SISTEM TERDISTRIBUSI

“Tutorial Java Stream”

Disusun Oleh:

Nama: Amelia

Kelas: TRPL 3B

No BP: 2111082006

DOSEN PENGAMPU

ERVAN ASRI.S.Kom.,M.Kom

POLITEKNIK NEGERI PADANG

2023/2024

JAVA STREAM

adalah fitur yang diperkenalkan dalam **Java 8** untuk memungkinkan pemrosesan data koleksi dengan cara yang lebih fungsional, ekspresif, dan efisien. Stream adalah urutan elemen yang dapat Anda proses secara sekuensial atau paralel. Dengan Java Stream, Anda dapat melakukan berbagai operasi pada data koleksi seperti List, Set, atau Map tanpa harus menulis loop tradisional.

Berikut adalah beberapa konsep utama terkait dengan Java Stream:

1. **Stream**: Stream adalah urutan elemen yang dapat Anda operasikan. Ini dapat berasal dari berbagai sumber, seperti List, Set, Array, atau bahkan data yang dihasilkan secara dinamis.

2. **Operasi Intermediate**: Operasi intermediate adalah operasi yang dapat Anda terapkan pada Stream dan menghasilkan Stream baru sebagai output. Contoh operasi intermediate meliputi `filter`, `map`, `distinct`, dan `sorted`. Operasi ini biasanya digunakan untuk memfilter atau mengubah data dalam Stream.

3. **Operasi Terminal**: Operasi terminal adalah operasi akhir yang Anda terapkan pada Stream dan menghasilkan hasil atau nilai non-Stream. Contoh operasi terminal meliputi `forEach`, `collect`, `reduce`, dan `count`. Operasi ini mengakhiri aliran data dan menghasilkan hasil akhir.

4. **Lazy Evaluation**: Salah satu fitur kunci dari Java Stream adalah lazy evaluation. Ini berarti Stream hanya dievaluasi saat operasi terminal diterapkan. Ini dapat meningkatkan efisiensi dalam beberapa kasus, karena Anda hanya menghitung data yang benar-benar Anda butuhkan.

5. **Parallel Processing**: Anda dapat dengan mudah menggunakan Java Stream untuk melakukan pemrosesan paralel dengan hanya mengganti metode `stream()` dengan `parallelStream()`. Ini memungkinkan Anda memanfaatkan CPU multi-core untuk memproses data dengan lebih cepat.

6. **Pipelines**: Anda dapat menggabungkan beberapa operasi Stream ke dalam pipa (pipeline) untuk melakukan serangkaian operasi pada data dengan satu pernyataan.

Contoh Program Java Stream

1. Menggunakan Stream untuk Menghitung Jumlah Elemen dalam Sebuah List

```java
import java.util.Arrays;
import java.util.List;

public class ContohStream1 {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9

        long count = numbers.stream()
                .count();

        System.out.println("Jumlah elemen dalam list: " + count);
    }
}
```

2. Menggunakan Stream untuk Menggandakan Setiap Elemen dalam Sebuah List

```java
import java.util.Arrays;
import java.util.List;

public class ContohStream2 {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

        List<Integer> hasil = numbers.stream()
                .map(n -> n * 2)
                .collect(Collectors.toList());

        System.out.println("List hasil: " + hasil);
    }
}
```

3. Menggunakan Stream untuk Memfilter Elemen dalam Sebuah List

```java
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class ContohStream3 {
    public static void main(String[] args) {
        List<String> kata = Arrays.asList("apel", "banana", "ceri", "du

        List<String> hasil = kata.stream()
                .filter(s -> s.startsWith("a"))
                .collect(Collectors.toList());

        System.out.println("Hasil filter: " + hasil);
    }
}
```

4. Menggunakan Stream untuk Mengurutkan Elemen dalam Sebuah List

```java
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class ContohStream4 {
    public static void main(String[] args) {
        List<String> kata = Arrays.asList("apel", "banana", "ceri", "du

        List<String> hasil = kata.stream()
                .sorted()
                .collect(Collectors.toList());

        System.out.println("List yang diurutkan: " + hasil);
    }
}
```