$f : A \rightarrow B$

$f : (A, B) \rightarrow C$

Consumer<T>

$f : (T) \rightarrow$ void

```java
public interface Consumer<T> {
  void accept(T t);
}
```

1

```java
Consumer<Integer> printer = new Consumer<>() {
  @Override
  public void accept(Integer x) {
    System.out.printf("Value is: %d", x);
  }
};

Consumer<Integer> printer32 = (Integer x) -> {
  System.out.printf("Value is: %d", x);
};

Consumer<Integer> printer31 = (Integer x) -> System.out.printf("Value is: %d", x);

Consumer<Integer> printer3 = x -> System.out.printf("Value is: %d", x);
```

we can eliminate {} if lambda has only 1 line statement

It doesn't matter how we name it because only one method

type can be taken from the left part

```java
public static List<Integer> onlyNegatives(List<Integer> xs) {
    List<Integer> out = new ArrayList<Integer>();
    xs.forEach(x -> {
        if (x < 0) out.add(x);
    });
    return out;
}
```

1 ms

1. manually create
2. manually adding
3. manually

```java
public static List<Integer> onlyNegatives2(List<Integer> xs) {
    return
        xs.stream()
          .filter(x -> x < 0)
          .collect(Collectors.toList());
}
```

declarative way

2 ms

2 rule

I/O access
+100 ms

Predicate<T>
test: (T) -> bool

Map<K,V> ~ Set<Entry<K,V>>
.stream .stream

classifier
KeyExtractor

Pizza (name, size)

- groupingBy (p -> p.name)

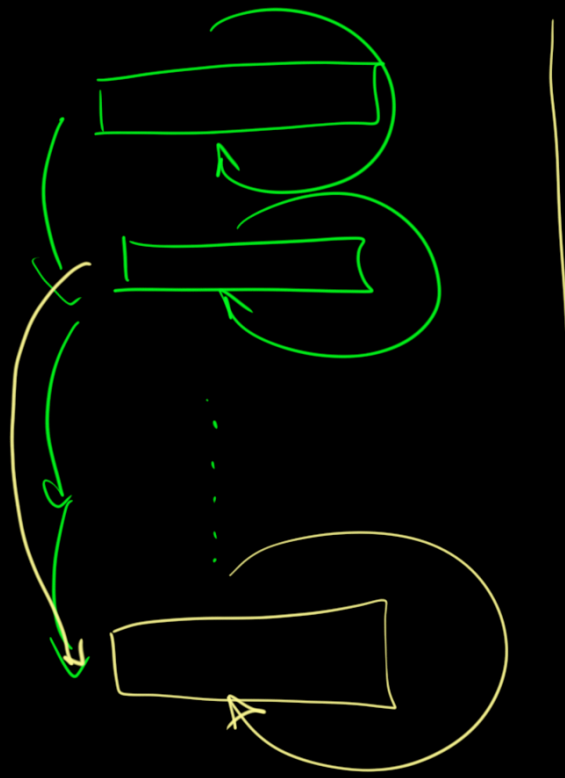
Map<String, List<Pizza>>

(p -> p.size)

Map<Int, List<Pizza>>

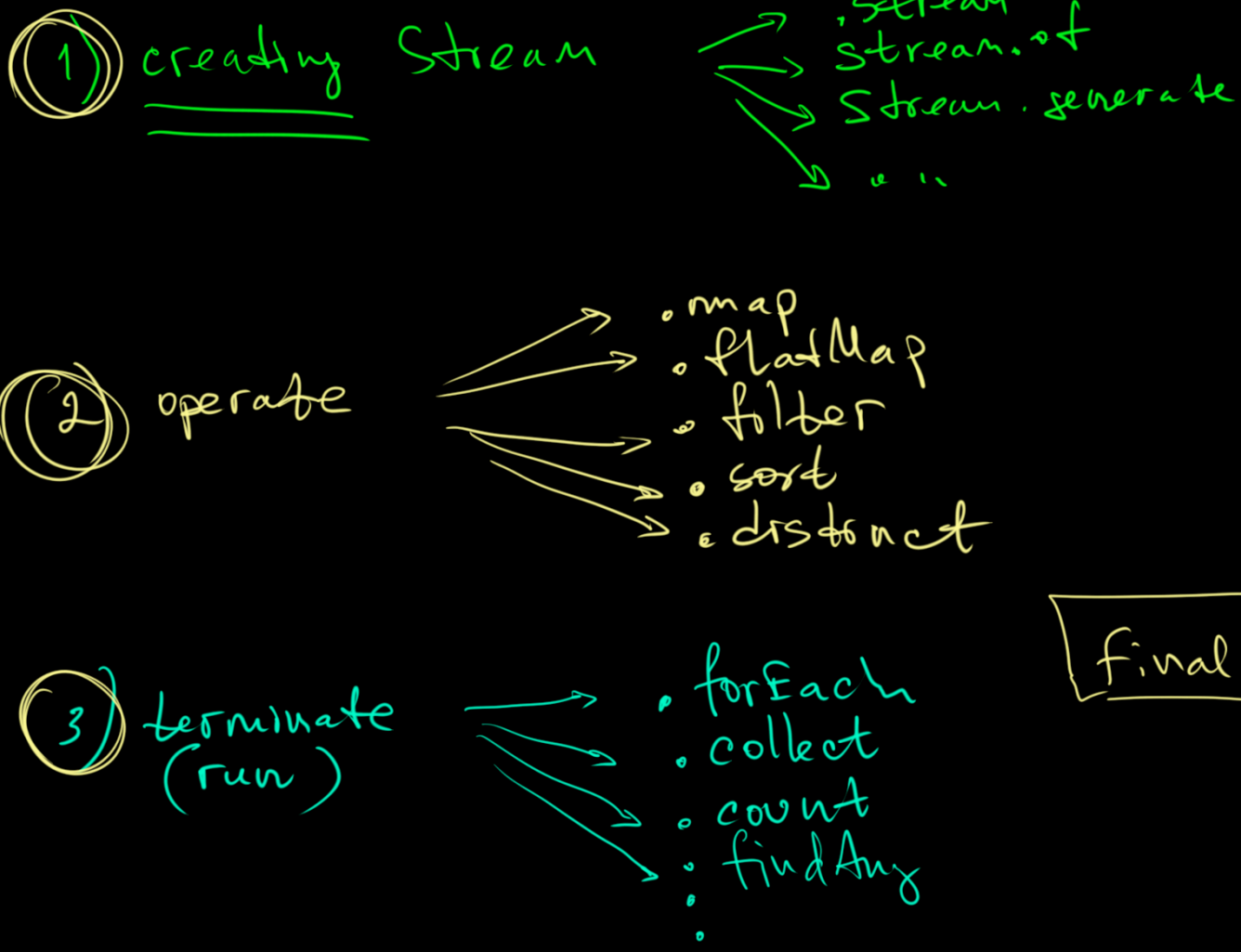
```
Stream<Pair<Character, Pair<Integer, List<Integer>>>> groupped =
    IntStream.range(0, line.length())                          IntStream
    .mapToObj(idx -> Pair.of(line.charAt(idx), idx))           Stream<Pair<Character, Integer>>
    .filter(p -> Character.isLetter(p.a))
    .collect(Collectors.groupingBy(p -> p.a))                  Map<Character, List<Pair<Character, Integer>>>
    .entrySet()                                                Set<Entry<Character, List<Pair<Character, Integer>>>>
    .stream()                                                  Stream<Entry<Character, List<Pair<Character, Integer>>>>
    .map(e ->
        Pair.of(
            e.getKey(),
            e.getValue().stream().map(p -> p.b).collect(Collectors.toList())
        )
    )                                                          Stream<Pair<Character, List<Integer>>>
    .sorted(Comparator.comparingInt(x -> x.b.size()))
    .map(p -> Pair.of(p.a, Pair.of(p.b.size(), p.b)));
```

Pair<Char, List> → Pair<Char, Pair<Int, List<Int>>>



```
Stream<Triple<Character, Integer, List<Integer>>> grouped = IntStream.range(0, line.length())
    .mapToObj(idx -> Pair.of(line.charAt(idx), idx)) Stream<Pair<Character, Integer>>
    .filter(p -> Character.isLetter(p.a))
    .map(p -> Pair.of(Character.toLowerCase(p.a), p.b))
    .collect(Collectors.groupingBy(p -> p.a)) Map<Character, List<Pair<Character, Integer>>>
    .entrySet() Set<Entry<Character, List<Pair<Character, Integer>>>>
    .stream() Stream<Entry<Character, List<Pair<Character, Integer>>>>
    .map(e ->
        Pair.of(
            e.getKey(),
            e.getValue().stream().map(p -> p.b).collect(Collectors.toList())
        )
    ) Stream<Pair<Character, List<Integer>>>
    .sorted(Comparator.comparingInt(x -> x.b.size()))
    .map(p -> Triple.of(p.a, p.b.size(), p.b));
```

all data passing between steps is done by Stream


1
creating Stream
.Stream
Stream.of
Stream.generate
...
2
operate
.map
.flatMap
.filter
.sort
.distinct
3
terminate
(run)
.forEach
.collect
.count
.findAny
final result here

```
List<String> xx = Arrays.asList(1, 2, 3)          List<Integer>
    .stream()                                     Stream<Integer>
    .map(x -> x + 30)
    .map(x -> new Pizza( n: "Margarita", x))      Stream<Pizza>
    .filter(p -> p.size > 31)
    .map(p -> p.name)                             Stream<String>
    .distinct()
    .map(n -> n.toUpperCase())
    .collect(Collectors.toList());
```

stream takes 1-by-1
and passes through ALL steps automatically
very efficient

```java
List<Integer> ages = Arrays.asList(22, 24, 26, 30, 44, 50);

List<String> names = Arrays.asList("Jim", "Sergio", "Bill", "Name");
```