# Streams

New Austrian Coding School

Programming - memo

An information flow **transforms** its elements following certain **steps**. With a Stream we **explain** those steps and they are **applied** to each element. The steps of a Stream explain ***what*** needs to be done with **each** element. We explain the ***how*** of those steps with **lambdas**.

## Still and moving information

Data that we want to manipulate is in one of these **two states**:

| | |
|---|---|
| ○ It can be **still**, motionless<br>○ **Waiting** for something to happen<br>○ It looks like a List, a Set or a Map<br>○ Similar to water drops in a **lake** | ○ It can be **moving**, changing<br>○ Being **transformed**<br>○ It looks like a Stream<br>○ Similar to water drops in a **river** |

## How to start a Stream

We can start a Stream by using the **stream** method of any collection or by using the **Stream** class.

| | |
|---|---|
| List<String> things = Arrays.*asList*("table", "chair");<br>things.stream() | Stream.*of*("table", "chair") |

## How to end a Stream

| | |
|---|---|
| ○ The **end** of a Stream gives the **purpose** of the whole information flow<br>○ **Without** an end, the Stream **doesn't** even start<br>○ There can **only** be **one** end<br>○ It is called a **terminal operation**<br>○ There are many terminal operations to choose | Examples:<br><br>.findFirst();<br>// returns an Optional with the possible first element<br>.forEach(e -> System.***out***.println(e));<br>// performs an operation for each element<br>.collect(Collectors.*toList*());<br>// collects every element in a list |

## How to continue a Stream

| | |
|---|---|
| ○ The **steps** of a Stream explain how the information flow **changes**<br>○ We can **add** as many steps as we **need** to achieve our goal<br>○ Every step returns another Stream<br>○ Every step is called an **intermediate operation**<br>○ There are many intermediate operations to choose | Examples:<br><br>.map(e -> e.toUpperCase())<br>// transforms one element into another<br>.filter(e-> e.contains("awesome"))<br>// keeps elements only if they meet the condition |

## Examples

```
List<Hero> heroes = Stream.of("Batman", "Wonder Woman", "Wolverine")
      .filter(name -> name.endsWith("man"))
      .map(name -> new Hero(name))
      .collect(Collectors.toList());

heroes.stream()
      .map(hero -> hero.getName())
      .filter(name -> name.contains(" "))
      .forEach(name -> System.out.println(name));
```