

JAVA 8 STREAMS

Deklarative und Parallele Arbeit auf Listen

-

Carlo Kurth

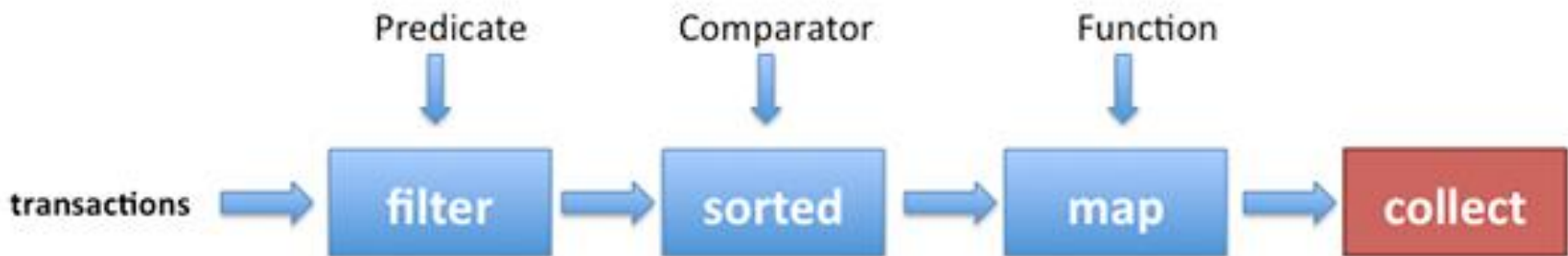
Bob Prevos

Der Stream im Allgemeinen

- Eingeführt in Java 8
- Neue Abstrakte Klasse `java.util.stream`
- Intention:
 - Arbeite auf großen Listen deklarativ wie bei SQL
 - Nutze dabei die Multicore Architektur aus
 - Mach es simpel und einfach zu implementieren

Die Pipeline

- Streams ermöglichen es aus Listen eine Pipeline zu formen



Beispiel

- Gegeben:
Liste von Transactions mit verschiedenen Typen und eindeutiger ID-Nummer
- Gesucht:
Liste der ID-Nummern der Transactions vom Typ Grocery, abfallend sortiert nach dem value der Transaction

Beispiel – ohne Stream

```
List<Transaction> groceryTransactions = new ArrayList<>();
for(Transaction t: transactions){
    if(t.getType() == Transaction.GROCERY){
        groceryTransactions.add(t);
    }
}
Collections.sort(groceryTransactions, new Comparator(){
    public int compare(Transaction t1, Transaction t2){
        return t2.getValue().compareTo(t1.getValue());
    }
});
List<Integer> transactionIds = new ArrayList<>();
for(Transaction t: groceryTransactions){
    transactionIds.add(t.getId());
}
```

Beispiel – mit Stream

```
List<Integer> transactionsIds =  
    transactions.stream()  
        .filter(t -> t.getType() == Transaction.GROCERY)  
        .sorted(comparing(Transaction::getValue).reversed())  
        .map(Transaction::getId)  
        .collect(toList());
```



Beispiel – mit Parallelem Stream

```
List<Integer> transactionsIds =  
    transactions.parallelStream()  
        .filter(t -> t.getType() == Transaction.GROCERY)  
        .sorted(comparing(Transaction::getValue).reversed())  
        .map(Transaction::getId)  
        .collect(toList());
```

Parallele Streams

- Aufteilung der Bearbeitung der Liste auf Threads
- Standardmäßig 1 Thread pro Core des Systems
- Automatische Threaderstellung und -zusammenführung

Parallele Streams – Tücken

- Collections wie ArrayList sind nicht Threadsafe:
 -  Die Datenquelle des Streams darf nicht von den Aktionen in der Pipeline modifiziert werden
- Alle Parallel Streams nutzen denselben fork-join thread pool:
 -  Ein fehlerhafter blockierender Thread kann Threads stören, die eigentlich unabhängig sein sollten

Das Barbershop-Problem

- Klassisches Problem zur Verdeutlichung der Mechanismen eines Betriebssystems
- Erstmals vorgeschlagen von Dijkstra

Hier eine Variation des Problems aus dem Buch „The Little Book of Semaphores“ von Allen B. Downey

Das Barbershop-Problem

- Der Barbershop setzt sich zusammen aus:
 - dem Warteraum mit n Stühlen
 - dem Barberroom mit m Stühlen
- Gibt es keinen freien Stuhl für einen Kunden, wartet der Kunde stehend
- Sind die Barber beschäftigt, aber Stühle frei, setzt sich der Kunde und wartet
- Der Barber setzt sich regelmäßig in seinen Stuhl und schläft