

SISTEM TERDISTRIBUSI

TUTORIAL JAVA STREAM

NAMA : NEISHA SALSABILA

KELAS : TRPL 3A

SEMESTER 5

JURUSAN TEKNOLOGI INFORMASI

PROGRAM STUDI REKAYASA PERANGKAT LUNAK

POLITEKNIK NEGERI PADANG

2023

Java Stream adalah fitur yang diperkenalkan dalam Java 8 untuk memungkinkan pemrosesan data berbasis koleksi dengan lebih mudah, fungsional, dan deklaratif. Stream memungkinkan melakukan berbagai operasi pada data koleksi seperti List, Set, Map, Array, atau bahkan data dari sumber lain dengan pendekatan yang mirip dengan SQL atau pemrograman fungsional.

Stream adalah aliran (sequence) elemen yang dapat diakses sekuensial atau paralel. Stream tidak menyimpan elemen-elemen dalam memori, yang berarti kita tidak perlu peduli tentang bagaimana elemen-elemen tersebut disimpan atau diakses. Operasi-operasi pada Stream biasanya bersifat deklaratif dan fungsional.

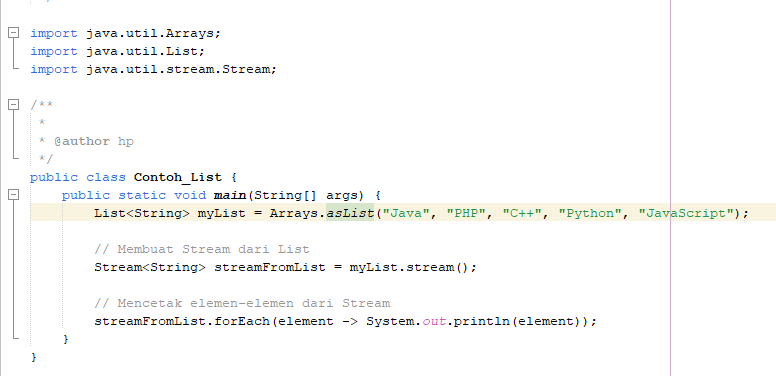
Kita dapat membuat Stream dari berbagai jenis koleksi seperti List, Set, Map, Array, atau bahkan menggunakan metode statis seperti Stream.of() dan Stream.iterate().

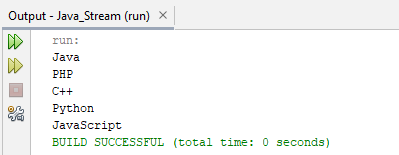
Berikut adalah tutorial tentang penggunaan Java Stream dengan beberapa contoh programnya :

1. Source

Java Stream dapat dibuat dari berbagai jenis sumber data, seperti List, Set, Map, Array, atau bahkan aliran data dari sumber eksternal seperti file.

1. List





* Import Package Java:

Pada awal kode, kita mengimpor pustaka yang diperlukan dengan pernyataan import. Ini termasuk :

java.util.Arrays: Digunakan untuk menginisialisasi List dengan elemen-elemen yang diberikan.

java.util.List: Digunakan untuk mendefinisikan tipe data List.

java.util.stream.Stream: Digunakan untuk membuat dan mengoperasikan Stream.

* Inisialisasi List :

List ini berisi beberapa elemen string seperti "java", "c++", dll. Kita menggunakan Arrays.asList(...) untuk menginisialisasi List ini dengan elemen-elemen yang telah diberikan.

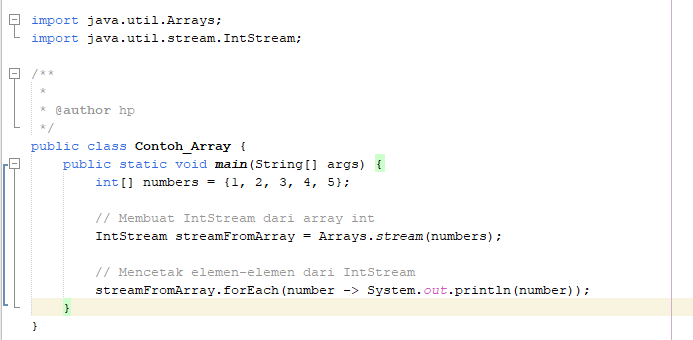
* Membuat Stream dari List :

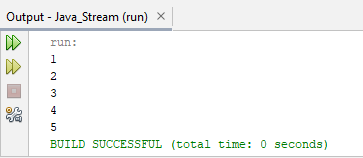
Kode selanjutnya membuat objek Stream dengan nama streamFromList dari List myList. Ini dilakukan dengan menggunakan metode stream() yang tersedia pada objek List.

* Mencetak Elemen-elemen dari Stream:

Setelah membuat Stream dari List, kita menggunakan metode forEach() untuk mencetak setiap elemen dari Stream ke Output.

1. Array





* Pada awal kode, kita mengimpor pustaka yang diperlukan dengan pernyataan import. Dalam contoh ini, kita mengimpor:

java.util.Arrays: Digunakan untuk menggunakan metode stream() dari kelas Arrays untuk membuat stream dari array.

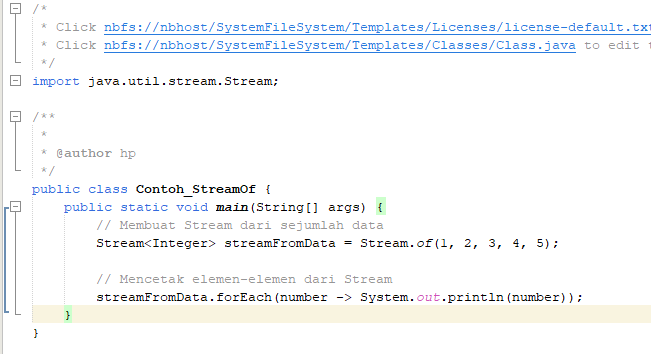
java.util.stream.IntStream: Digunakan untuk bekerja dengan IntStream.

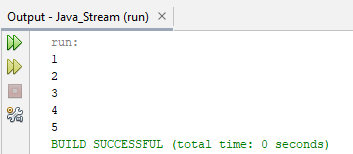
* Mencetak Elemen-elemen dari Stream:

Setelah membuat Stream dari List, kita menggunakan metode forEach() untuk mencetak setiap elemen dari Stream ke Output. Dalam contoh ini, setiap elemen string dalam List akan dicetak satu per satu.

* Metode forEach() menerima sebuah lambda expression yang akan diterapkan pada setiap elemen dalam Stream. Lambda expression ini digunakan untuk mencetak setiap elemen ke Output menggunakan System.out.println().

1. Stream.of





* Pada awal kode, kita mengimpor pustaka yang diperlukan dengan pernyataan import. Dalam contoh ini, kita mengimpor:

java.util.Arrays: Digunakan untuk menggunakan metode stream() dari kelas Arrays untuk membuat stream dari array.

java.util.stream.IntStream: Digunakan untuk bekerja dengan IntStream.

* Inisialisasi Array numbers :

Kita mendefinisikan array numbers yang berisi beberapa nilai bilangan bulat.

* Membuat IntStream dari Array:

Kode selanjutnya menggunakan Arrays.stream(numbers) untuk membuat IntStream dari array numbers. Arrays.stream(...) adalah cara untuk mengubah array menjadi stream dalam Java.

* Mencetak Elemen-elemen dari IntStream:

Setelah membuat IntStream dengan nama streamFromArray, kita menggunakan metode forEach() untuk mencetak setiap elemen dari stream ke Output.

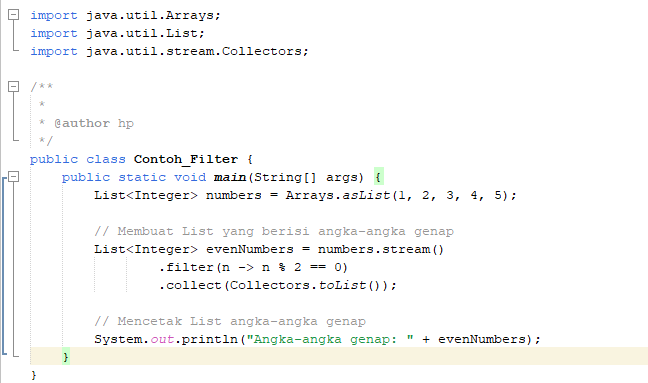
Lambda expression number -> System.out.println(number) digunakan untuk mencetak setiap elemen stream.

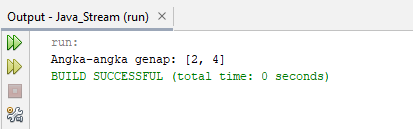
1. Operasi Intermidate

Operasi intermediate adalah operasi yang dapat Anda tumpuk (chain) satu sama lain untuk memproses elemen dalam stream. Beberapa operasi intermediate umum meliputi :

1. filter()

Menggunakan filter() untuk memilih elemen berdasarkan kriteria tertentu:





* Import Package Java:

Pada awal kode, kita mengimpor pustaka yang diperlukan dengan pernyataan import. Dalam contoh ini, kita mengimpor:

java.util.Arrays: Digunakan untuk menginisialisasi List dengan elemen-elemen yang diberikan.

java.util.List: Digunakan untuk mendefinisikan tipe data List.

java.util.stream.Collectors: Digunakan untuk mengumpulkan hasil filter menjadi List.

* Inisialisasi List numbers :

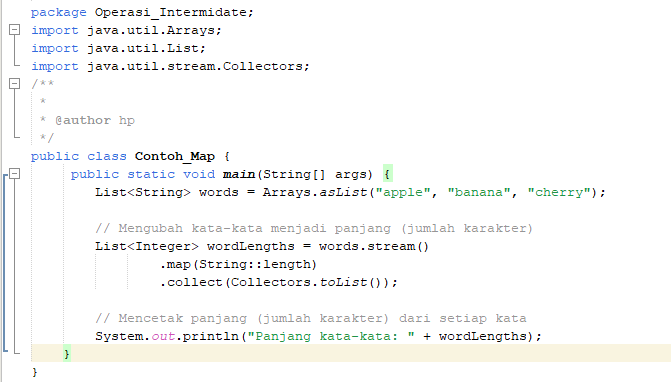
Kode selanjutnya inisialisasi List ‘numbers’ dengan beberapa angka, yaitu [1, 2, 3, 4, 5].

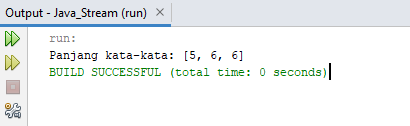
* Menggunakan Java Stream untuk Memfilter Angka Genap:
* Pada langkah ini, kita menggunakan Java Stream untuk memfilter angka-angka genap dari List ‘numbers’.
* Kita memanggil metode .’stream()’ pada List ‘numbers’ untuk mendapatkan Stream dari List tersebut.
* Kemudian kita menggunakan operasi .’filter(n -> n % 2 == 0)’ untuk mempertahankan hanya elemen-elemen yang memenuhi kondisi bahwa mereka adalah angka genap.
* Operasi ini akan menghasilkan Stream yang hanya berisi angka-angka genap.
* Mengumpulkan Hasil Filter ke List Baru:
* Setelah memfilter angka-angka genap, kita menggunakan metode .’collect(Collectors.toList())’ untuk mengumpulkan hasil filter tersebut ke dalam List baru bernama evenNumbers.
* Hasil filter yang sebelumnya dalam bentuk Stream akan dikumpulkan ke dalam List ‘evenNumbers’.
* Mencetak Hasil Filter:

Akhirnya, kita mencetak hasil filter, yaitu angka-angka genap, ke Output dengan menggunakan ‘System.out.println()’. map()

1. map()

Menggunakan map() untuk mengubah elemen dalam stream.





* Import Package Java:
* Pada awal kode, kita mengimpor pustaka yang diperlukan dengan pernyataan import. Dalam contoh ini, kita mengimpor:

java.util.Arrays: Digunakan untuk menginisialisasi List dengan elemen-elemen yang diberikan.

java.util.List: Digunakan untuk mendefinisikan tipe data List.

java.util.stream.Collectors: Digunakan untuk mengumpulkan hasil transformasi menjadi List.

* Inisialisasi List words:

Kode selanjutnya inisialisasi List words dengan beberapa kata, yaitu ["apple", "banana", "cherry"].

* Menggunakan Java Stream untuk Mengubah Kata-kata Menjadi Panjang:
* Pada langkah ini, kita menggunakan Java Stream untuk mengubah kata-kata dalam List words menjadi panjang (jumlah karakter) dari setiap kata.
* Kita memanggil metode .stream() pada List words untuk mendapatkan Stream dari List tersebut.
* Kemudian kita menggunakan operasi .map(String::length) untuk mengubah setiap kata menjadi panjangnya dalam bentuk integer.
* Operasi ini akan menghasilkan Stream yang berisi panjang dari setiap kata.
* Mengumpulkan Hasil Transformasi ke List Baru:

Setelah mengubah kata-kata menjadi panjang, kita menggunakan metode ‘.collect(Collectors.toList())’ untuk mengumpulkan hasil transformasi tersebut ke dalam List baru dengan nama wordLengths.

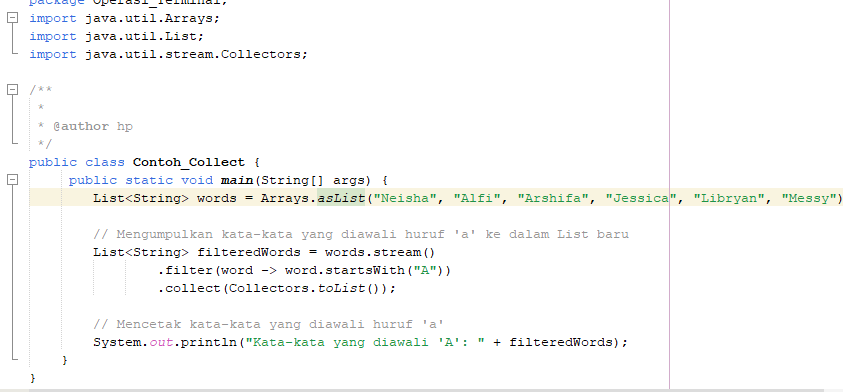
* Mencetak Hasil Transformasi:

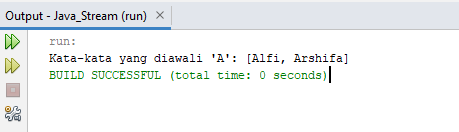
Akhirnya, kita mencetak hasil transformasi, yaitu panjang (jumlah karakter) dari setiap kata, ke Output dengan menggunakan System.out.println().

1. Operasi Terminal

Operasi terminal adalah operasi yang mengakhiri stream dan menghasilkan hasil akhir atau nilai tertentu. Beberapa operasi terminal umum meliputi:

1. collect()





* Import Package Java:

Pada awal kode, kita mengimpor pustaka yang diperlukan dengan pernyataan import. Dalam contoh ini, kita mengimpor:

java.util.Arrays: Digunakan untuk menginisialisasi List dengan elemen-elemen yang diberikan.

java.util.List: Digunakan untuk mendefinisikan tipe data List.

java.util.stream.Collectors: Digunakan untuk mengumpulkan hasil filter menjadi List.

* Inisialisasi List words:

Kode selanjutnya inisialisasi List words dengan beberapa kata, yaitu "Neisha", "Alfi", "Arshifa", "Jessica", "Libryan", "Messy"].

* Menggunakan Java Stream untuk Filter Kata-kata:
* Pada langkah ini, kita menggunakan Java Stream untuk memfilter kata-kata dalam List words yang diawali dengan huruf 'a'.
* Kita memanggil metode .stream() pada List words untuk mendapatkan Stream dari List tersebut.
* Kemudian kita menggunakan operasi .filter(word -> word.startsWith("a")) untuk mempertahankan hanya kata-kata yang memenuhi kondisi bahwa mereka diawali dengan huruf 'a'.
* Operasi ini akan menghasilkan Stream yang berisi kata-kata yang memenuhi kondisi tersebut.
* Mengumpulkan Hasil Filter ke List Baru:

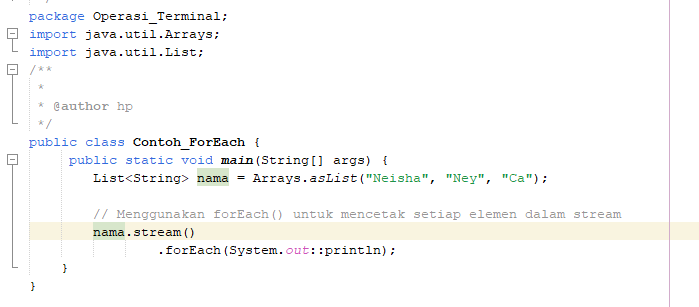
Setelah memfilter kata-kata, kita menggunakan metode . collect(Collectors.toList()) untuk mengumpulkan hasil filter tersebut ke dalam List baru dengan nama filteredWords.

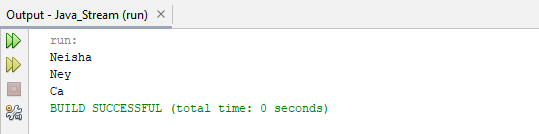
* Mencetak Hasil Filter:

Akhirnya, kita mencetak hasil filter, yaitu kata-kata yang diawali dengan huruf 'a', ke Output dengan menggunakan System.out.println().

1. forEach()

forEach() untuk menjalankan tindakan pada setiap elemen dalam stream.





* Import Package Java:

Pada awal kode, kita tidak perlu mengimpor pustaka tambahan karena hanya menggunakan pustaka dasar Java.

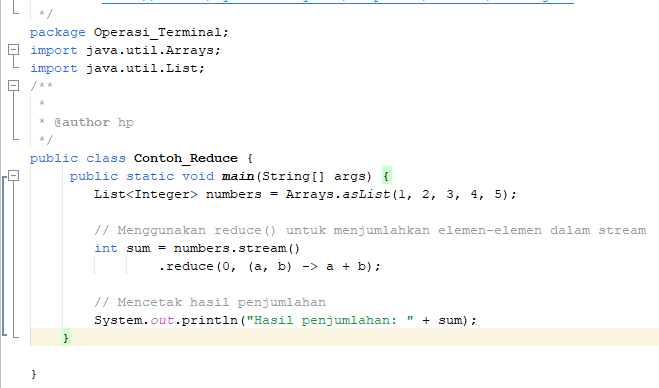
* Inisialisasi List nama:

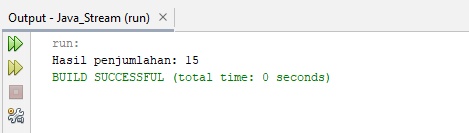
Kode selanjutnya inisialisasi List fruits dengan beberapa buah, yaitu Neisha", "Ney", "Ca"].

* Menggunakan Java Stream dan forEach():
* Pada langkah ini, kita menggunakan Java Stream untuk memproses elemen-elemen dalam List fruits.
* Kita memanggil metode .stream() pada List fruits untuk mendapatkan Stream dari List tersebut.
* Kemudian kita menggunakan operasi .forEach(System.out::println) untuk menjalankan tindakan mencetak setiap elemen stream.
* Lambda expression System.out::println digunakan untuk mencetak setiap elemen ke Output.

1. reduce()

Menggunakan reduce() untuk menggabungkan elemen dalam stream menjadi satu hasil.





* Import Package Java:

Pada awal kode, kita tidak perlu mengimpor pustaka tambahan karena hanya menggunakan pustaka dasar Java.

* Inisialisasi List numbers:

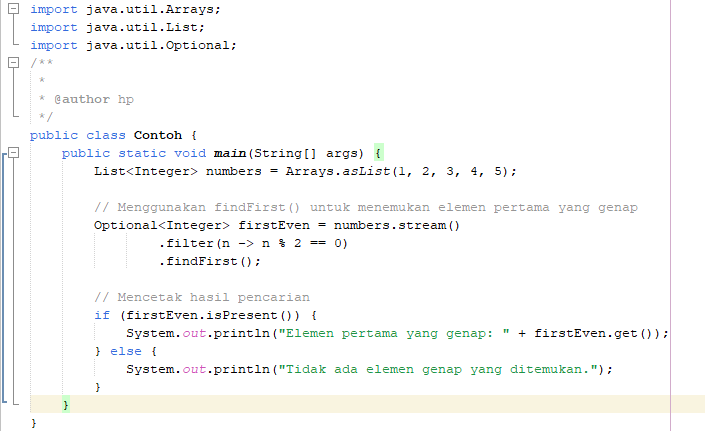
Kode selanjutnya inisialisasi List numbers dengan beberapa angka, yaitu [1, 2, 3, 4, 5].

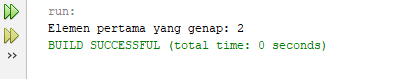
* Menggunakan Java Stream dan reduce():
* Pada langkah ini, kita menggunakan Java Stream untuk memproses elemen-elemen dalam List numbers.
* Kita memanggil metode .stream() pada List numbers untuk mendapatkan Stream dari List tersebut.
* Kemudian kita menggunakan operasi .reduce(0, (a, b) -> a + b) untuk menjumlahkan semua elemen dalam stream.
* Argumen pertama (0) dalam reduce() adalah nilai awal atau nilai identitas untuk operasi penjumlahan.
* Lambda expression (a, b) -> a + b digunakan untuk menjalankan operasi penjumlahan.
* Operasi ini akan menghasilkan satu nilai hasil yang merupakan jumlah dari semua elemen dalam stream.
* Mencetak Hasil Penjumlahan:

Akhirnya, kita mencetak hasil penjumlahan ke Output dengan menggunakan System.out.println().

1. Operasi Bersyarat (Short-Circuit)

Beberapa operasi terminal bersyarat akan menghentikan proses stream jika kondisi tertentu terpenuhi, seperti findFirst() dan anyMatch().





* Import Package Java:

Pada awal kode, kita tidak perlu mengimpor pustaka tambahan karena hanya menggunakan pustaka dasar Java.

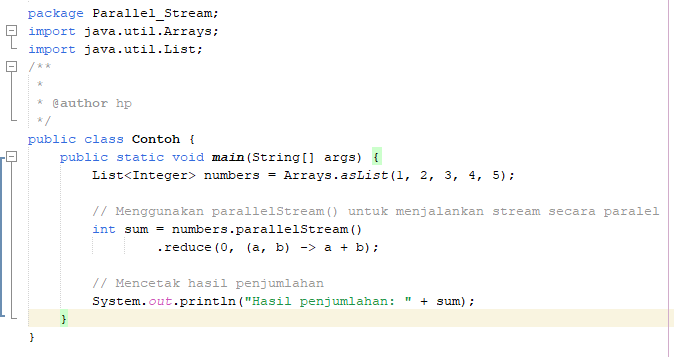
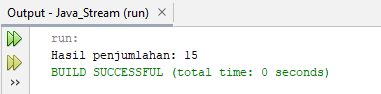
* Inisialisasi List numbers:

Kode selanjutnya inisialisasi List numbers dengan beberapa angka, yaitu [1, 2, 3, 4, 5].

* Menggunakan Java Stream dan findFirst():
* Pada langkah ini, kita menggunakan Java Stream untuk memproses elemen-elemen dalam List numbers.
* Kita memanggil metode .stream() pada List numbers untuk mendapatkan Stream dari List tersebut.
* Kemudian kita menggunakan operasi .filter(n -> n % 2 == 0) untuk memfilter hanya elemen yang merupakan angka genap.
* Operasi ini akan menghasilkan Stream yang hanya berisi angka genap.
* Terakhir, kita menggunakan operasi .findFirst() untuk menemukan elemen pertama dalam stream yang memenuhi kondisi. Operasi ini akan mengembalikan Optional<Integer> yang mungkin berisi elemen pertama yang memenuhi kondisi atau kosong jika tidak ada elemen yang memenuhi kondisi.
* Mencetak Hasil Pencarian:
* Setelah menggunakan findFirst(), kita memeriksa apakah hasilnya ada (ada elemen yang memenuhi kondisi) dengan if (firstEven.isPresent()). Jika ada, maka kita mencetak elemen tersebut ke Output dengan firstEven.get().
* Jika tidak ada elemen yang memenuhi kondisi, kita mencetak pesan bahwa tidak ada elemen genap yang ditemukan.

1. Menggunakan Parallel Stream

Kita dapat mengubah stream menjadi parallel stream untuk memproses elemen secara paralel dengan cara yang lebih efisien:



* Import Package Java:

Pada awal kode, kita tidak perlu mengimpor pustaka tambahan karena hanya menggunakan pustaka dasar Java.

* Inisialisasi List numbers:

Kode selanjutnya inisialisasi List numbers dengan beberapa angka, yaitu [1, 2, 3, 4, 5].

* Menggunakan Parallel Stream:
* Pada langkah ini, kita menggunakan parallel stream (parallelStream()) untuk memproses elemen-elemen dalam List numbers secara paralel.
* Kita memanggil metode .parallelStream() pada List numbers untuk mendapatkan parallel stream.
* Kemudian kita menggunakan operasi .reduce(0, (a, b) -> a + b) untuk menjumlahkan semua elemen dalam parallel stream.
* Argumen pertama (0) dalam reduce() adalah nilai awal atau nilai identitas untuk operasi penjumlahan.
* Lambda expression (a, b) -> a + b digunakan untuk menjalankan operasi penjumlahan.
* Operasi ini akan menghasilkan satu nilai hasil yang merupakan jumlah dari semua elemen dalam parallel stream.
* Mencetak Hasil Penjumlahan:

Setelah menggunakan reduce(), kita mencetak hasil penjumlahan ke Output dengan menggunakan System.out.println().