Modul Praktikum

Fungsi

I. Fungsi Imperatif dan Deklaratif

Pertama Ketikan kode di bawah ini

```
public class DoubleNumberImp {
     public static int[] findAndDoubleEvenNumbers(int[] arr) { int[]
          numbers = {1, 2, 3, 4, 5};
int[] result = new int[numbers.length]; for (int i =
          0; i < numbers.length; i++) {
               if (arr[i] % 2 == 0) {
                    result[i] = arr[i] * 2;
}
}
return result;
}

public static void main(String[] args) {

int[] numbers = {1, 2, 3, 4, 5};
int[] result = new int[numbers.length]; result =
          findAndDoubleEvenNumbers(numbers);

          for(int i=0; i<result.length;i++){
                    System.out.print(result[i]+" ");
}
}
}
```

Lalu Ketikkan kode di bawah ini

```
import java.util.Arrays;


public class DoubleNumberDeclarative {

     public static int[] findAndDoubleEvenNumbers(int[] arr) {

          return Arrays.stream(arr)

.filter(num -> num % 2 == 0)

.map(num -> num * 2)

.toArray();

}


public static void main(String[] args) {


int[] numbers = {1, 2, 3, 4, 5};

int[] result = new int[numbers.length]; result =

          findAndDoubleEvenNumbers(numbers); for(int

          i=0; i<result.length;i++){

System.out.print(result[i]+" ");

}

}
}
```

1. Bagaimana hasil dari kedua output kode di atas
    - Hasil kode yang imperative adalah : 0 4 0 8 0
    - Hasil kode yang declarative adalah : 4 8
2. Dari Hasil output dan jalannya kedua program tersebut coba analisis bagaimana perbedaan fungsi imperatif dan deklaratif?
    - untuk fungsi yang imperative mengenerate elemen array yang telah ditentukan sekaligus memfilter elemen yang memenuhi kriteria tersebut, alhasil 0 4 0 8 0
        - jadi `int[] result = new int[numbers.length]; menghasilkan array [0,0,0,0,0] setelah itu baru ditambahkan dengan elemen yang sesuai. ( eager Evaluation )`
    - untuk fungsi yang declarative dia memilah dahulu apa yang dibutuhkannya (number -> number % 2 == 0) setelah itu membuat array baru sesuai dengan kriteria dan akhirnya diproses menggunakan .map setelah itu dikonversi lagi ke bertipe array :D (karena saat didalam proses stream mempunyai tipe data unik jadi harus di konversi ke tipe array lagi setelah selesai menggunakan .stream)

- Apa kegunaan dari map dan filter dari kode deklaratif di atas?
    - .map dan .filter merupakan intermediate function dalam java streams
    - .map digunakan untuk menggapply function yg di telah ditentukan, dalam konteks ini adalah (num -> num * 2) disetiap element yang ada
    - .filter hampir sama seperti .map tetapi dia hanya return element yang hanya sesuai dengan fungsi yang diberikan (number -> number % 2 == 0)
        - .filter menggunakan fungsi bersifat predicate yang artinya hanya mereturn tipe data boolean saja (yang ini nya () ->)

II. Fungsi Imperatif dan Deklaratif

Kedua Ketikkan kode di bawah ini

```
public class PrimeInteratif {
    public static boolean isPrime(long number) {
        for(long i = 2; i <= Math.sqrt(number); i++)
        {
            if(number % i == 0) return false;
        }
        return number > 1;
    }

    public static void main(String[]
        args) { boolean hasil;
        hasil = isPrime(9220000000000000039L);
        System.out.println(hasil);
    }
}
```

Lalu Ketikkan kode di bawah ini

```
import java.util.stream.LongStream;

public class PrimeDeclarative {
    public static boolean isPrime(long number) {
        return number > 1 && LongStream
.rangeClosed(2, (long) Math.sqrt(number))
.parallel()
.noneMatch(index -> number % index == 0);
}

    public static void main(String[] args) {
        boolean hasil;
hasil = isPrime(9220000000000000039L);
        System.out.println(hasil);
  }
```

- Apa kegunaan dari Longstream berdasarkan dari kode pemrograman deklaratif di atas?
    - untuk LongStream adalah stream untuk dikhususkan untuk memproses tipe data Long yang didalamnya juga ada fitur tambahan untuk memproses tipe data long

- Dari Hasil output dan jalannya kedua program tersebut coba analisis bagaimana perbedaan fungsi imperatif dan deklaratif?

    - hasil output dari 2 kode adalah true tetapi mempunyai perbedaan waktu yang signifikan dalam mengeluarkan output

    - perbedaan dari 2 fungsi dari kode yang diberikan adalah :

        - untuk yang menggunakan methode imperative terlihat lebih kompleks, dan juga menggunakan for loop dengan interasi yang lama sekali karena mengulang sesuai hasil akar dari argumen yang diberikan

        - sedangkan untuk fungsi declarative lebih terlihat lebih teknis dan lebih simpel dalam penggunaan method dari LongStream kode ini lebih cepat dikarenakan menggunakan method paralel() yang akan membagi tugasnya ke berbagai threads dalam processor

        - dan tugas yang dibagi tersebut adalah .noneMatch(index -> number % index == 0).

Tugas

Buatlah Pemrograman untuk menghitung faktorial dengan cara imperatif dan deklaratif

Imperative :

```
public static int factorial(int amount){
        for (int i = amount; amount > 1; i--) {
            if(i == 1){
                return amount;
            }
                amount *= i - 1;
        }
        return 0;
    }
```

```
public static void main(String[] args) {
        System.out.println(factorial(5)); // 120
    }
```

- untuk imperative dalam for loop tersebut variabel i akan diisi oleh jumlah faktorial dari argumen setelah itu dikurangi setiap looping
    - setelah itu jika i == 1 maka hasil total amount akan dikembalikan
        - ini dibuat untuk pengestop looping
    - dan jika amount > 1 maka loop terus sekaligus mereplace hasil amount dan mengalikan hasil tersebut dengan i lagi
        - jadi seperti amount = amount * (i - 1)
    - return 0 sebenarnya untuk menghindari error yang diharuskan oleh java

Declarative :

```
public static int factorials(int amount){
        return IntStream
        .rangeClosed(1, amount)
        .reduce(1, (prev, current) -> prev * current );
    }
```

```
public static void main(String[] args) {
        System.out.println(factorials(5));
    }
```

- untuk argumen amount akan digunakan untuk .rangeClosed() untuk menghasilkan angka dari 1 sampai yang ditentukan
- setelah itu menggunakan .reduce untuk menjumlah hasil perkaliannya dari hasil rangeClosed itu [1,2,3,4,5]
    - jadi seperti [hasil, hasil * elemen yang berikutnya]
    - [1 * 1] -> [1 * 2] [2, 2 * 3] [6, 6 * 4] [24, 24 * 5][120, limit reached]

- .reduce digunakan untuk menghasilkan satu hasil dari berbagai elemen yang ada (Ugarte)
    - jadi berbagai elemen tersebut bisa diproses dengan berbagai operasi :D

Works Cited

Baeldung. “Java Stream Filter with Lambda Expression.” *Baeldung*, 17 October 2023, https://www.baeldung.com/java-stream-filter-lambda. Accessed 2 November 2023.

Juneja, Jai. “What is Predicate in Java 8?” *Scaler*, https://www.scaler.com/topics/predicate-in-java-8/. Accessed 2 November 2023.

Oracle. “Stream (Java SE 9 & JDK 9).” *Oracle Help Center*, https://docs.oracle.com/javase/9/docs/api/java/util/stream/Stream.html#map-java.util.function.Function-. Accessed 2 November 2023.

“Stream (Java Platform SE 8).” *Oracle Help Center*, https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html. Accessed 2 November 2023.

Ugarte, Alejandro. “Guide to Stream.reduce().” *Baeldung*, 17 October 2023, https://www.baeldung.com/java-stream-reduce. Accessed 1 November 2023.