Tutorial Mengenai Java Stream

1. Berikut cara membuat stream:

Kita dapat membuat stream dari berbagai jenis koleksi, seperti List, Set, dan Array.
Conto berikut adalah penggunaan stream pada List:

```
import java.util.List;
import java.util.stream.Stream;

public class Main {
    public static void main(String[] args) {
        List<String> fruits = List.of("Apple", "Banana", "Cherry", "Date");

        Stream<String> stream = fruits.stream();
    }
}
```

2. Operasi Filter:

Kita dapat menggunakan metode “filter()” untuk memfilter elemen dalam stream berdasarkan kondisi tertentu.
Berikut merupakan contoh :

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

public class Main {
    public static void main(String[] args) {
        List<String> fruits = List.of("Apple", "Banana", "Cherry", "Date");

        Stream<String> filteredStream = fruits.stream()
                .filter(fruit -> fruit.startsWith("A"));

        List<String> filteredFruits =
filteredStream.collect(Collectors.toList());
        System.out.println(filteredFruits); // Output: [Apple]
```

```
    }
}
```

3. Operasi Map

Dengan metode map “map()”, kita dapat mengubah setiap elemen dalam stream menjadi elemen lain.
Berikut merupakan contoh:

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

public class Main {
    public static void main(String[] args) {
        List<String> fruits = List.of("Apple", "Banana", "Cherry", "Date");

        Stream<String> mappedStream = fruits.stream()
                .map(fruit -> fruit.toUpperCase());

        List<String> uppercaseFruits =
mappedStream.collect(Collectors.toList());
        System.out.println(uppercaseFruits); // Output: [APPLE, BANANA,
CHERRY, DATE]
    }
}
```

4. Operasi Reduce:

Metode reduce() dapat digunakan untuk menggabungkan elemen-elemen dalam stream menjadi satu nilai:
Berikut merupan contoh:

```
import java.util.List;
import java.util.Optional;
import java.util.stream.Stream;

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

```
        Optional<Integer> sum = numbers.stream()
                .reduce((a, b) -> a + b);

        sum.ifPresent(result -> System.out.println("Sum: " + result));
// Output: Sum: 15
    }
}
```

5. Collectors:

Anda dapat mengumpulkan hasil dari operasi stream ke dalam berbagai bentuk koleksi menggunakan Collectors
Berikut merupakan contoh:

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

public class Main {
    public static void main(String[] args) {
        List<String> fruits = List.of("Apple", "Banana", "Cherry",
"Date");

        List<String> filteredFruits = fruits.stream()
                .filter(fruit -> fruit.startsWith("A"))
                .collect(Collectors.toList());

        System.out.println(filteredFruits); // Output: [Apple]
    }
}
```

Contoh Program

Contoh 1 Menjumlahkan Seluruh Bilangan dalam Sebuah List

```
import java.util.List;
public class Main {
    public static void main(String[] args) {
        List<Integer> numbers = List.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

        int sum = numbers.stream()
                .mapToInt(Integer::intValue)
                .sum();

        System.out.println("Jumlah semua bilangan: " + sum); // Output:
Jumlah semua bilangan: 55
    }
}
```

Contoh 2 Mengunakan Reduce untuk Menghitung Jumlah

```
import java.util.List;
import java.util.Optional;

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

        // Menggunakan reduce untuk menghitung jumlah semua angka.
        Optional<Integer> sum = numbers.stream()
                .reduce((a, b) -> a + b);

        sum.ifPresent(result -> System.out.println("Sum: " + result)); //
Output: Sum: 15
    }
}
```