob",
"Charlie");
//mungkin duplikat
```

   - List.of() digunakan untuk membuat daftar pengguna awal yang tidak dapat dimodifikasi.
   - Ini berguna jika aplikasi memiliki daftar pengguna default yang harus tetap sama sepanjang waktu.
   - Contohnya, aplikasi mungkin memiliki akun demo untuk "Alice", "Bob", dan "Charlie", yang digunakan untuk uji coba tanpa risiko perubahan data.

```
public CryptoTradingSystem() {
    // 6. Immutable Collection untuk harga awal aset
    //    Alasan: Menggunakan Map.of memastikan nilai awal harga
kripto tidak bisa diubah setelah inisialisasi.
    this.cryptoPrices.putAll(Map.of(
            "BTC", 65000.0, "ETH", 3500.0, "ADA", 1.2, "XRP", 0.6,
"SOL", 150.0
```

```
    ));
}
```

- Map.of() digunakan untuk menetapkan harga awal aset yang tidak dapat dimodifikasi setelah aplikasi berjalan.
- Contohnya, harga awal untuk Bitcoin = $65,000, yang hanya dapat diubah oleh API real-time, bukan oleh pengguna aplikasi.

BAB VIII
KESIMPULAN

Pemilihan struktur data yang tepat dalam Java sangat bergantung pada kebutuhan aplikasi yang sedang dikembangkan. **List, Set, dan Map** digunakan untuk mengelola data dengan berbagai aturan unik, sedangkan **Record** membantu dalam representasi objek immutable. **Optional** mengurangi risiko NullPointerException, sementara **Concurrent Collection** memastikan akses data yang aman dalam lingkungan multi-threading. **Queue dan Dequeue** sangat berguna dalam pengelolaan antrean tugas, sedangkan **Immutable Collection** membantu menjaga integritas data yang tidak boleh diubah.

Dengan pemahaman yang baik tentang konsep-konsep ini, pengembang perangkat lunak dapat meningkatkan efisiensi, keamanan, dan keandalan aplikasi mereka, terutama dalam sistem yang membutuhkan pemrosesan data secara paralel dan dalam skala besar.

BAB IX
SOURCE CODE

```
import java.util.*;
import java.util.concurrent.*;

// 2. Record digunakan untuk merepresentasikan transaksi kripto secara
immutable
//    Alasan: Record membuat objek transaksi bersifat immutable, sehingga
aman untuk pemrograman konkurensi.
record CryptoTrade(String user, String asset, double amount, boolean
isBuyOrder, double price) {}

public class CryptoTradingSystem {
    // 1. List digunakan untuk menyimpan riwayat transaksi secara thread-
safe
    //    Alasan: Menyimpan elemen berurutan, memungkinkan duplikat, cocok
untuk riwayat transaksi.
    private final List<CryptoTrade> tradeHistory = new
CopyOnWriteArrayList<>();

    // 1. Set digunakan untuk menyimpan daftar aset kripto yang dimiliki
pengguna tanpa duplikasi
    //    Alasan: Menggunakan Set memastikan bahwa seorang pengguna tidak
memiliki aset yang sama lebih dari sekali.
    private final Map<String, Set<String>> userAssets = new
ConcurrentHashMap<>();

    // 1. Map digunakan untuk menyimpan harga aset kripto secara real-time
    //    Alasan: Menyimpan pasangan key-value, cocok untuk harga aset atau
mapping pengguna ke aset.
    private final Map<String, Double> cryptoPrices = new
ConcurrentHashMap<>();

    // 5. Queue digunakan untuk antrean order yang belum dieksekusi
    //    Alasan: ConcurrentLinkedQueue memastikan FIFO (First-In, First-
Out) dan aman digunakan dalam lingkungan multithreading.
    private final Queue<CryptoTrade> orderQueue = new
ConcurrentLinkedQueue<>();

    // 6. Immutable Collection untuk daftar mata uang kripto yang tersedia
    //    Alasan: Menggunakan Set.of memastikan daftar aset ini tetap
konstan dan tidak bisa dimodifikasi.
    private final Set<String> supportedCryptos = Set.of("BTC", "ETH",
"ADA", "XRP", "SOL");
    //keunikan jadi pertimbangan

    // 6. Immutable Collection List.of untuk daftar pengguna awal
    //    Alasan: Menggunakan List.of memastikan daftar ini tetap konstan
dan tidak dapat dimodifikasi.
    private final List<String> initialUsers = List.of("Alice", "Bob",
"Charlie");
    //mungkin duplikat

    public CryptoTradingSystem() {
        // 6. Immutable Collection untuk harga awal aset
        //    Alasan: Menggunakan Map.of memastikan nilai awal harga kripto
tidak bisa diubah setelah inisialisasi.
        this.cryptoPrices.putAll(Map.of(
                "BTC", 65000.0, "ETH", 3500.0, "ADA", 1.2, "XRP", 0.6,
```

```
"SOL", 150.0
        ));
    }

    // Fungsi untuk menempatkan order beli atau jual
    public void placeOrder(String user, String asset, double amount,
boolean isBuyOrder) {
        // 3. Menggunakan Optional untuk validasi aset
        //    Alasan: Optional membantu menghindari NullPointerException
dan memberikan cara yang lebih aman untuk menangani data yang mungkin tidak
ada.
        Optional<String> validAsset =
Optional.ofNullable(supportedCryptos.contains(asset) ? asset : null);

        validAsset.ifPresentOrElse(a -> {
            double price = cryptoPrices.get(asset); // Mengambil harga
terbaru
            CryptoTrade trade = new CryptoTrade(user, asset, amount,
isBuyOrder, price);
            orderQueue.offer(trade); // 5. Menambahkan order ke antrean
            System.out.println("Order ditambahkan: " + trade);
        }, () -> System.out.println("Aset tidak didukung!"));
    }

    // Fungsi untuk membatalkan (dequeue) order sebelum diproses
    public void dequeueOrder(String user, String asset) {
        // 7. Menggunakan removeIf untuk menghapus order tertentu dari
antrean
        //    Alasan: removeIf lebih efisien dibandingkan iterasi manual,
sehingga menghindari kondisi balapan (race condition).
        boolean removed = orderQueue.removeIf(trade ->
trade.user().equals(user) && trade.asset().equals(asset));
        if (removed) {
            System.out.println("Order untuk " + user + " dengan aset " +
asset + " berhasil dihapus dari antrean.");
        } else {
            System.out.println("Order tidak ditemukan atau sudah
diproses.");
        }
    }

    // Fungsi untuk memproses antrean order
    public void processOrders() {
        // 4. Menggunakan ExecutorService untuk memproses order secara
paralel
        //    Alasan: Thread pool meningkatkan efisiensi eksekusi transaksi
secara bersamaan tanpa harus membuat thread baru setiap kali.
        ExecutorService executor = Executors.newFixedThreadPool(3);
        List<Future<?>> futures = new ArrayList<>();

        while (!orderQueue.isEmpty()) {
            CryptoTrade trade = orderQueue.poll(); // 5. FIFO
            futures.add(executor.submit(() -> executeTrade(trade)));
        }

        // Menunggu semua task selesai
        for (Future<?> future : futures) {
            try {
                future.get();
            } catch (InterruptedException | ExecutionException e) {
                e.printStackTrace();
```

```
            }
        }

        executor.shutdown();
        try {
            executor.awaitTermination(5, TimeUnit.SECONDS);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    // Fungsi eksekusi transaksi
    private void executeTrade(CryptoTrade trade) {
        tradeHistory.add(trade); // 1. Menyimpan transaksi ke dalam list
        userAssets.computeIfAbsent(trade.user(), k ->
ConcurrentHashMap.newKeySet()).add(trade.asset());
        System.out.println("Order dieksekusi: " + trade);
    }

    // Menampilkan riwayat transaksi
    public void printTradeHistory() {
        tradeHistory.forEach(System.out::println);
    }

    // Menampilkan daftar aset pengguna
    public void printUserAssets(String user) {
        System.out.println("Aset milik " + user + ": " +
userAssets.getOrDefault(user, Set.of()));
    }

    // Menampilkan harga kripto saat ini
    public void printCryptoPrices() {
        System.out.println("Harga Kripto Saat Ini: " + cryptoPrices);
    }

    public static void main(String[] args) {
        CryptoTradingSystem system = new CryptoTradingSystem();

        system.placeOrder("Alice", "BTC", 0.5, true);
        system.placeOrder("Bob", "ETH", 2.0, true);
        system.placeOrder("Charlie", "ADA", 50.0, true);
        system.placeOrder("Charlie", "XRP", 200.0, true);

        // Membatalkan salah satu order sebelum diproses
        system.dequeueOrder("Charlie", "XRP");

        system.processOrders();

        System.out.println("\nRiwayat Transaksi:");
        system.printTradeHistory();
        system.printUserAssets("Alice");
        system.printCryptoPrices();
    }
}
```

BAB X
LINK GITHUB

https://github.com/isanzzi/collection-framework-java