Stream to dynamiczny byt, którego elementy są generowane na bieżąco podczas wykonywania kolejnych operacji. Operacje pośrednie (intermediate) są wykonywane podczas przetwarzania danych, natomiast operacje terminalne kończą operacje na strumieniach, np. zapisują wynik, obliczają średnią lub zapisują dane do nowego zbioru.

Metoda forEach - pozwala na wykonanie określonej operacji na każdym elemencie strumienia.

Operacje ograniczające strumień:

- takeWhile: Pobiera elementy ze strumienia dopóki spełniony jest określony warunek.
- dropWhile: Pomija elementy ze strumienia dopóki spełniony jest określony warunek.
- toList: Zapisuje elementy do listy (dodana w Java 16).
- toArray: Zapisuje elementy do tablicy.
- map: Zamienia elementy ze strumienia jednego typu na elementy innego typu.
- flatMap: Rozprasza elementy ze zagnieżdżonych strumieni.

Metoda collect:

Metoda **collect** służy do zapisywania elementów strumienia do określonej kolekcji, na przykład do listy czy mapy.

Referencja do metody:

Referencja do metody umożliwia skrócenie wyrażeń lambda, wskazując na istniejącą metodę jako implementację interfejsu funkcjonalnego.

Optional:

Optional to klasa w Javie, która może zawierać wartość lub być pusta, co pomaga unikać problemów związanych z wartościami null.

Summarizing:

Operacje `summarizing` pozwalają na uzyskanie podsumowań statystycznych, takich jak suma, średnia, minimum, maksimum, itp.

GroupingBy vs PartitioningBy:

- **groupingBy**: Działa na podstawie klucza i grupuje elementy według określonego kryterium.
- **partitioningBy**: Działa na podstawie predykatu i dzieli elementy na dwie grupy (prawda/fałsz).

Parallel Stream:

Strumienie równoległe (parallel stream) pozwalają na równoczesne przetwarzanie strumieni w wielu wątkach, co może przyspieszyć operacje na dużych zbiorach danych.

D**istinct** - usuwa z strumienia duplikaty.

Najpopularniejsze metody w Stream API:

1. **filter(Predicate<T> predicate):** Pozwala na przefiltrowanie elementów strumienia na podstawie określonego warunku. Zachowuje tylko te elementy, które spełniają warunek określony przez Predicate.

2. **map(Function<T, R> mapper)**: Przekształca każdy element strumienia za pomocą określonej funkcji (mapper). Przykładowo, można zmienić typ elementu lub przekształcić go w inny sposób.

3. **flatMap(Function<T, Stream<R>> mapper)**: Jest używana do rozpraszania elementów zagnieżdżonych strumieni na jeden strumień. Przydatna, gdy mamy strumień z elementami, które są również strumieniami.

4. **forEach(Consumer<T> action)**: Wykonuje określoną operację dla każdego elementu strumienia. Jest to operacja terminalna, co oznacza, że jest to ostatnia operacja wykonywana na strumieniu.

5. **collect(Collector<T, A, R> collector):** Końcowa operacja, która zbiera elementy strumienia do określonej kolekcji (na przykład listy, zbioru, mapy). Umożliwia bardziej zaawansowane operacje agregujące.

6. **reduce(BinaryOperator<T> accumulator):** Redukuje elementy strumienia do jednego wyniku za pomocą określonego operatora binarnego. Przykładowo, można użyć do obliczenia sumy lub znalezienia maksimum.

7. **distinct():** Usuwa duplikaty z strumienia, bazując na implementacji metody equals() dla obiektów. Operacja ta jest również operacją terminalną.

8. **limit(long maxSize)** i **skip(long n):** limit ogranicza liczbę elementów w strumieniu do określonego rozmiaru, natomiast skip pomija określoną liczbę elementów.

9. **sorted() i sorted(Comparator<T> comparator):** Pierwsza forma sortuje elementy strumienia w naturalnym porządku, druga umożliwia zastosowanie niestandardowego porównywania.

10. **anyMatch(Predicate<T> predicate), allMatch(Predicate<T> predicate)** i **noneMatch(Predicate<T> predicate)**: Są to operacje sprawdzające warunki dla elementów strumienia. anyMatch zwróci true, jeśli przynajmniej jeden element spełnia warunek.