

OMO

9 - Map/filter/reduce

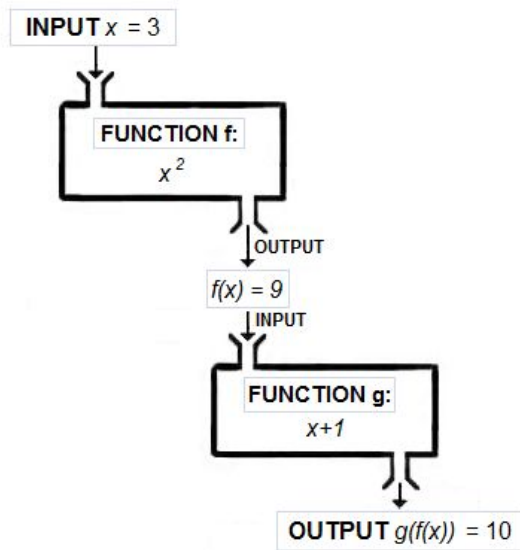
Ing. David Kadleček, PhD

kadlecd@fel.cvut.cz, david.kadlecek@cz.ibm.com

Map/filter/reduce v Java

Map/filter/reduce v Java = Java 1.8 streams API

Funkcionální přístup (řetězíme operace do sebe) + **Abstrakce kontrolního flow**



Cykly (for, while ...) - řešíme jak iterovat, iterujeme, počítáme



Iterátory - iterujeme a počítám my

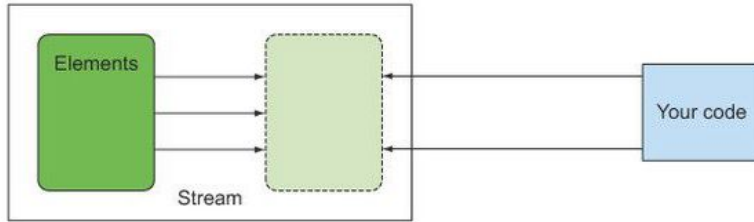


Map/filter/reduce - nemusíme dělat nic - iteruje a počítá samo

Map/filter/reduce v Java

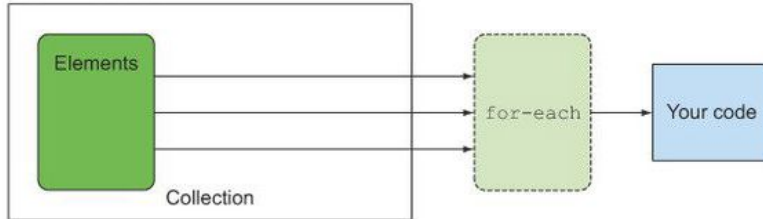
Stream

Internal iteration

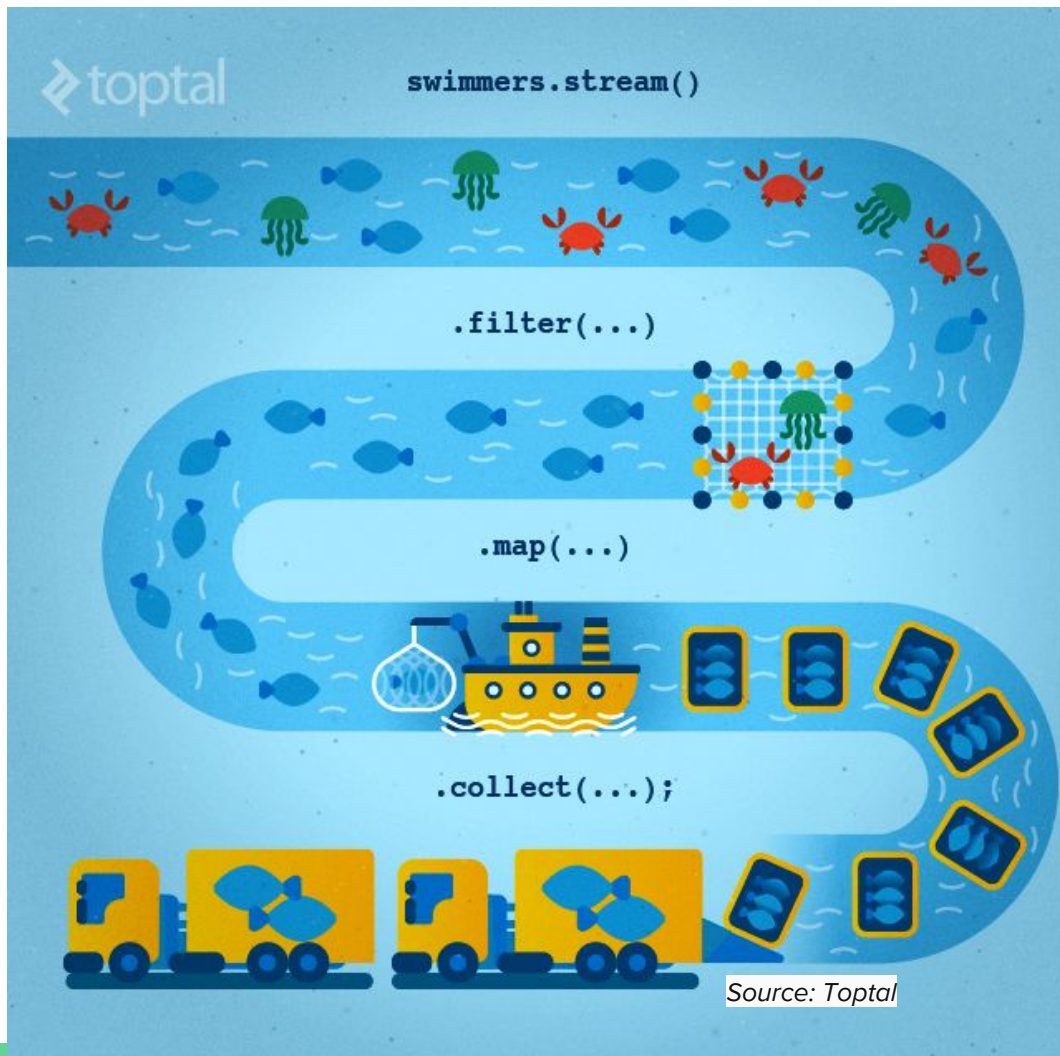


Collection

External iteration



Map/filter/reduce v Java



Jak vytvořit stream

Stream lze získat z libovolné kolekce:

```
Collection<String> names = Arrays.asList("John", "David", "Martin");  
Stream<String> streamingNames = names.stream();
```

Stream lze vytvořit z posloupnosti pevně daných prvků:

```
Stream<String> streamingNames = Stream.of("John", "David", "Martin")
```

Další možností definice streamu je **indukce**. Stream je zadán prvním prvkem a unární operací, která je aplikována na poslední vytvořený prvek za účelem vytvoření prvku následujícího.

```
Stream<Boolean> streamingBooleans = Stream.iterate(false, i -> !i);
```

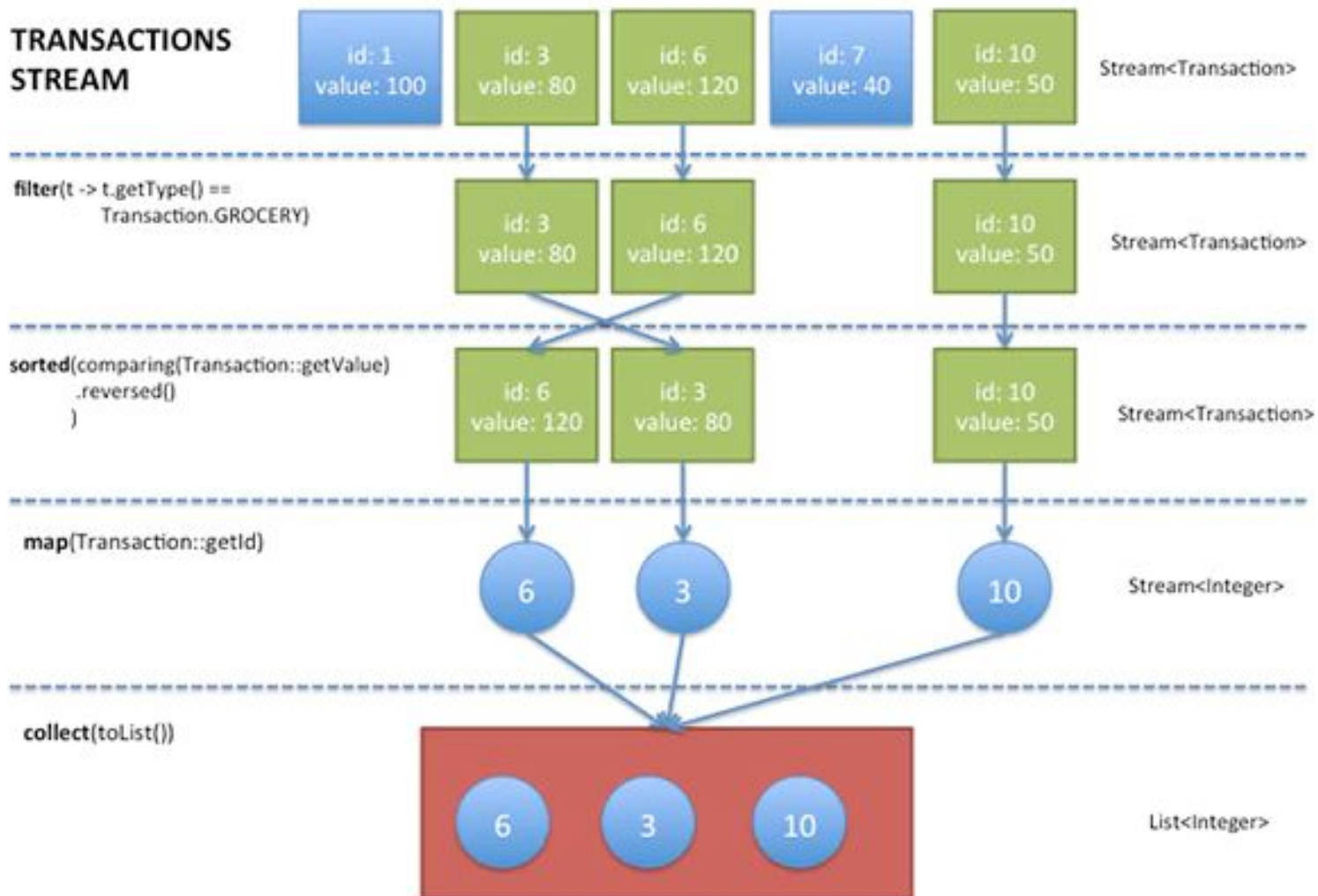
Příklad map/filter/reduce

Předpokládejme, že existuje třída Person s atributy name (jméno), age (věk), city (město). Dále mějme kolekci nějakých lidí uloženou v kolekci people.

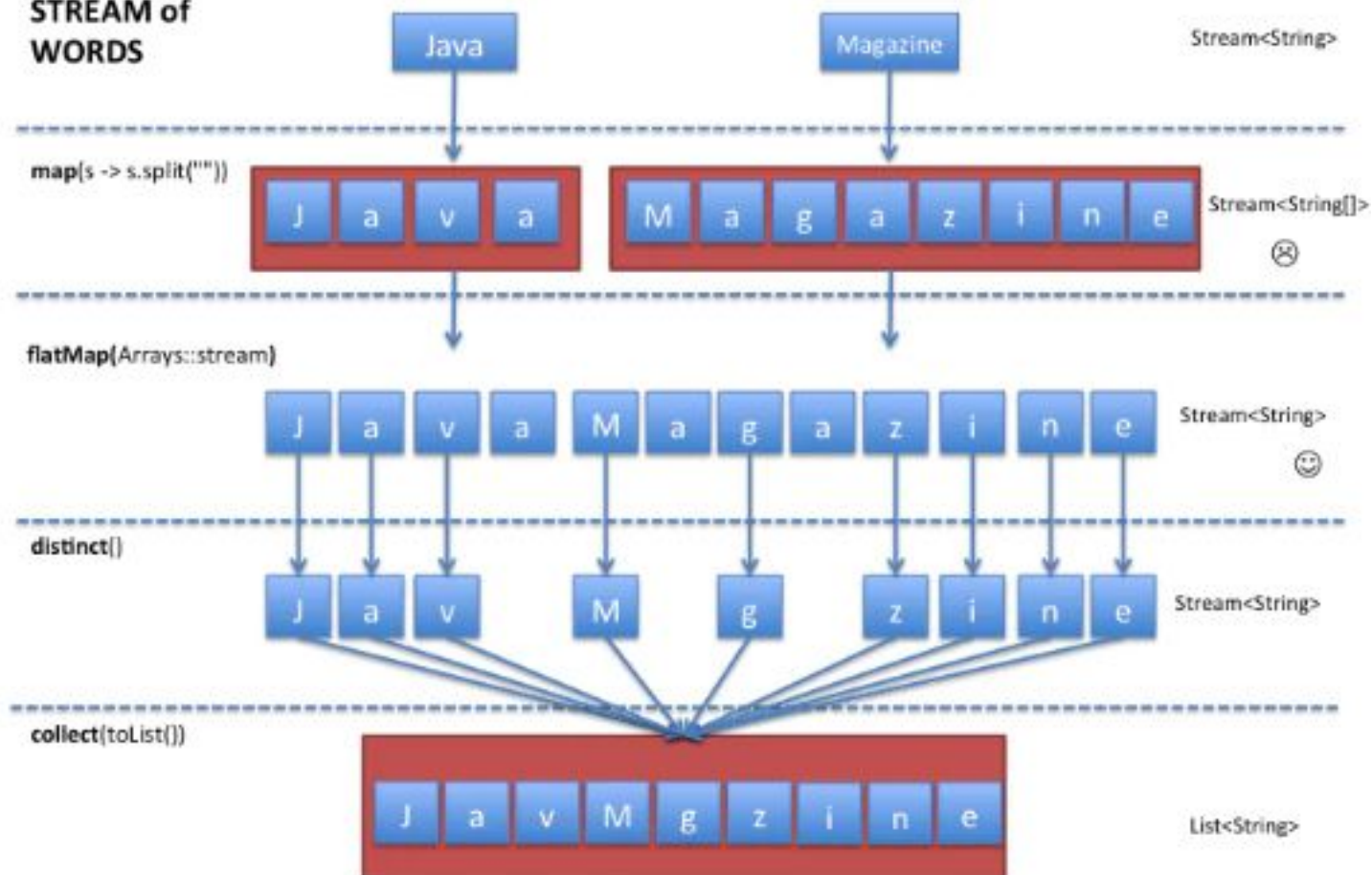
Chceme najít jméno nejstaršího člověka z Washingtonu:

```
String name = people
    // převést na stream
    .stream()
    // dál propustit pouze lidi z Washingtonu
    .filter(p -> p.getCity().equals("Washington"))
    // seřadit podle věku sestupně
    .sorted(Comparator.comparingInt((Person p) -> p.getAge()).reversed())
    // najít první takovou osobu
    .findFirst()
    .get()
    // získat jméno osoby
    .getName();
```

TRANSACTIONS STREAM



STREAM of WORDS

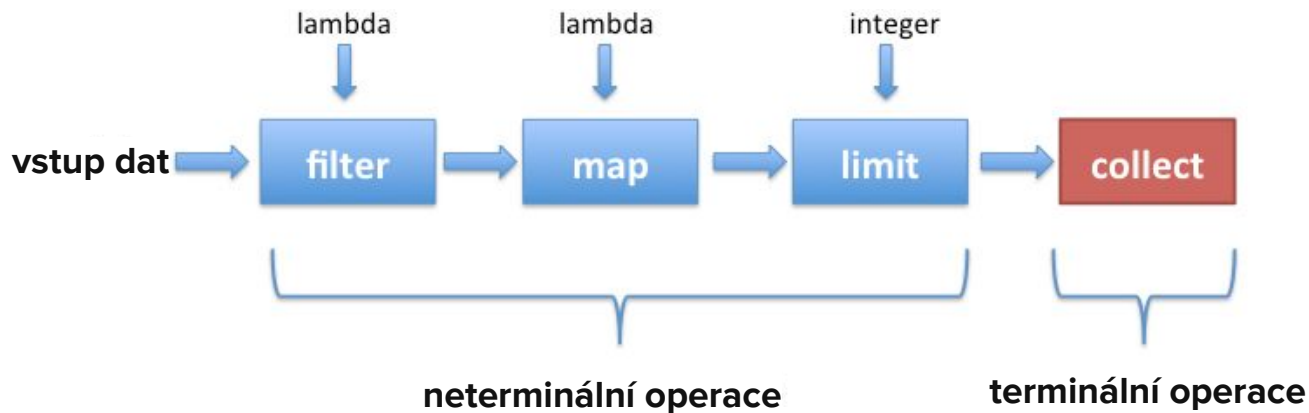


Pipeline

Použitím streamu vzniká tzv. **pipeline**, což je konečná posloupnost operací aplikovaná na nějaký stream.

Operace se dělí do skupin:

- **terminální operace** (terminal) - má postranní efekt nebo produkuje hodnotu; po spuštění této operace je proud uzavřen a nelze jej použít (např. **forEach**)
- **neterminální operace** (intermediate) - nemá žádné postranní efekty; vytváří vždy nový stream
 - stavové (stateful) - stav operace je ovlivněn procházejícími prvky (např. **sorted**)
 - bezstavové (stateless) - operace nemá žádný vnitřní stav (např. **filter**)



Map/filter/reduce operace

Mějme abstraktní datový typ `Seq<E>` reprezentující *sekvenci* elementů typu `E`.

Např., `[1, 2, 3, 4] ∈ Seq<Integer>` .

Map aplikuje unární funkci na každý element sekvence a vrací novou sekvenci výsledků ve stejném pořadí:

map : $(E \rightarrow F) \times \text{Seq}\langle E \rangle \rightarrow \text{Seq}\langle F \rangle$

Filtr testuje každý element unárním predikátem. Elementy, které vyhovují jsou ponechána a ostatní odstraněny. Vracen je nový list.

filter : $(E \rightarrow \text{boolean}) \times \text{Seq}\langle E \rangle \rightarrow \text{Seq}\langle E \rangle$

Map/filter/reduce operace

Reduce kombinuje elementy sekvence dohromady s pomocí binární operace. Dále bere v potaz inicializační hodnotu, kterou inicializuje redukci nebo vrátí zpět u prázdné sekvence.

reduce : $(F \times E \rightarrow F) \times \text{Seq}\langle E \rangle \times F \rightarrow F$

reduce(f, list, init) kombinuje elementy listu zleva doprava:

result 0 = init

result 1 = f(result 0, list[0])

result 2 = f(result 1, list[1])

...

result n = f(result n-1, list[n-1])

Sečtení kolekce čísel:

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6);  
Integer sum = numbers.stream().reduce(0, (a,b)->a+b);
```

Příklady operací

Operace	Popis	Druh
filter	filtruje prvky podle zadaného predikátu	bezstavová-NT
map	převádí prvky na jiné pomocí zadaného zobrazení	bezstavová-NT
flatMap	Jako map, ale vnořené kolekce převede do jediné kolekce	bezstavová-NT
groupBy	Seskupení podle vybraného atributu	bezstavová-NT
reduce	generická operace redukce	terminální
max	konkrétní operace redukce	terminální
limit	omezí maximální délku streamu na zadaný počet prvků	terminální
forEach	všechny prvky streamu postupně odešle konzumentovi	terminální
sorted	prvky v streamu budou do dalších operací předávány seřazené	stavová-NT
findFirst	vezme první prvek z streamu	bezstavová-NT
count	spočítá prvky v streamu	bezstavová-NT

Další příklady map/filter/reduce

Průměrný věk

```
double avgAge = people
    // převést na proud
    .stream()
    // od osoby získat její věk
    .mapToInt(p -> p.getAge())
    // z čísel vypočítat průměr
    .average()
    .getAsDouble();
```

Setřídění

```
List<String>
myList=Arrays.asList("a1","a2","b1","c2","c1");
myList
    .stream()
    .filter(s -> s.startsWith("c"))
    .map(String::toUpperCase)
    .sorted()
    .forEach(System.out::println);
```

OUTPUT:

C1

C2

Kolektory

Kolektor je třída, která agreguje prvky ze streamu. Existují kolektory, které prvky jednoduše uloží do seznamu, provedou seskupení podle nějakého kritéria a tyto skupiny uloží do mapy, a podobně. Lze vytvářet i vlastní kolektory.

```
// sesbírá jména lidí do seznamu
```

```
List<String> list = people.stream().map(Person::getName).collect(Collectors.toList());
```

```
// seskupí osoby do mapy podle města, kde žijí
```

```
Map<String, List<Person>> personByCity = people  
    .stream()  
    .collect(Collectors.groupingBy(Person::getCity));
```

```
// do mapy uloží počet osob v každém městě
```

```
Map<String, Integer> peoplePerCity = people  
    .stream()  
    .collect(Collectors.groupingBy(Person::getCity, Collectors.counting()));
```

Performance pohled

Stream může být nastaven jako **paralelní**

```
Arrays.asList("John", "David", "Martin").parallelStream()
```

Neterminální operace (intermediate) jsou **lazy**

```
//Created a list of students
Stream<String> streamOfNames = students.stream()
    .map(student -> {
        System.out.println("In Map - " + student.getName());
        return student.getName();
    });
//Just to add some delay
for (int i = 1; i <= 5; i++) {
    Thread.sleep(1000);
    System.out.println(i + " sec");
}
//Called a terminal operation on the stream
streamOfNames.collect(Collectors.toList());
```

Operace se na streamu začnou provádět až ve chvíli, kdy potřebujeme výstup



OUTPUT:

1 sec

2 sec

3 sec

4 sec

5 sec

In Map - Tom

In Map - Chris

In Map - Dave


```
List<String> ids = students.stream()
    .filter(s -> {System.out.println("filter - "+s); return s.getAge() > 20;})
    .map(s -> {System.out.println("map - "+s); return s.getName();})
    .limit(3)
    .collect(Collectors.toList());
```

Prvky pokud možno proplouvají pipeline po jedné od vstupu až na výstup. *id - 8* prvního studenta prochází filtrem a okamžitě pokračuje do mapy. Pak *id - 9*, pak *id - 10* (neprošlo filtrem) atd.



```
OUTPUT:
filter - 8
map - 8
filter - 9
map - 9
filter - 10
filter - 11
map - 11
```