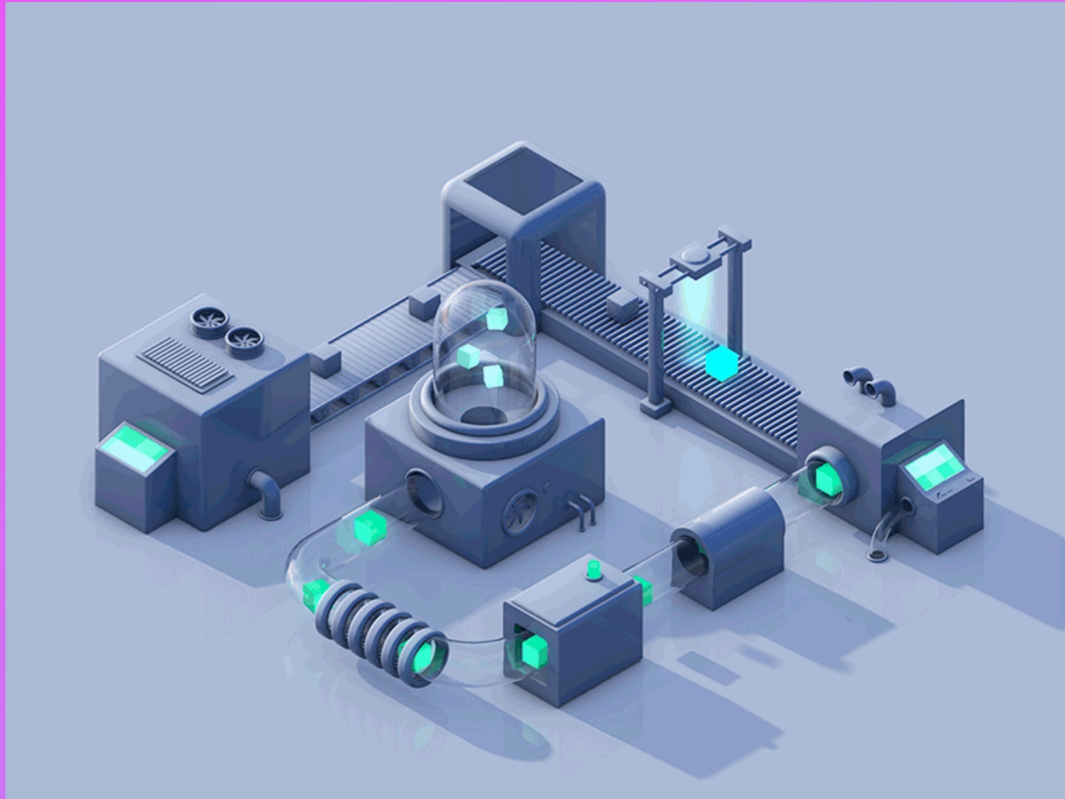
czm

Search . . .



09

Java Streams

- Abstrakce Control Flow
- Map
- Filter
- Reduce
- Collect
- Paralelizace map reduce a performance pohled

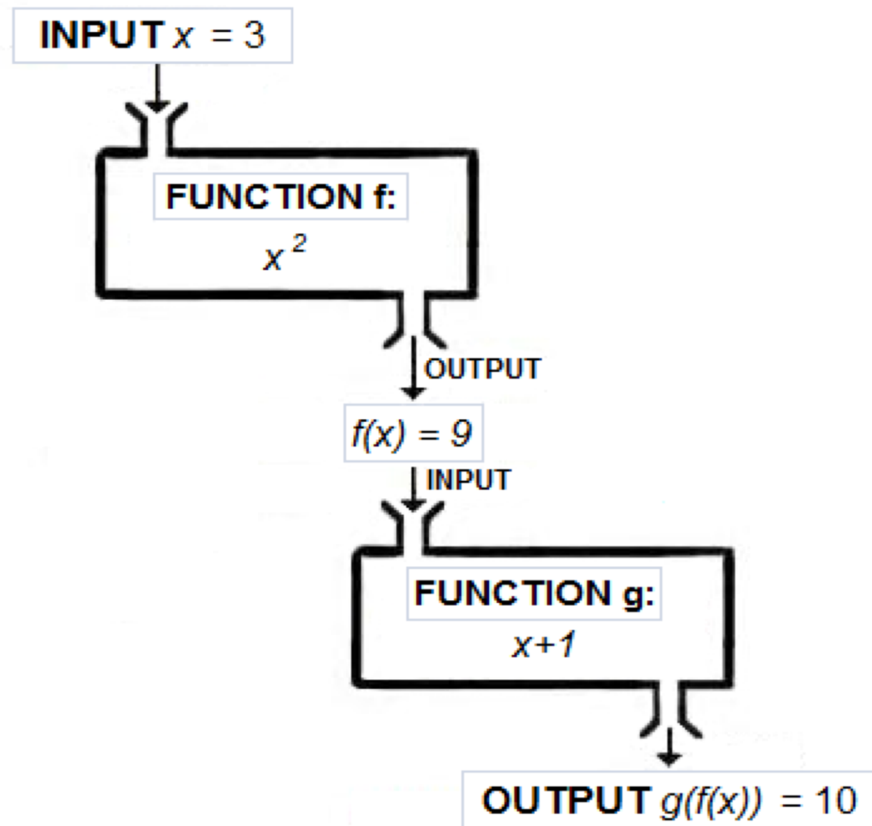
Next >

Objektové Modelování - podzim 2022

kadlecd@fel.cvut.cz, www.czm.fel.cvut.cz

09 Map/filter/reduce v Java

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

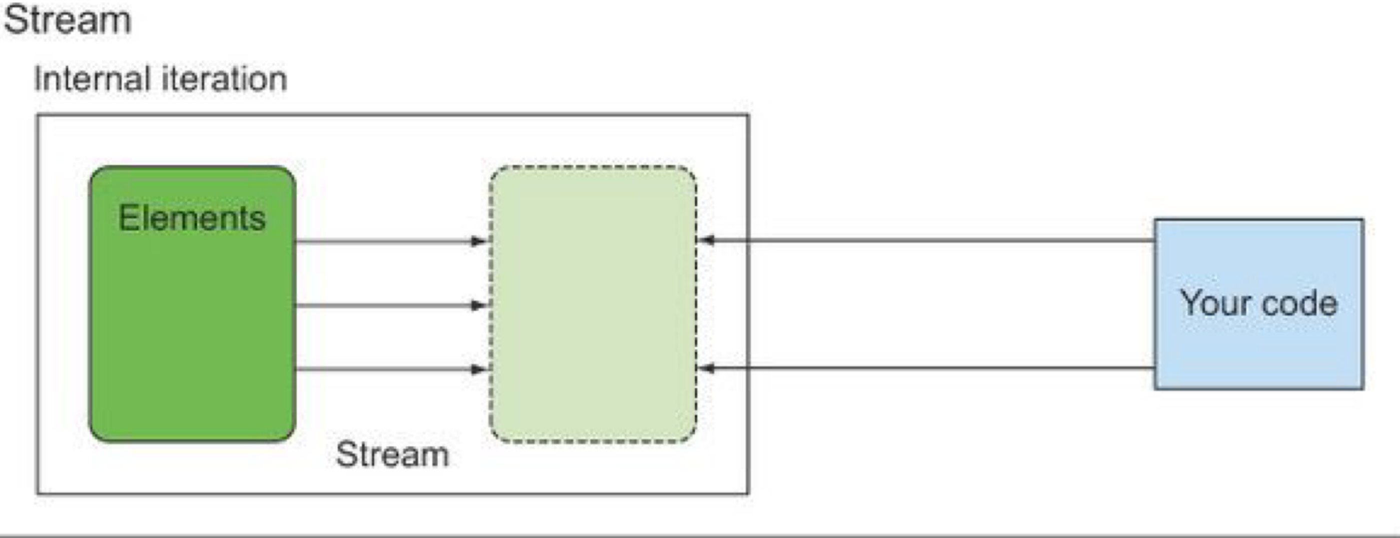


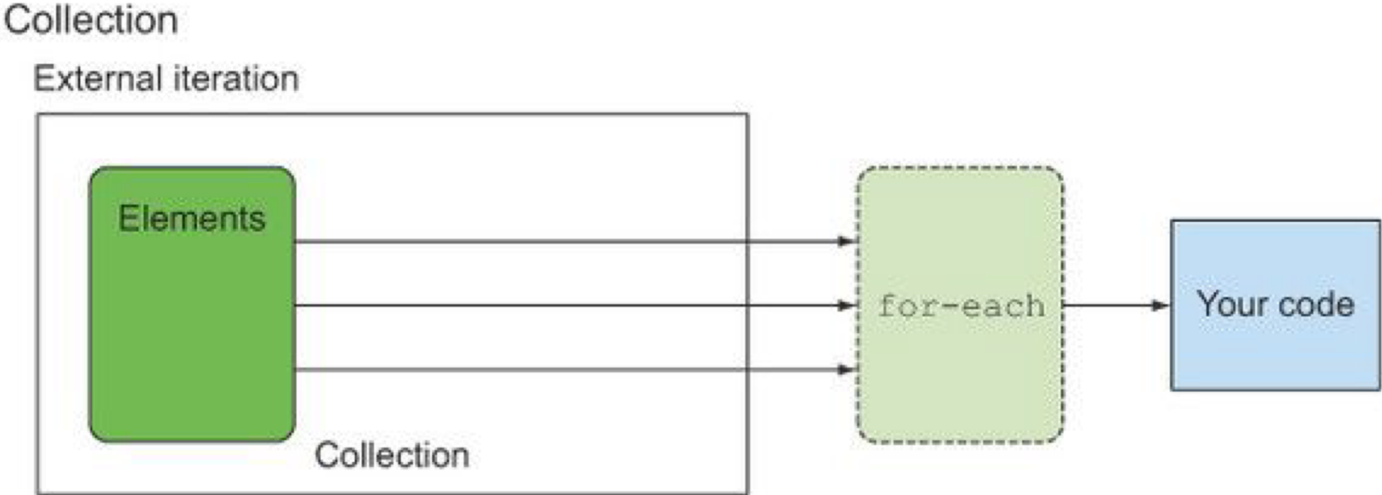
Cykly (for, while …) - řešíme jak iterovat, iterujeme, počítáme

Iterátory - iterujeme a počítám my

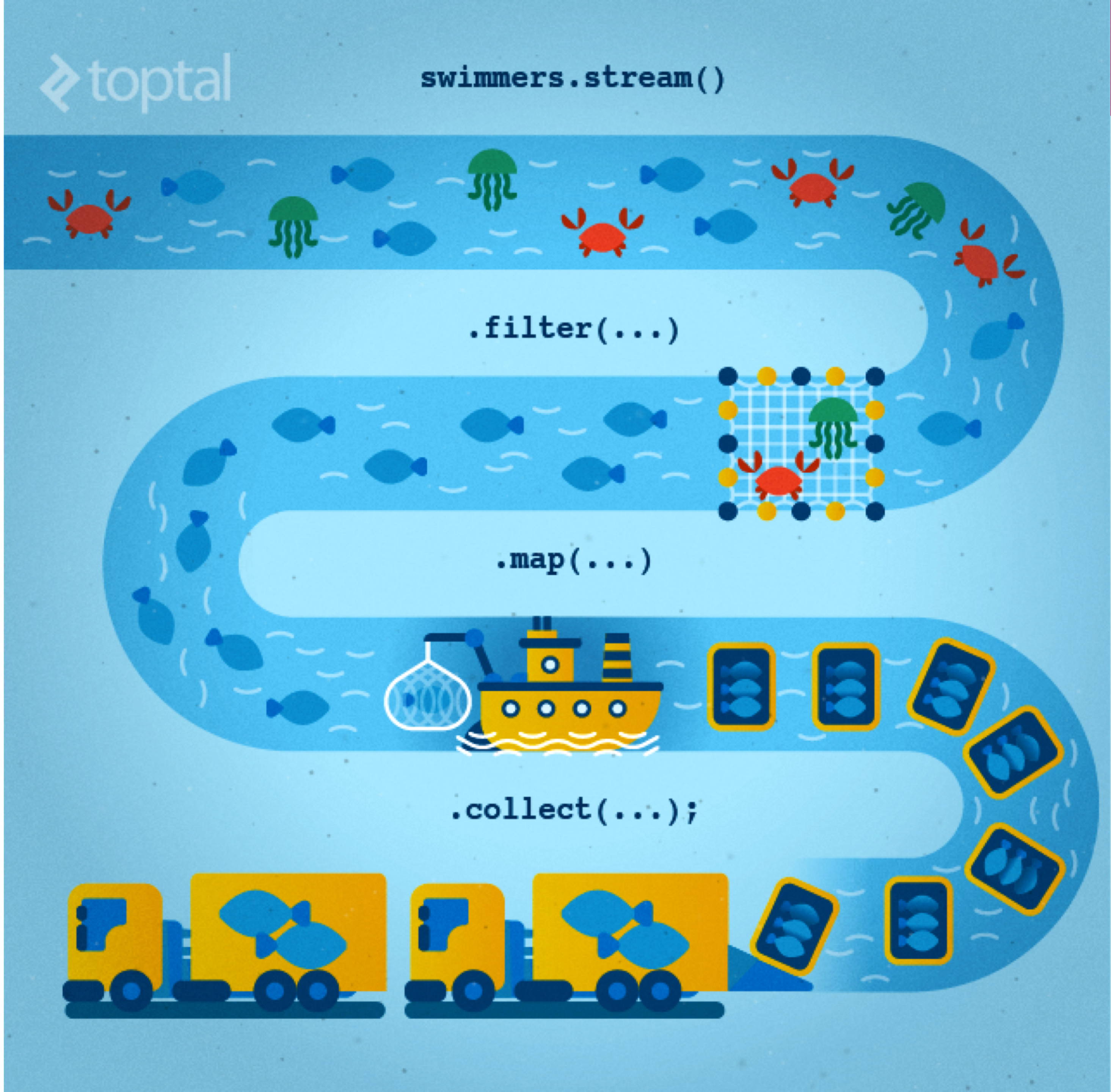
Map/filter/reduce - nemusíme dělat nic - iteruje a počítá samo

09 Map/filter/reduce v Java





09 Map/filter/reduce v Java

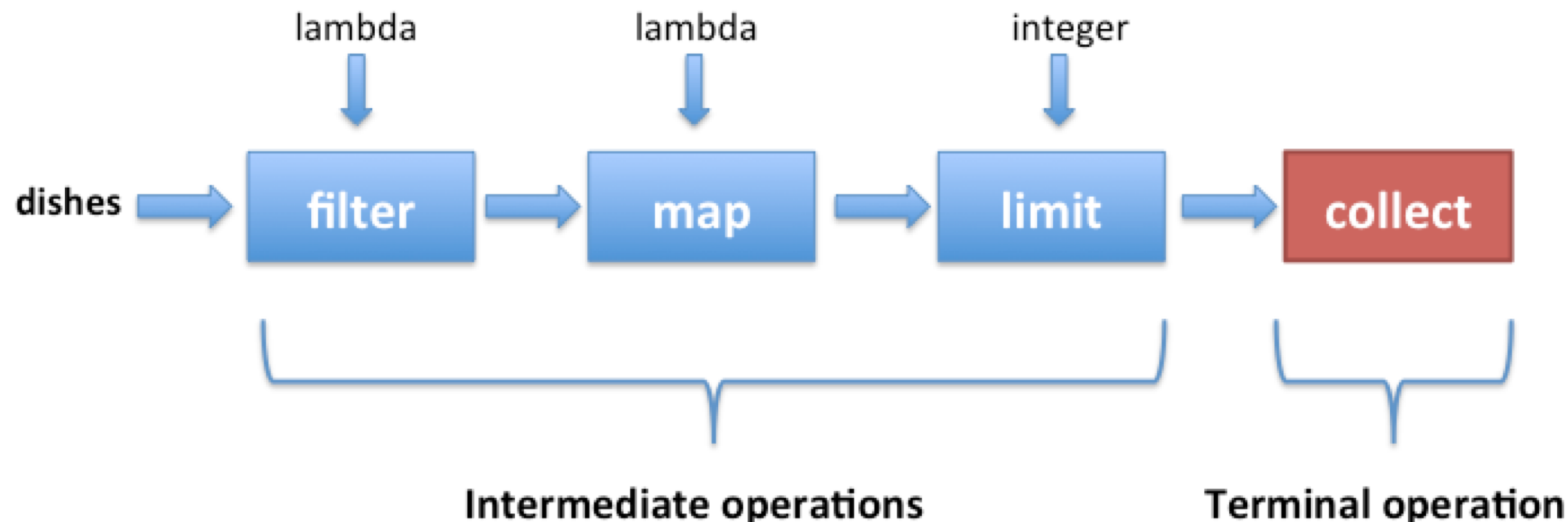


09 Map/filter/reduce v Java

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) - operace ovlivňuje další prvky (např. sorted)
  - bezstavové (stateless) - operace neovlivňuje další prvky (např. filter)



09 Map/filter/reduce v Java

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);
```

09 Map/filter/reduce v Java

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();
```

09 Map/filter/reduce v Java

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 → F) × Seq<E> → Seq<F>

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

filter : (E → boolean) × Seq<E> → Seq<E>

09 Map/filter/reduce v Java

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 × E → F) × Seq<E> × F → F

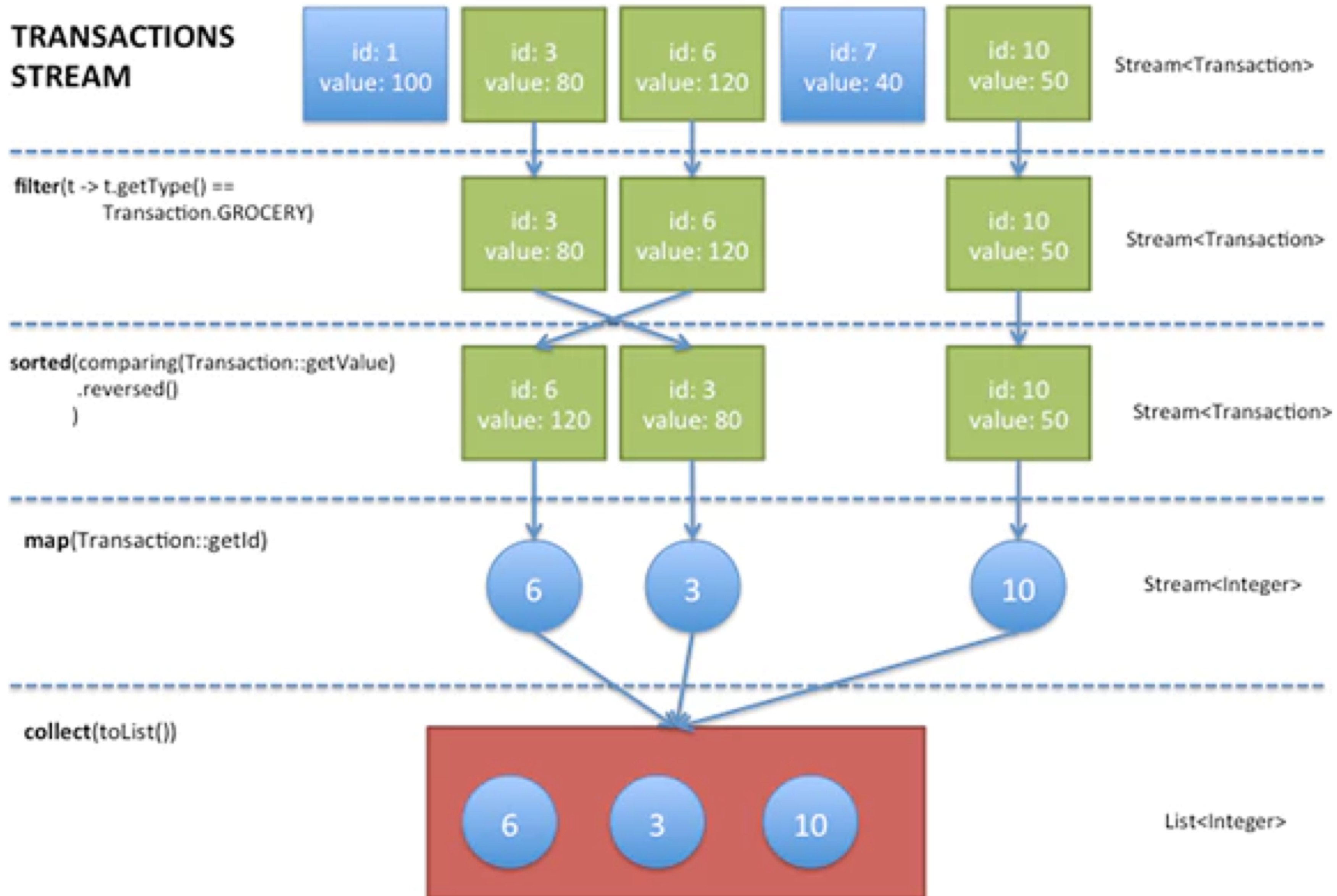
`reduce(f, list, init)` kombinuje elementy listu zleva doprava:

$$\begin{aligned}
&result\ 0 = init \\
&result\ 1 = f(result\ 0\ ,\ list[0]) \\
&result\ 2 = f(result\ 1\ ,\ list[1]) \\
&\ldots \\
&result\ n = f(result\ n\text{-}1\ ,\ list[n\text{-}1])
\end{aligned}$$

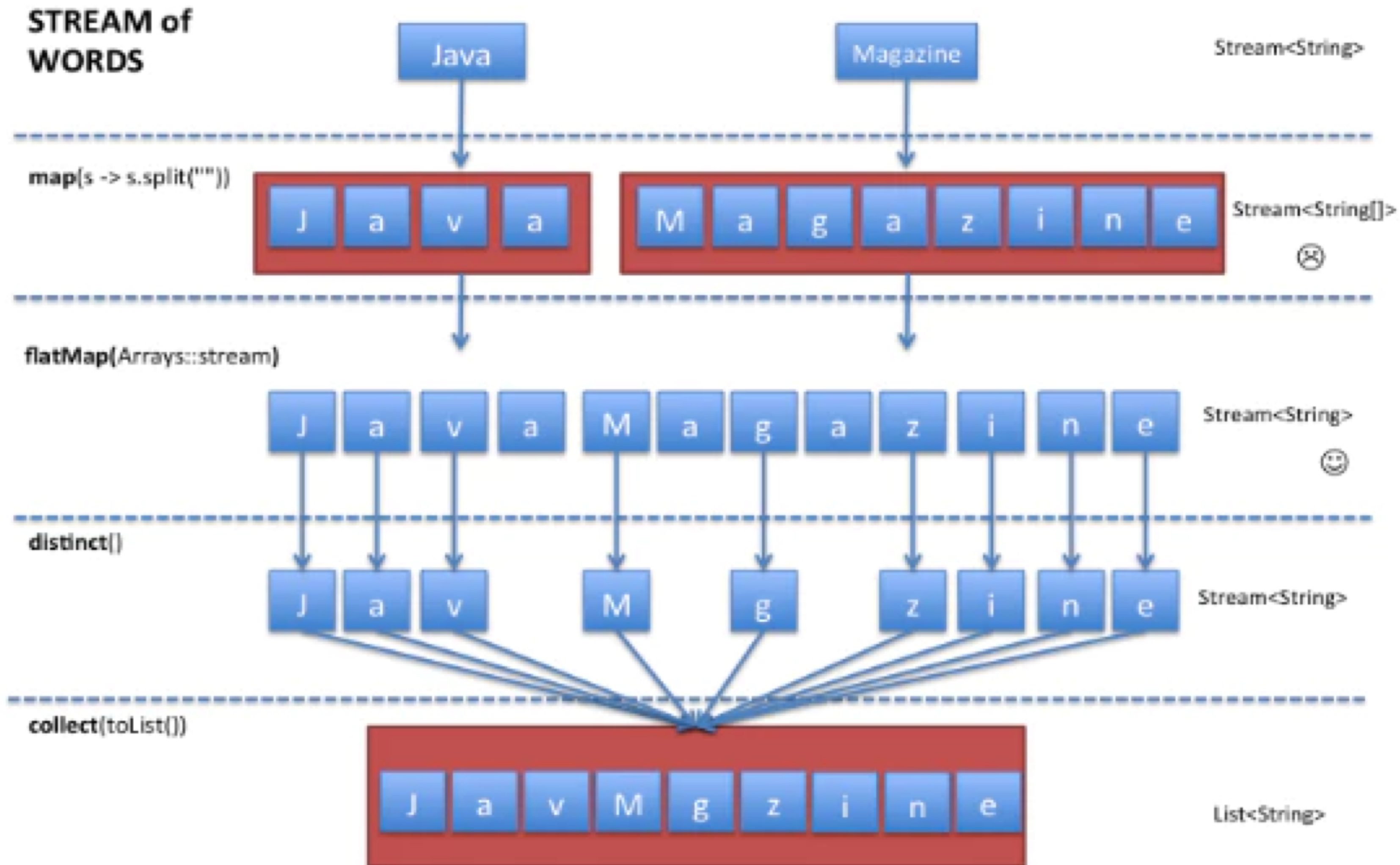
Sečtení kolekce čísel:

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

09 Map/filter/reduce v Java



09 Map/filter/reduce v Java



09 PŘÍKLADY OPERACÍ

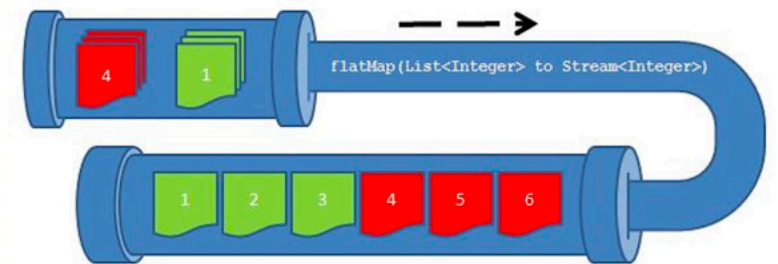
Operace	Popis	Druh
filter	filtruje prvky podle zadaného predikátu	bezstavová-NT
map	převádí prvky na jiné pomocí zadaného zobrazení	bezstavová-NT
groupBy	Seskupení podle vybraného atributu	bezstavová-NT
reduce	generická operace redukce	terminální
max	konkrétní operace redukce	terminální
limit	omezí maximální délku streamu na zadaný počet prvků	terminální
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

09 PŘÍKLADY OPERACÍ

Operace	Popis	Druh
flatMap	Jako map, ale vnořené kolekce převede do jediné kolekce	bezstavová-NT

[[2, 3, 5], [7, 11, 13], [17, 19, 23]]

[2, 3, 5, 7, 11, 13, 17, 19, 23]




```
class GFG {
    public static void main(String[] args) {
        // Creating a list of Prime Numbers
        List<Integer> PrimeNumbers = Arrays.asList(5, 7, 11,13);
        // Creating a list of Odd Numbers
        List<Integer> OddNumbers = Arrays.asList(1, 3, 5);
        // Creating a list of Even Numbers
        List<Integer> EvenNumbers = Arrays.asList(2,4,6, 8);
        List<<List<Integer>> listOfListofInts = Arrays.asList(PrimeNumbers, OddNumbers, EvenNumbers);
        System.out.println("The Structure before flattening is : " + listOfListofInts);
        // Using flatMap for transformating and flattening.
        List<Integer> listofInts  = listOfListofInts.stream()
                                    .flatMap(list -> list.stream())
                                    .collect(Collectors.toList());
         System.out.println("The Structure after flattening is : " + listofInts);
    }
}
```

09 PŘÍKLADY OPERACÍ

Operace	Popis	Druh
forEach	Poskytnutá funkce se zavolá na každém elementu	terminální
peek	Poskytnutá funkce se zavolá na každém elementu	bezstavová-NT

```
empList.stream().forEach( e ->
 e.salaryIncrement(10.0));

names.forEach(name -> {
  System.out.println(name);
});
```

```
public void whenIncrementSalaryUsingPeek_thenApplyNewSalary() {
    Employee[] arrayOfEmps = {
        new Employee(1, "Jeff Bezos", 100000.0),
        new Employee(2, "Bill Gates", 200000.0),
        new Employee(3, "Mark Zuckerberg", 300000.0)
    };

    List<Employee> empList = Arrays.asList(arrayOfEmps);

    empList.stream()
      .peek(e -> e.salaryIncrement(10.0))
      .peek(System.out::println)
      .collect(Collectors.toList());
}
```

Map/filter/reduce v Java

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);
```

09 Map/filter/reduce v Java

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.

```java
// 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()));
```

09 Map/filter/reduce v Java

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());
```

Operace	Popis	Druh
groupingBy	Zgrupuje data	bezstavová-NT

```
// 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()));
```

09 Map/filter/reduce v Java

Operace	Popis	Druh
partitioningBy	Rozdělí stream na dva	bezstavová-NT

```
public void whenStreamPartition_thenGetMap() {
    List<Integer> intList = Arrays.asList(2, 4, 5, 6, 8);
    Map<Boolean, List<Integer>> isEven = intList.stream().collect(
      Collectors.partitioningBy(i -> i % 2 == 0));

    assertEquals(isEven.get(true).size(), 4);
    assertEquals(isEven.get(false).size(), 1);
}
```

Opačnou operaci, tzv. zip mohu udělat např. takto:

```
List<String> names = new ArrayList<>(Arrays.asList("John", "Jane", "Jack", "Dennis"));
List<Integer> ages = new ArrayList<>(Arrays.asList(24, 25, 27));

IntStream
  .range(0, Math.min(names.size(), ages.size()))
  .mapToObj(i -> names.get(i) + ":" + ages.get(i))
```

09 Map/filter/reduce v Java

```
public void writeFile() throws IOException {
    String[] words = {
      "hello",
      "refer",
      "world",
      "level"
    };

    try (PrintWriter pw = new PrintWriter(
      Files.newBufferedWriter(Paths.get(fileName)))) {
        Stream.of(words).forEach(pw::println);
    }
}

private List<String> getPalindrome(Stream<String> stream, int length) {
    return stream.filter(s -> s.length() == length)
      .filter(s -> s.compareToIgnoreCase(
        new StringBuilder(s).reverse().toString()) == 0)
      .collect(Collectors.toList());
}

public void readFile() throws IOException {
    List<String> str = getPalindrome(Files.lines(Paths.get(fileName)), 5);
    assertThat(str, contains("refer", "level"));
}
```

09 Map/filter/reduce v Java

Stream může být nastaven jako **paralelní**

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

Neterminální operace (intermediate) jsou **lazy**

09 Map/filter/reduce v Java

```
//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());
```

=>

```
OUTPUT:
1 sec
2 sec
3 sec
4 sec
5 sec
In Map - Tom
In Map - Chris
In Map - Dave
```

09 Map/filter/reduce v Java

```
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.

=>

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

09 Map/filter/reduce v Java

Streamy se ne úplně dobře debuggují - musíte např. zavolat collect, abyste si prohlédli prvky streamu, což vám stream zavře. Řešení např. stáhnout *IntelliJ - Java Stream Debugger plugin*.

```
public class StreamExample {
  private static IntStream factorize(int n) {...}

  public static void main(String[] args) {  args: {}
    int[] result = Arrays.stream(new int[]{10, 87, 97, 43, 121, 20})
            .flatMap(StreamExample::factorize)
            .distinct()
            .sorted()
            .toArray();
    System.out.println(Arrays.toS
  }
}
```