

Streams in Java

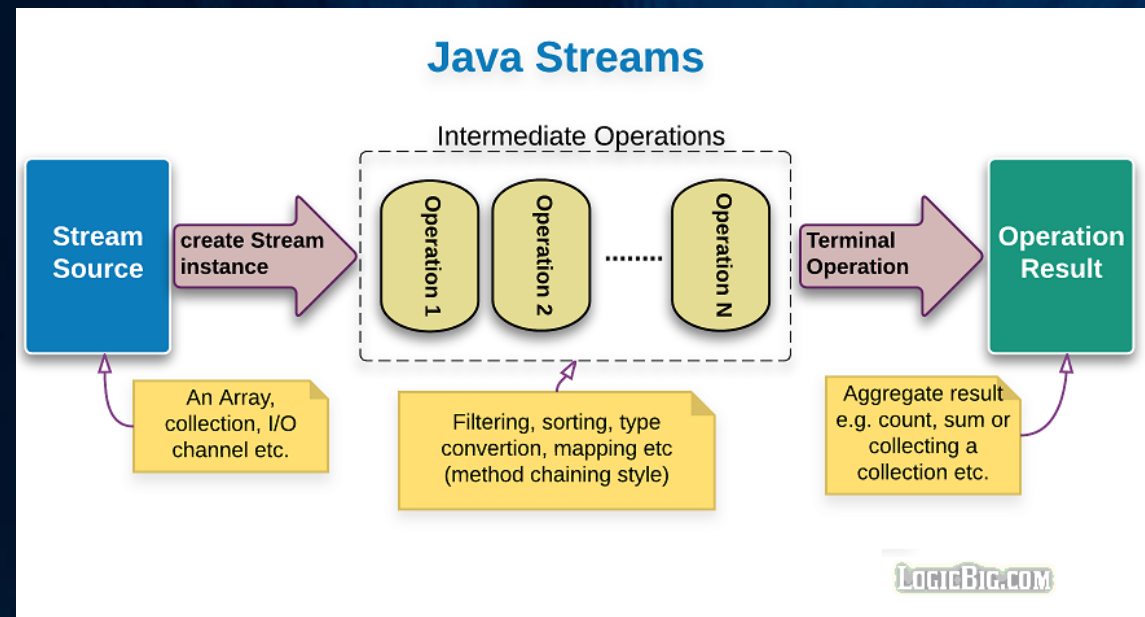
VON LEN UND OLE

Gliederung

- Was sind Streams?
- Der Lambda Ausdruck
- Verwendung von Streams
- Parallele und sequenzielle Streams
- Praktisches Beispiel (Ole)
- stream()-Funktionen
- Praktisches Beispiel (Len)

Was sind Streams?

- Sind mit Java 8 zum `java.util`-Package hinzugekommen
- Stellen Ströme auf Referenzen dar und erlauben den Zugriff auf diese
- Die Daten, die durch die Referenzen repräsentiert werden, werden durch den Stream selbst nicht verändert

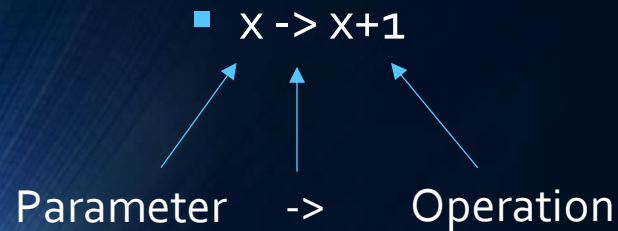


Der Lambda Ausdruck

- Der Lambda Ausdruck stellt eine „quasi-Methode“ ohne Namen dar
- Es wird kein Rückgabebetyp benötigt
 - Dieser wird vom Compiler bestimmt
- Wie schauts aus?

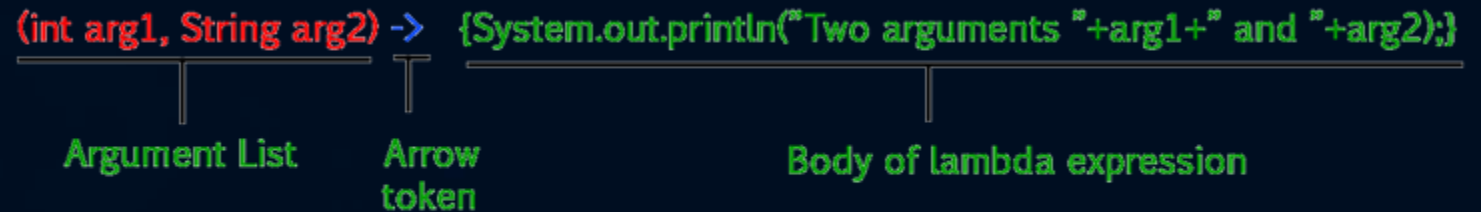
■ $x \rightarrow x+1$

Parameter \rightarrow Operation



`(int arg1, String arg2) -> {System.out.println("Two arguments "+arg1+" and "+arg2);}`

Argument List Arrow token Body of lambda expression



Die Verwendung von Streams

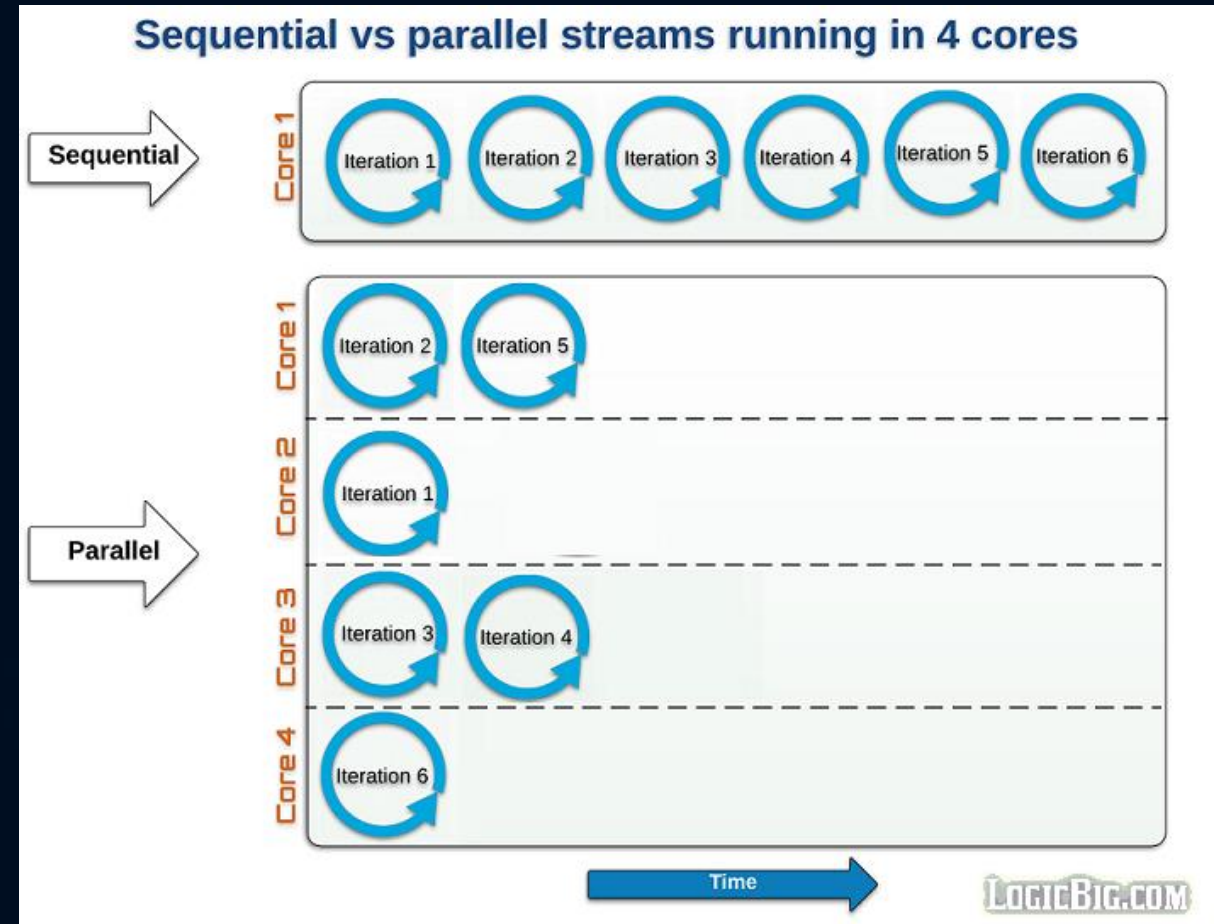
- Streams sollen den Quellcode lesbarer und kürzer machen
- Stellen eine alternative zur klassischen for-Schleife dar um Zugriffe auf eine Collection zu ermöglichen

```
for(Person person : personList){  
    if(person.age > 20){  
        over20Club.add(person);  
    }  
}
```

```
personList.stream()  
    .filter(person -> person.age > 20)  
    .forEach(person -> System.out.println(person.name));
```

Parallele und sequenzielle Streams

- Sequenzielle Streams werden auf einem Thread bearbeitet
- Parallele Streams hingegen wird der Stream auf so vielen Threads bearbeitet wie es CPU-Kerne gibt
 - Geeignet wenn sehr große Datenmengen bearbeitet werden



stream()-Funktionen

- Mithilfe von Streams kann man Listen filtern und sortieren:

- Filtern

- `List<Objekt> listenname = liste.stream()
 .filter(liste -> liste.getEigenschaft().equals(filterEigenschaft))
 .collect(Collectors.toList());`

- Sortieren

- `List<Objekt> listenname = ausgangsliste.stream()
 .sorted(Comparator.comparing(Objekt::getEigenschaft))
 .collect(Collectors.toList());`

- Man kann eine Liste auch mehrfach nach unterschiedlichen Eigenschaften sortieren, indem man `.thenComparing(Objekt::getZweiteEigenschaft)` hinter die erste `comparing()`-Funktion anfügt

- Die Sortierreihenfolge ist durch ein `.reversed()` nach der `sorted()`-Funktion umdrehbar

stream()-Funktionen

- Es lassen sich durch Streams auch Informationen der Liste entnehmen
 - Wenn man prüfen möchte ob Elemente der Liste Eigenschaften erfüllen gibt es die:
 - "All match"-Funktion
 - Durch diese Funktion wird geprüft ob alle Elemente der Liste eine Eigenschaft erfüllen
 - `Boolean allMatch = liste.stream().allMatch(liste -> VERGLEICH);`
 - "Any match"-Funktion
 - Durch diese Funktion wird geprüft es ein Element in der Liste gibt, welche Eigenschaften erfüllen
 - `Boolean anyMatch = liste.stream().anyMatch(liste -> VERGLEICH);`
 - "None match"-Funktion
 - Durch diese Funktion wird geprüft ob alle Elemente der Liste keine Eigenschaft erfüllen
 - `Boolean noneMatch = liste.stream().noneMatch(liste -> VERGLEICH);`
 - Bei diesen Funktionen wird ein Boolean-Wert zurückgegeben

stream()-Funktionen

- Außerdem lassen sich Maximal- und Minimalwerte ausgeben:
 - Max"-Funktion
 - `people.stream()`
 `.max(Comparator.comparing(Liste::getEigenschaft))`
 `.ifPresent(System.out::println);`
 - „Min"-Funktion
 - `people.stream()`
 `.min(Comparator.comparing(Liste::getEigenschaft))`
 `.ifPresent(System.out::println);`
- Bei diesen Funktionen wird das Element der Liste mit dem höchsten/niedrigsten Wert der gegebenen Eigenschaft zurückgegeben

Quellen

- <https://www.geeksforgeeks.org/stream-map-java-examples/>
- <https://entwickler.de/java/was-sind-streams>
- <https://www.baeldung.com/java-8-streams>
- https://www.youtube.com/watch?v=Q93JsQ8vcwY&ab_channel=Amigoscode
- <https://ertan-toker.de/java-streams-tutorial-and-examples/>
- <https://www.youtube.com/watch?v=dQw4wgWgXcQ>