Documentation Java Stream API

Méthodes Stream

Méthodes terminales - boolean

- `allMatch(Predicate<? super T> predicate)` | `boolean`
  Vérifie si tous les éléments satisfont la condition
- `anyMatch(Predicate<? super T> predicate)` | `boolean`
  Vérifie si au moins un élément satisfait la condition
- `noneMatch(Predicate<? super T> predicate)` | `boolean`
  Vérifie qu'aucun élément ne satisfait la condition

Méthodes intermédiaires - Stream

- `builder()` | `Stream.Builder<T>`
  Crée un constructeur de stream
- `concat(Stream<? extends T> a, Stream<? extends T> b)` | `Stream<T>`
  Concatène deux streams en un seul
- `distinct()` | `Stream<T>`
  Élimine les doublons du stream
- `empty()` | `Stream<T>`
  Crée un stream vide
- `filter(Predicate<? super T> predicate)` | `Stream<T>`
  Garde seulement les éléments qui satisfont la condition
- `generate(Supplier<T> s)` | `Stream<T>`
  Génère un stream infini avec une fonction fournisseur
- `iterate(T seed, UnaryOperator<T> f)` | `Stream<T>`
  Génère un stream infini par itération (seed, f(seed), f(f(seed))…)
- `limit(long maxSize)` | `Stream<T>`
  Limite le stream aux n premiers éléments
- `map(Function<? super T,? extends R> mapper)` | `Stream<R>`
  Transforme chaque élément avec la fonction donnée
- `of(T t)` | `Stream<T>`
  Crée un stream d'un seul élément
- `of(T... values)` | `Stream<T>`
  Crée un stream à partir des éléments donnés
- `peek(Consumer<? super T> action)` | `Stream<T>`
  Exécute une action sur chaque élément sans modifier le stream
- `skip(long n)` | `Stream<T>`
  Ignore les n premiers éléments
- `sorted()` | `Stream<T>`
  Trie les éléments dans l'ordre naturel
- `sorted(Comparator<? super T> comparator)` | `Stream<T>`
  Trie les éléments avec le comparateur donné

Méthodes spécialisées

- `collect(Collector<? super T,A,R> collector)` | `R`
  Collecte les éléments dans une structure de données (List, Set, Map…)
- `collect(Supplier<R> supplier, BiConsumer<R,? super T> accumulator, BiConsumer<R,R> combiner)` | `R`
  Collecte avec un collecteur personnalisé
- `flatMap(Function<? super T,? extends Stream<? extends R>> mapper)` | `Stream<R>`
  Transforme chaque élément en stream puis les aplatit en un seul stream
- `flatMapToDouble(Function<? super T,? extends DoubleStream> mapper)` | `DoubleStream`
  FlatMap vers un stream de double
- `flatMapToInt(Function<? super T,? extends IntStream> mapper)` | `IntStream`
  FlatMap vers un stream d'entiers
- `flatMapToLong(Function<? super T,? extends LongStream> mapper)` | `LongStream`
  FlatMap vers un stream de long
- `mapToDouble(ToDoubleFunction<? super T> mapper)` | `DoubleStream`
  Transforme en stream de double
- `mapToInt(ToIntFunction<? super T> mapper)` | `IntStream`
  Transforme en stream d'entiers
- `mapToLong(ToLongFunction<? super T> mapper)` | `LongStream`
  Transforme en stream de long

Méthodes terminales - long

- `count()` | `long`
  Compte le nombre d'éléments

Méthodes terminales - Optional

- `findAny()` | `Optional<T>`
  Trouve un élément quelconque (utile pour les streams parallèles)
- `findFirst()` | `Optional<T>`
  Trouve le premier élément
- `max(Comparator<? super T> comparator)` | `Optional<T>`
  Trouve l'élément maximum selon le comparateur
- `min(Comparator<? super T> comparator)` | `Optional<T>`
  Trouve l'élément minimum selon le comparateur
- `reduce(BinaryOperator<T> accumulator)` | `Optional<T>`
  Réduit le stream en une seule valeur par accumulation

Méthodes - T

- `reduce(T identity, BinaryOperator<T> accumulator)` | `T`
  Réduit avec une valeur d'identité (valeur par défaut)

Méthodes - U

- `reduce(U identity, BiFunction<U,? super T,U> accumulator, BinaryOperator<U> combiner)` | `U`
  Réduit avec une valeur d'identité et un combinateur

Méthodes - void

- `forEach(Consumer<? super T> action)` | `void`
  Exécute une action sur chaque élément
- `forEachOrdered(Consumer<? super T> action)` | `void`
  Exécute une action sur chaque élément dans l'ordre

Méthodes - Object[]

- `toArray()` | `Object[]`
  Convertit le stream en tableau d'objets
- `toArray(IntFunction<A[]> generator)` | `A[]`
  Convertit le stream en tableau du type spécifié

Predicate methods

- `and(Predicate<? super T> other)` | `Predicate<T>`
  Combine deux prédicats avec un ET logique
- `isEqual(Object targetRef)` | `Predicate<T>`
  Crée un prédicat qui teste l'égalité avec l'objet donné
- `negate()` | `Predicate<T>`
  Inverse le prédicat (NON logique)
- `or(Predicate<? super T> other)` | `Predicate<T>`
  Combine deux prédicats avec un OU logique
- `test(T t)` | `boolean`
  Teste si l'élément satisfait la condition

Optional methods

Méthodes statiques de création

- `empty()` | `static Optional<T>`
  Crée un Optional vide
- `of(T value)` | `static Optional<T>`
  Crée un Optional avec une valeur (ne doit pas être null)
- `ofNullable(T value)` | `static Optional<T>`
  Crée un Optional, vide si la valeur est null

Méthodes de test de présence

- `isPresent()` | `boolean`
  Vérifie si une valeur est présente
- `isEmpty()` | `boolean`
  Vérifie si l'Optional est vide

Méthodes d'accès à la valeur

- `get()` | `T`
  Récupère la valeur (lance une exception si vide)
- `orElse(T other)` | `T`
  Récupère la valeur ou retourne une valeur par défaut
- `orElseGet(Supplier<? extends T> supplier)` | `T`
  Récupère la valeur ou exécute un fournisseur
- `orElseThrow()` | `T`
  Récupère la valeur ou lance une exception
- `orElseThrow(Supplier<? extends X> exceptionSupplier)` | `T`
  Récupère la valeur ou lance l'exception fournie

Méthodes conditionnelles

- `ifPresent(Consumer<? super T> action)` | `void`
  Exécute une action si la valeur est présente
- `ifPresentOrElse(Consumer<? super T> action, Runnable emptyAction)` | `void`
  Exécute une action si présent, sinon exécute l'action vide

Méthodes de transformation

- `map(Function<? super T,? extends U> mapper)` | `Optional<U>`
  Transforme la valeur si présente
- `flatMap(Function<? super T,? extends Optional<? extends U>> mapper)` | `Optional<U>`
  Transforme et aplatit l'Optional
- `filter(Predicate<? super T> predicate)` | `Optional<T>`
  Filtre la valeur selon un prédicat

Méthodes de combinaison

- `or(Supplier<? extends Optional<? extends T>> supplier)` | `Optional<T>`
  Retourne cet Optional ou l'alternative si vide

Méthodes de conversion

- `stream()` | `Stream<T>`
  Convertit l'Optional en Stream

Méthodes héritées d'Object

- `equals(Object obj)` | `boolean`
  Teste l'égalité avec un autre objet
- `hashCode()` | `int`
  Retourne le code de hachage
- `toString()` | `String`
  Retourne une représentation textuelle