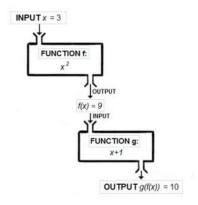
# Map, filter, reduce pattern v Java

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
```

- Stream = interní iterace X Collection = externí iterace
- 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:

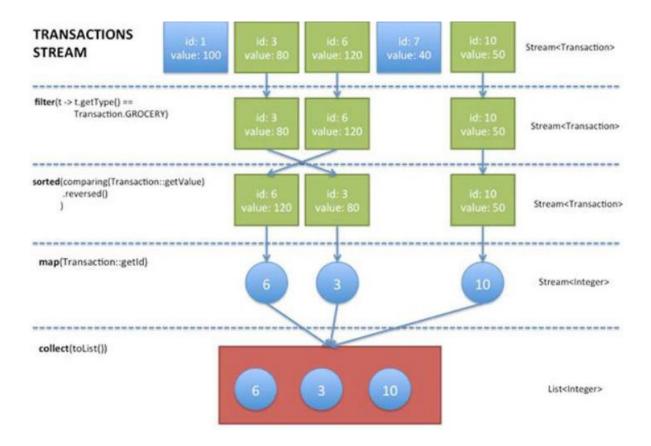
```
Stream<String> streamingNames = Stream.of("John", "David", "Martin")
```

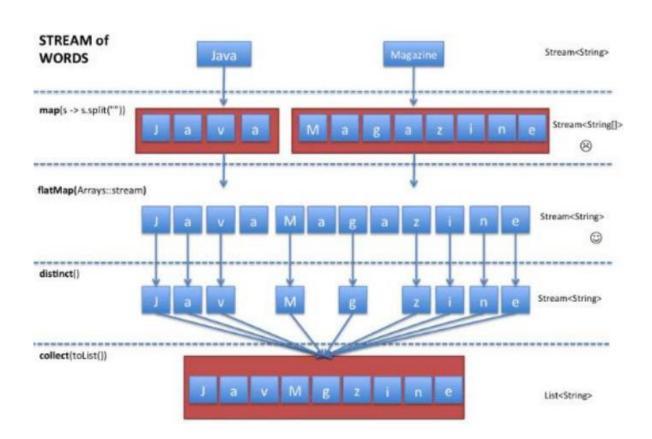
- případně pomocí **indukce** – stream zadán prvním prvkem a unární operací, která je aplikována na poslední vytvořený prvek (za účelem vytvoření prvku následujícího)

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

- příklad:
  - o třída Person s atributy name, age, city
  - o kolekce nějakých lidí uložená 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();
```



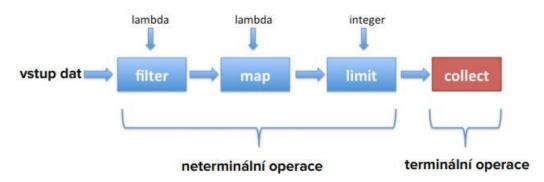


### M/F/R operace

- **pipeline** = konečná posloupnost operací aplikovaná na nějaký stream
- operace se dělí na:
  - o terminální (terminal)
    - má postranní efekt nebo produkuje hodnotu
    - po spuštění je proud uzavřen a nelze jej použít
    - např. forEach

#### o neterminální (intermediate)

- nemá žádné postranní efekty
- vytváří vždy nový stream
- dále se dělí na:
  - stavové (stateful) stav operace ovlivněn procházejícími prvky (např. sorted)
  - **bezestavové (stataless)** operace nemá žádný vnitřní stav (např. filter)



- mějme abstraktní datový typ Seq<E> (sekvence elementů typu E)
  - o např. [1,2,3,4] ∈ Seq<Integer>

#### - Map:

- o aplikuje unární funkci na každý element sekvence
- o vrací novou sekvenci výsledků ve stejném pořadí
- o map:  $(E \rightarrow F) \times Seq \rightarrow Seq$

#### - Filter:

- o testuje každý element unárním predikátem
- o elementy, které vyhovují, jsou ponechány (ostatní odstraněny)
- filter: (E  $\rightarrow$  boolean) × Seq  $\rightarrow$  Seq

#### - Reduce:

- o kombinuje elementy sekvence dohromady s pomocí binární operace
- o bere v potaz inicializační hodnotu (in. redukci)
- o reduce:  $(F \times E \rightarrow F) \times Seq \times F \rightarrow F$  ... reduce(f, list, init)

#### Sečtení kolekce čísel:

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

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

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

```
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);

C1
C2
    ... setřídění
```

# Kolektory

- třídy, které agregují prvky ze streamu
- existují kolektory, které prvky jednoduše uloží do seznamu, provedou seskupení dle nějakého kritéria a tyto skupiny uloží do mapy,...
- lze vytvářet i vlastní kolektory

```
// sesbirá 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 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);
                                                          OUTPUT:
   System.out.println(i + " sec");
                                                          1 sec
//Called a terminal operation on the stream
                                                          2 sec
streamOfNames.collect(Collectors.toList());
                                                          3 sec
                                                          4 sec
      Operace se na streamu začnou provádět až ve
                                                          5 sec
      chvíli, kdy potřebujeme výstup
                                                          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 pipelinou 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.