**JAVA STREAM**

## **Pendahuluan**

## ## Sebelum Belajar Materi ini ##

## sebaiknya mempelajari Java Dasar, Java Object Oriented Programming, Java Generic, Java Collection, Java Lambda, Apache Maven, Java Unit Test

## ## Agenda ##

Pengenalan Java Stream, Membuat Stream, Lazy Evaluation, Stream Builder, Stream Operations, Collectors, Parallel Stream

## **Pengenalan Java Stream**

* Java Stream diperkenalkan pertama kali di Java versi 8 atau 1.8
* Java Stream adalah implementasi di Java untuk aliran data
* Java Stream erat kaitannya dengan Java Collection dan Lambda Expression
* Semua fitur Java Stream terdapat di package java.util.stream
* <https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/util/stream/package-summary.html>

## Apa itu Stream? ##

* Stream adalah aliran data
* Berbeda dengan Array atau Collection yang merupakan kumpulan data
* Biasanya dalam Array atau Collection, kita selalu focus terhadap data, sedangkan dalam Stream, kita akan focus terhadap operasi apa yang akan kita lakukan

## Class java.util.stream.Stream<T> ##

* Java Stream diimplementasikan oleh sebuah class bernama Stream di dalam package java.util.stream
* Java Stream tidak sekompleks Java Collection, namun walaupun sederhana, fitur Java Stream sangat powerfull
* Class Stream merupakan Generic Class, sehingga kitab isa membuat aliran data dengan tipe yang kita inginkan

## **Membuat Stream**

* Ada banyak cara untuk membuat Stream di Java
* Bisa membuat langsung Stream dari Array, atau dari Collection

## Menjalankan Stream ##

* Secara default, Stream itu bersifat Cold, artinya data di Stream tidak akan mengalir sampai kita memintanya
* Ada banyak cara untuk meminta Stream mulai mengalirkan datanya, hal ini akan kita bahas di bagian Stream Operations
* Untuk saat ini, salah satu method yang bis akita gunakan agar aliran data di Stream mengalir adalah menggunakan method forEach
* Stream hanya bisa jalan sekali, mirip seperti aliran data, setelah mengalir, aliran data tidak bisa diulang lagi dari awal

## **Stream Builder**

* Kadang kita ingin membuat Stream secara manual, seperti menambahkan datanya ke Stream secara manual misalnya
* Java menyediakan Stream Builder untuk membuat Stream secara manual, dan kita bisa menggunakannya seperti Collection, kita bisa membuat Stream Builder, dan menambahkan data ke Stream Builder, setelah selesai, baru kita buat Stream nya
* https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/util/stream/Stream.Builder.html

## **Stream Operations**

* Berbeda dengan Java Collection, di Java Stream hal yang sangat menarik adalah fitur Stream Operations nya
* Stream Operations adalah kumpulan operasi-operasi yang bis akita gunakan untuk memanipulasi Stream itu sendiri
* Ada baanyak sekali jenis – jenis Stream Operation yang nanti akan kita bahas satu persatu
* Namun secara garis besar, Stream Operations tidak akan memodifikasi data aslinya, melainkan hasil dari Stream Operation adalah sebuah Stream baru

## **Stream Pipiline**

* Saat kita membuat Stream, biasanya kita akan melakukan banyak operasi terhadap Stream tersebut
* Dan biasanya kita akan membuat Stream Pipiline
* Stream Pipiline terdiri dari sebuah suber stream(bisa array, collection dan lain-lain), lalu diikuti dengan kosong atau lebih stream operations dan diakhiri dengan operasi akhir, misalnya forEach
* Saat menggunakan Stream, hamper kebanyakan kita pasti akan membuat sebuah Stream Pipiline

## **Lazy Evaluation**

* Stream secara dedfault adalah lazy
* Dia tidak akan mengalirkan data jika belum menggunakan operation
* Selain itu, data hanya akan dialirkan seperlunya saja, tergantung terminal operation nya

## intermediate & Terminal Operations ##

* Secara garis besar, ada 2 jenis Stream Operation, yaitu Intermediate dan Terminal Operations
* Intermediate Operations merupakan lazy operation, dimana Stream tidak akan dieksekusi sampai memang dibutuhkan
* Sedangkan Terminal Operation merupakan operasi yang mentrigger sebuah Stream berjalan
* Karena Intermediate Operations adalah lazy, maka secara garis besar, semua Intermediate Operations akan mengembalikan Stream lagi

## **Transformation Operations**

* Ada banyak Stream Operation di Java Stream, kita akan mulai bahas dari Transformation Operations
* Transformation Operations adalah operasi yang digunakan untuk mengubah bentuk Stream
* Ada banyak function yang bisa digunakan untuk mengubah bentuk Stream menjadi sebuah Stream baru, contohnya map dan flatMap

## Method Transformation Operations ##

|  |  |
| --- | --- |
| Method | Keterangan |
| Map(T -> U) | Mengubah Stream T menjadi Stream U |
| flatMap(T -> Stream<U>) | Mengubah Stream T menjadi Stream U |

## **Filtering Operations**

* Filtering operations adalah operasi di Stream yang digunakan melakukan filter data Stream
* Contohnya, missal kita punya Stream data dari 1 sampai 10, lalu kita missal hanya ingin menggunakan data yang ganjil atau genap saja

## Method Filtering Operations ##

|  |  |
| --- | --- |
| Method | Keterangan |
| filter(U -> Boolean) | Mengambil data yang masuk kriteria filter |
| distinct() | Menghapus semua data duplikat |

## **Retrieving Operations**

* Retrieving Operations adalah operasi pada Stream untuk melakukan pengambilan Sebagian data
* Secara garis besar, cara kerjanya hampir mirip dengan Filtering

## Method Retrieving Operations ##

|  |  |
| --- | --- |
| Method | Keterangan |
| limit(n) | Mengambil sejumlah n data |
| skip(n) | Menghiraukan sejumlah n data |
| takeWhile(T -> Boolean) | Mengambil data selama kondisi true |
| dropWhile(T -> Boolean) | Menghiraukan data selama kondisi true |

## Retrieving Single Element ##

* Stream juga memiliki kemampuan untuk mengambil satu element saja
* Namun operasi jenis ini merupakan operasi terminal, sehingga akan secara otomatis menjalankan aliran data di Stream

## Method Retriving Single Element ##

|  |  |
| --- | --- |
| Method | Keterangan |
| findAny() | Mengambil random satu element |
| findFirst() | Mengambil element pertama |

## **Ordering Operations**

* Java Stream juga mendukung operasi untuk melakukan pengurutan data Stream nya
* Secara default, data akan diurutkan mengikutin Comparable yang terdapat di data yang ada di Stream
* Jika kita ingin mengurutkan secara manual, kitab isa menggunakan Comparator sendiri

## Method Ordering Operations ##

|  |  |
| --- | --- |
| Method | Keterangan |
| sorted() | Mengurutkan berdasarkan comparable data |
| sorted(Comparator) | Mengurutkan berdasarkan comparator |

## **Aggreagate Operations**

* Java Stream mendukung banyak operasi untuk melakukan proses aggregate
* Seperti menghitung jumlah data, menghitung data max dan menghitung data min

## Method Aggregate Operations ##

|  |  |
| --- | --- |
| Method | Keterangan |
| max(Comparator) | Mencari data max sesuai comparator |
| min(Comparator) | Mencari data min sesuai comparator |
| count() | Menghitung total data |

## Manual Aggregate Menggunakan Reduce ##

* Java Stream juga menyediakan sebuah operasi yang bernama reduce
* Reduce bisa digunakan untuk melakukan proses aggregate secara manual
* Misal kita ingin menjumlahkan seluruh angka yang terdapat di Stream, kita bisa melakukan ini menggunakan reduce operator
* Di bahasa pemrograman lain, reduce mirip seperti operasi fold

## **Check Operations**

* Check Operations adalah operasi yang digunakan untuk melakukan pengecekan data di dalam Stream
* Ada banyak sekali operasi yang bisa digunakan untuk melakukan pengecekan
* Dan hasil operasi check adalah boolean

## Method Check Operations ##

|  |  |
| --- | --- |
| Method | Keterangan |
| anyMatch(T -> Boolean) | Apakah ada salah satu data yang match dengan kondisi |
| allMatch(T -> Boolean) | Apakah semua data match dengan kondisi |
| noneMatch(T -> Boolean) | Apakah semua data tidak match dengan kondisi |

## **For Each Operations**

* Untuk mengiterasi data satu persatu, sebelumnya kita sudah sering menggunakan operasi forEach
* Selain forEach, ada juga method untuk melakukan for each, tapi tanpa harus melakukan terminal operations

## Method For Each Operations ##

|  |  |
| --- | --- |
| Method | Keterangan |
| forEach(T -> void) | Melakukan iterasi satu per satu data di Stream. Ini adalah terminal operation |
| peek(T -> void) | Melakukan iterasi satu per satu data di Stream, namun mengembalikan Stream lagi, dan ini bukanlah terminal operation |

## **Primitive Stream**

* Sebelumnya kita sudah tahu bahwa implementasi Stream di Java adalah java.util.stream.Stream<T>
* Namun bagaimana jika kita butuh melakukan stream terhadap data primitive seperti int, long atau double?
* Karena untuk Generic hanya bisa menampung tipe data Object, maka di Java Stream, dibuat implementasi Stream khusus untuk tipe data primitive

## Primitive Stream Class ##

|  |  |
| --- | --- |
| Class | Keterangan |
| java.util.stream.IntStream | Stream untuk tipe data int |
| java.util.stream.LongStream | Stream untuk tipe data long |
| java.util.stream.DoubleStream | Stream untuk tipe data double |

## Primitive Stream Class ##

* Hampir semua stream operator yang pernah kita bahas, ada juga di primitive stream class
* Bahkan ada beberapa operator yang lebih sederhana, seperti untuk aggregate, kita tidak perlu menggunakan comparator lagi, bahkan ada operator average() untuk menghitung rata – rata di primitive stream
* Cara pembuatan primitive stream pun hampir mirip dengan Stream biasa, kita bisa gunakan static method di class nya, missal IntStream.of(…), IntStream.builder(), dan lain – lain

## **Collectors**

## Collect Operation ##

* Stream memiliki sebuah operator bernama collect(Collector), function ini biasanya digunakan untuk meng-collect data Stream dan kita ubah menjadi struktur data yang kita inginkan, biasanya kebanyakan developer menggunakan operator collect() untuk mengubah Stream menjadi Collection
* Operator collect() membutuhkan parameter Collector, namun biasanya kita jarang sekali membuat implementasi interface Collector, karena terlalu kompleks
* <https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/util/stream/Collector.html>
* Untungnya Java Stream sudah menyediakan sebuah class helper untuk membuat Collector, bernama Collectors

## Collectors ##

* Collectors adalah class helper yang bisa digunakan untuk membuat Collector
* Ini mempermudah kita ketika ingin melakukan operasi collect terhadap sebuah Stream
* Ada banyak sekali static method yang terdapat di class Collectors, dan nanti kita akan coba bahas beberapa method yang sering digunakan
* https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/util/stream/Collectors.html

## Membuat Map dengan Collectors ##

* Selain Collection, Collectors juga bisa digunakan untuk membuat Map dari String
* Yang membedakan dengan List atau Set, kita harus tentukan function untuk membentuk Key dan Value nya jika ingin membuat Map
* Ada banyak function yang bisa kita gunakan, seperti Collectors.toMap(…), Collectors.toConcurrentMap(…) dan Collectors.toUnmodifiableMap(…)

## **Grouping By**

* Collectors juga bisa digunakan untuk melakukan grouping by sebuah Stream
* Hasil dari grouping by adalah Map<Group, List<Value>>

## Partitioning By ##

* Selain grouping by, Collectors juga bisa digunakan untuk partitioning by
* Hanya saja hasil dari partitioning by hanyalah 2 group boolean(true, false)
* Hal ini berarti partitioning by hanya akan menghasilkan Map<Boolean, List<Value>>

## **Parallel Stream**

* Salah satu fitur yang menarik di Java Stream adalah, Stream bisa dijalankan secara parallel
* Materi tentang Parallel Programming akan dibahas lebih detail di materi terkait Java Thread
* Secara garis besar, parallel artinya beberapa proses berjalan secara bersamaan
* Secara default, Parallel Stream akan dijalankan di ForkJoinPool, dimana akan di running secara default menggunakan Thread sejumlah maksimal total CPU kita