

Programação Orientada por Objetos

Introdução às Coleções

Prof. Cédric Grueau

Prof. José Sena Pereira

Departamento de Sistemas e Informática
Escola Superior de Tecnologia de Setúbal
Instituto Politécnico de Setúbal

2022/2023



Sumário



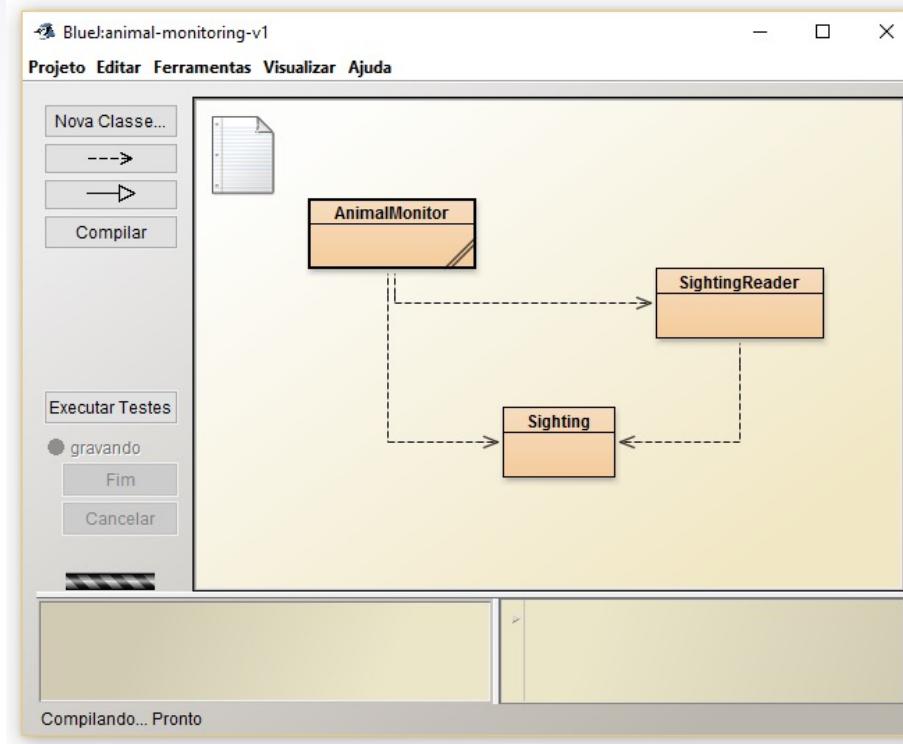
- ▶ Exemplo Monitorização de animais
- ▶ Processamento funcional de coleções
- ▶ Expressões lambda

Exemplo – Monitorização de animais

- ▶ Requisitos da aplicação:
 - ▶ Sistema de monitorização de populações animais.
 - ▶ Processa relatórios dos avistamentos de animais enviados por vários observadores.
 - ▶ Cada relatório consiste no nome e número de animais avistados, quem enviou o relatório (um inteiro), a área da observação (um inteiro) e a altura em que foi feita a observação (um inteiro representando o número de dias desde o inicio da monitorização)

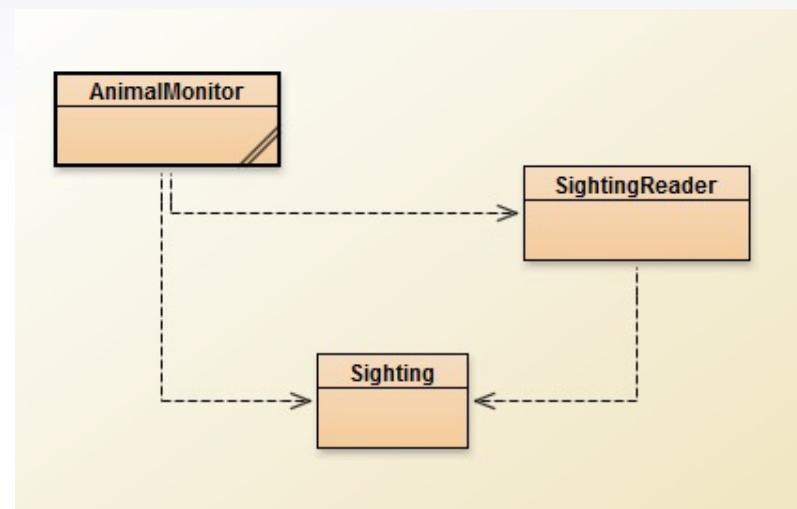


Exemplo – Monitorização de animais



Exemplo – Monitorização de animais

- ▶ Classes da aplicação:
 - ▷ **Sighting**
 - ▷ Relatório da observação.
 - ▷ **SightingReader**
 - ▷ Usada para ler uma lista de observações a partir de um ficheiro.
 - ▷ **AnimalMonitor**
 - ▷ Faz o processamento de relatórios de observações.



Exemplo – Monitorização de animais

- ▶ Representação da observação - classe **Sighting**

```
public class Sighting {  
    private String animal;  
    private int spotter; // observador  
    private int count;  
    private int area;  
    private int period;  
  
    public Sighting(String animal, int spotter,  
                    int count, int area, int period) {  
        this.animal = animal;  
        this.spotter = spotter;  
        this.count = count;  
        this.area = area;  
        this.period = period;  
    }  
    // restantes métodos  
}
```

Exemplo – Monitorização de animais

- ▶ Classe **Sighting** – métodos seletores **get**

```
•  
public String getAnimal() {  
    return animal;  
}  
  
public int getSpotter() {  
    return spotter;  
}  
  
public int getCount() {  
    return count;  
}  
  
public int getArea() {  
    return area;  
}  
  
public int getPeriod() {  
    return period;  
}
```

Exemplo – Monitorização de animais

- ▶ Classe **Sighting** – método **getDetails**

Podia ter sido
chamado
toString

```
public String getDetails() {  
  
    return animal +  
        ", count = " + count +  
        ", area = " + area +  
        ", spotter = " + spotter +  
        ", period = " + period;  
}
```



Exemplo – Monitorização de animais

- ▶ Obtenção de observações a partir de ficheiro – classe **SightingReader**

```
public class SightingReader {  
    public SightingReader() {}  
  
    public ArrayList<Sighting> getObservations(String filename)  
    {  
        // Devolve uma lista de observações  
        // lida a partir de um ficheiro.  
    }  
}
```

Não precisamos de saber os detalhes de implementação

Exemplo – Monitorização de animais

- ▶ Processamento das observações - classe **AnimalMonitor**

Lê a lista de observações e adiciona-a à lista sightings

```
public class AnimalMonitor {  
    private ArrayList<Sighting> sightings;  
  
    public AnimalMonitor() {  
        this.sightings = new ArrayList<>();  
    }  
  
    public void addSightings(String filename) {  
        SightingReader reader = new SightingReader();  
        sightings.addAll(reader.getSightings(filename));  
    }  
  
    // restantes métodos  
}
```

Exemplo – Monitorização de animais

- ▶ Classe **AnimalMonitor**
 - método ????????

O que faz?

```
public void ???????() {  
  
    for(Sighting record : sightings) {  
        System.out.println(record.getDetails());  
    }  
}
```

Exemplo – Monitorização de animais

- ▶ Classe **AnimalMonitor**
– método **printList**

```
public void printList() {  
  
    for(Sighting record : sightings) {  
        System.out.println(record.getDetails());  
    }  
}
```

Mostra todas as
observações



Exemplo – Monitorização de animais

- Classe **AnimalMonitor**
 - método ????????

O que faz?

```
public void ???????(String animal) {  
  
    for(Sighting record : sightings) {  
        if(animal.equals(record.getAnimal())) {  
            System.out.println(record.getDetails());  
        }  
    }  
}
```

Exemplo – Monitorização de animais

- ▶ Classe **AnimalMonitor**
 - método
printSightingsOf

Mostra apenas as observações de um dado animal

```
public void printSightingsOf(String animal) {  
  
    for(Sighting record : sightings) {  
        if(animal.equals(record.getAnimal())) {  
            System.out.println(record.getDetails());  
        }  
    }  
}
```

Exemplo – Monitorização de animais

- ▶ Classe **AnimalMonitor**
 - método ??????????

O que faz?

```
public void ???????(int spotter) {  
  
    for(Sighting record : sightings) {  
        if(record.getSpotter() == spotter) {  
            System.out.println(record.getDetails());  
        }  
    }  
}
```

Exemplo – Monitorização de animais

- Classe **AnimalMonitor**
 - método
printSightingsBy

Mostra as observações feitas por um dado observador

```
public void printSightingsBy(int spotter) {  
  
    for(Sighting record : sightings) {  
        if(record.getSpotter() == spotter) {  
            System.out.println(record.getDetails());  
        }  
    }  
}
```

Exemplo – Monitorização de animais

- ▶ Classe **AnimalMonitor**
– método **?????????**



```
public int ???????(String animal) {  
    int total = 0;  
    for(Sighting sighting : sightings) {  
        if(animal.equals(sighting.getAnimal())) {  
            total = total + sighting.getCount();  
        }  
    }  
    return total;  
}
```



Exemplo – Monitorização de animais

- ▶ Classe **AnimalMonitor**
- método **getCount**

Retorna o total de observações de um dado animal

```
public int getCount(String animal) {  
  
    int total = 0;  
    for(Sighting sighting : sightings) {  
        if(animal.equals(sighting.getAnimal())) {  
            total = total + sighting.getCount();  
        }  
    }  
    return total;  
}
```

Exemplo – Monitorização de animais

- ▶ Classe **AnimalMonitor**
– método **?????????**

O que faz?

```
public void ???????(ArrayList<String> animalNames,  
                     int dangerThreshold) {  
  
    for(String animal : animalNames) {  
        if(getCount(animal) <= dangerThreshold) {  
            System.out.println(animal + " is endangered.");  
        }  
    }  
}
```

Exemplo – Monitorização de animais

- ▶ Classe **AnimalMonitor**
 - método
printEndangered

Mostra todos os animais cujas observações tinham um número de avistamentos abaixo de um dado limiar

```
public void printEndangered(ArrayList<String> animalNames,  
                           int dangerThreshold) {  
  
    for(String animal : animalNames) {  
        if(getCount(animal) <= dangerThreshold) {  
            System.out.println(animal + " is endangered."  
        }  
    }  
}
```

Exemplo – Monitorização de animais

- ▶ Classe **AnimalMonitor**
 - método **?????????**

O que faz?

```
public void ????????() {  
  
    Iterator<Sighting> it = sightings.iterator();  
    while(it.hasNext()) {  
        Sighting record = it.next();  
        if(record.getCount() == 0) {  
            it.remove();  
        }  
    }  
}
```

Exemplo – Monitorização de animais

- ▶ Classe **AnimalMonitor**
 - método
removeZeroCounts

Elimina todos os relatórios cujo número de animais avistados é 0.

```
public void removeZeroCounts() {  
  
    Iterator<Sighting> it = sightings.iterator();  
    while(it.hasNext()) {  
        Sighting record = it.next();  
        if(record.getCount() == 0) {  
            it.remove();  
        }  
    }  
}
```

Exemplo – Monitorização de animais

- ▶ Classe **AnimalMonitor**
 - método ?????????



```
public ArrayList<Sighting> ???????(String animal, int area) {  
    ArrayList<Sighting> records = new ArrayList<>();  
    for(Sighting record : sightings) {  
        if(animal.equals(record.getAnimal())) {  
            if(record.getArea() == area) {  
                records.add(record);  
            }  
        }  
    }  
    return records;  
}
```

Exemplo – Monitorização de animais

- ▶ Classe AnimalMonitor – método
getSightingsInArea

Retorna uma lista de observações de um animal especificado numa dada área.

```
public ArrayList<Sighting> getSightingsInArea(String animal, int area) {  
    ArrayList<Sighting> records = new ArrayList<>();  
    for(Sighting record : sightings) {  
        if(animal.equals(record.getAnimal())) {  
            if(record.getArea() == area) {  
                records.add(record);  
            }  
        }  
    }  
    return records;  
}
```

Exemplo – Monitorização de animais

- ▶ Classe **AnimalMonitor**
 - método ?????????

O que faz?

```
public ArrayList<Sighting> ???????(String animal) {  
    ArrayList<Sighting> filtered = new ArrayList<>();  
    for(Sighting record : sightings) {  
        if(animal.equals(record.getAnimal())) {  
            filtered.add(record);  
        }  
    }  
    return filtered;  
}
```

Exemplo – Monitorização de animais

- ▶ Classe **AnimalMonitor**
 - método
getSightingsOf

Filtre as observações,
retornando uma lista apenas
com os relatórios de um
dado animal.

```
public ArrayList<Sighting> getSightingsOf(String animal) {  
  
    ArrayList<Sighting> filtered = new ArrayList<>();  
    for(Sighting record : sightings) {  
        if(animal.equals(record.getAnimal())) {  
            filtered.add(record);  
        }  
    }  
    return filtered;  
}
```

Processamento Imperativo de Coleções

- ▶ Processamento imperativo de coleções
 - ▶ Todos os processamentos de coleções feitos até agora utilizaram um **paradigma de programação imperativo**.
 - ▶ Consistiam num ciclo que ia obter cada um dos elementos da coleção e efetuava uma determinada ação com esse elemento.A estrutura deste processamento era semelhante a:

```
ciclo (para cada elemento da coleção)
    obter o elemento
    fazer algo com o elemento
fim do ciclo
```

O ciclo podia ser qualquer um: **while, do-while, for, for-each**

Processamento Funcional de Coleções

- ▶ Processamento funcional de coleções
 - ▶ Em muitos casos o **processamento funcional de coleções** é mais prático, conciso e expressivo do que o processamento imperativo. O Java adotou este tipo de processamento a partir da versão 8.
 - ▶ O conceito de **lambda** está na base do funcionamento do processamento funcional:
 - ▶ A interpretação mais simples é que é possível **passar como parâmetro de um método um pedaço de código** que o método pode executar depois quando necessitar. Anteriormente apenas passámos para métodos, valores de tipos primitivos e objetos.

Processamento Funcional de Coleções

- ▶ Processamento funcional de coleções
 - ▶ Processamento de coleções com **expressões lambda**:
 - ▶ A abordagem usada com expressões lambda é algo como: “**Faz isto a cada elemento da coleção**”, onde “isto” representa um pedaço de código. A estrutura deste processamento é do tipo:

```
Coleção.fazIstoParaCadaElemento(pedaço de código)
```

- ▶ O código fica mais simples.
- ▶ O ciclo parece ter desaparecido.

Expressões Lambda

- ▶ Sintaxe de uma **expressão lambda**
 - ▶ As expressões lambda são parecidas com os métodos mas, ao contrário dos métodos, não têm de pertencer a uma classe e não necessitam de um nome.
 - ▶ Exemplo
 - Método para mostrar os detalhes de um relatório de avistamento

```
public void printSighting(Sighting record) {  
    System.out.println(record.getDetails());  
}
```

Expressão lambda equivalente ao método anterior

```
(Sighting record) -> {  
    System.out.println(record.getDetails());  
}
```

Expressões Lambda

- ▶ Sintaxe de uma **expressão lambda**
 - ▶ Exemplo: Expressão lambda equivalente ao método anterior
 - ▶

```
(Sighting record) -> {
    System.out.println(record.getDetails());
}
```
 - ▶ Sem o modificador **public** ou **private**
 - ▶ Sem tipo de retorno
 - ▶ O compilador consegue saber o tipo de retorno pelo que é retornado do método
 - ▶ Sem nome do método
 - ▶ Começa logo com a lista de parâmetros
 - ▶ Uma seta (**->**) separa a lista de parâmetros do corpo do lambda

Expressões Lambda

- ▶ Expressões lambda
 - ▷ São usadas habitualmente **para tarefas simples** que não requerem a complexidade de uma classe.
 - ▷ São também conhecidas **for funções anónimas** (ou métodos anónimos)
 - ▷ Utilização em coleções - o método **forEach**: Coleção.fazIstoParaCadaElemento(pedaço de código)

myList.forEach(... Código que se aplica a cada elemento da coleção ...)

O parâmetro deste método é uma **expressão lambda**

Expressões Lambda

▶ Expressões lambda –

Exemplo com o método
printList:

Já não aparece
o ciclo

```
public void printList() {  
    for(Sighting record : sightings) {  
        System.out.println(record.getDetails());  
    }  
}  
  
// Com expressões lambda:  
public void printList() {  
    sightings.forEach(  
        (Sighting record) -> {  
            System.out.println(record.getDetails());  
        }  
    );  
}
```

Parece
mais
complexo!

Expressões Lambda

- ▶ Expressões lambda - Exemplo com o método printList:

Simplificações

1. O tipo do parâmetro pode ser omitido uma vez que o compilador consegue determiná-lo: é o mesmo que o tipo dos elementos da coleção.

Simplificação
1

```
//Com expressões lambda:  
  
public void printList() {  
    sightings.forEach(  
        (record) -> {  
            System.out.println(record.getDetails());  
        }  
    );  
}
```

Expressões Lambda

- ▶ Expressões lambda – Exemplo com o método `printList`:

Simplificações

2. Se existir apenas um parâmetro não é necessário colocar os parênteses.

```
// Com expressões lambda:  
  
public void printList() {  
  
    sightings.forEach(  
        record -> {  
            System.out.println(record.getDetails());  
        }  
    );  
}
```

Simplificação
2

Expressões Lambda

- ▶ Expressões lambda - Exemplo com o método `printList`:

Simplificações

3. Se existir apenas uma instrução no corpo do lambda não é necessário usar chavetas, nem colocar o ponto e vírgula final.

```
// Com expressões lambda:  
  
public void printList() {  
    sightings.forEach(  
        record -> System.out.println(record.getDetails())  
    );  
}
```

Simplificação
3

Expressões Lambda

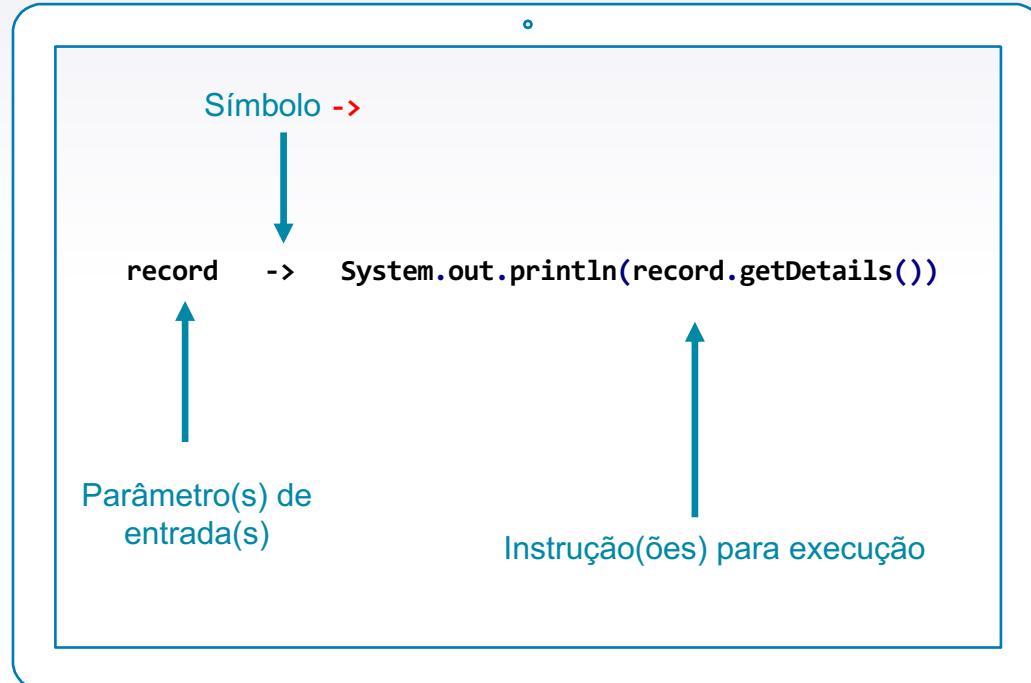
- ▶ Expressões lambda – Exemplo com o método **printList**:

```
public void printList() {  
    for(Sighting record : sightings) {  
        System.out.println(record.getDetails());  
    }  
}  
  
// Com expressões lambda:  
public void printList() {  
    sightings.forEach(record ->  
        System.out.println(record.getDetails()));  
}
```

Mais simples!

Expressões Lambda

- ▶ Sintaxe de uma **expressão lambda**



Streams

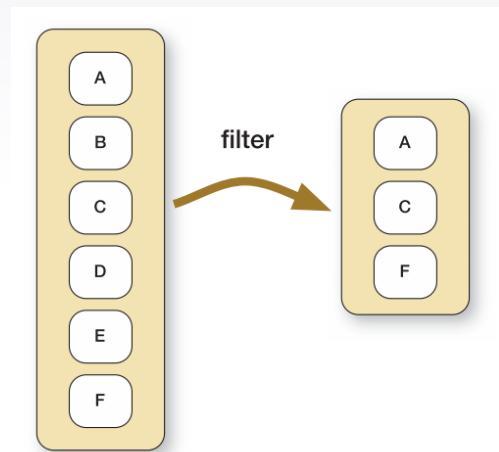
- ▶ Conceito de **stream**
 - ▶ O java 8 introduziu o conceito alargado de **stream** que é entendido como uma coleção mas com características especiais:
 - ▶ Os elementos de uma stream não são acedidos por um índice mas sim sequencialmente.
 - ▶ O conteúdo e a ordenação de uma stream não pode ser alterado.
 - ▶ Neste caso ter-se-ia que criar uma nova stream
 - ▶ Uma stream pode ser potencialmente infinita!
 - ▶ Por exemplo o processamento de mensagens oriundas de uma rede social
 - ▶ É utilizado para unificar o processamento de conjuntos de dados independentemente da sua origem.
 - ▶ O método **forEach** da classe **ArrayList** é um exemplo de um método que utiliza a noção de stream.

Streams

- ▶ Conceito de **stream**
 - ▶ Algumas classes como a classe **ArrayList**, e outras classes de coleção, disponibilizam um método **stream** que retorna uma stream com os elementos da coleção por ordem dos seus índices.
 - ▶ As operações habituais em coleções podem ser feitas através de streams sem necessidade de utilizar ciclos. Estas operações utilizam expressões lambda.
 - ▶ Iremos estudar as operações de filtragem (**filter**), mapeamento (**map**) e redução (**reduce**) utilizadas em streams.

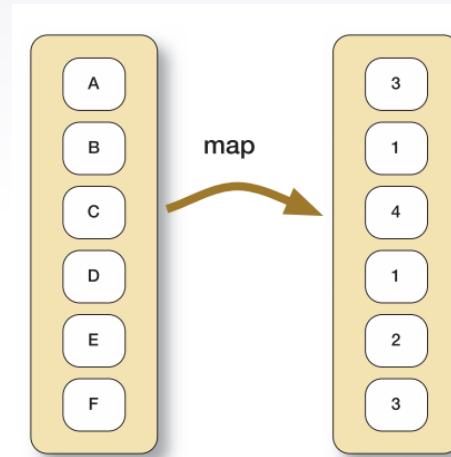
Streams: o método filter

- ▶ O método **filter**
 - ▶ Este método recebe uma **stream**, seleciona alguns dos seus elementos e cria uma nova **stream** apenas com os elementos selecionados
 - ▶ Diz-se que aplicamos um filtro à **stream**. Um exemplo poderia ser a seleção de todos os relatórios do animal “elefante”.



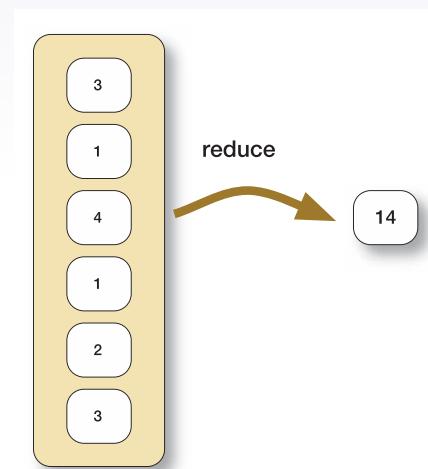
Streams: o método map

- ▶ O método **map**
 - ▶ Este método recebe uma **stream** e cria uma nova **stream** onde os elementos iniciais são mapeados (transformados) outros elementos
 - ▶ Um exemplo poderia ser substituir cada objeto **Sighting** da coleção inicial pelo número de animais observados em cada um deles.



Streams: o método reduce

- ▶ O método **reduce**
 - ▶ Este método recebe uma **stream** e reduz todos os elementos a um único valor.
 - ▶ Um exemplo poderia ser termos o número de todos os elefantes observados e depois somarmos tudo. É um valor único com a soma de todos.



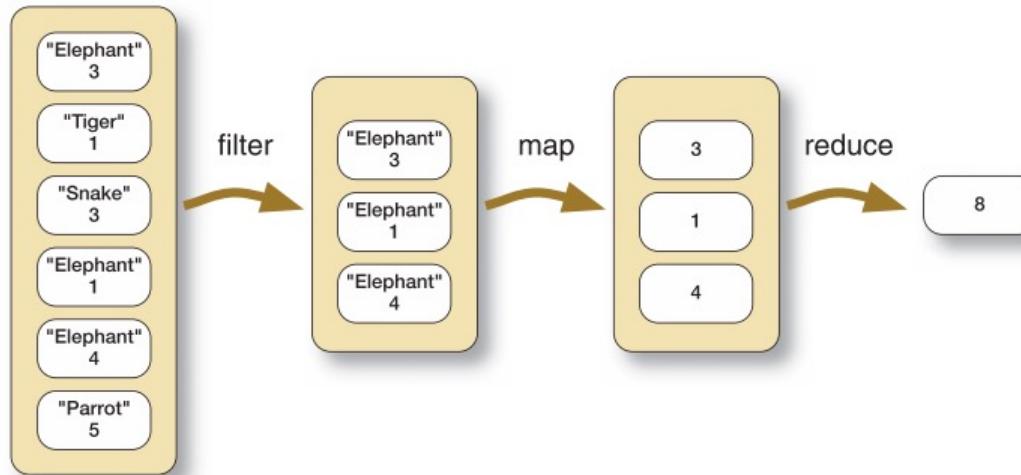
Streams: pipelines

- ▶ Pipelines
 - ▶ As streams e os seus métodos tornam-se mais úteis se os pudermos juntar como uma sequência de operações. É esse o conceito de **pipelines**.
 - ▶ Por exemplo queremos saber quantos avistamentos de elefantes foram registados:
 1. Filtramos a coleção de relatórios para obtermos apenas os relatórios respeitantes a elefantes.
 2. Mapeamos cada relatório obtido (um objeto da classe **Sighting**) no número de animais que está nesse relatório.
 3. Reduzimos a um único valor que corresponde à soma de todos os números de animais obtidos anteriormente.

Streams: pipelines

▶ Pipelines

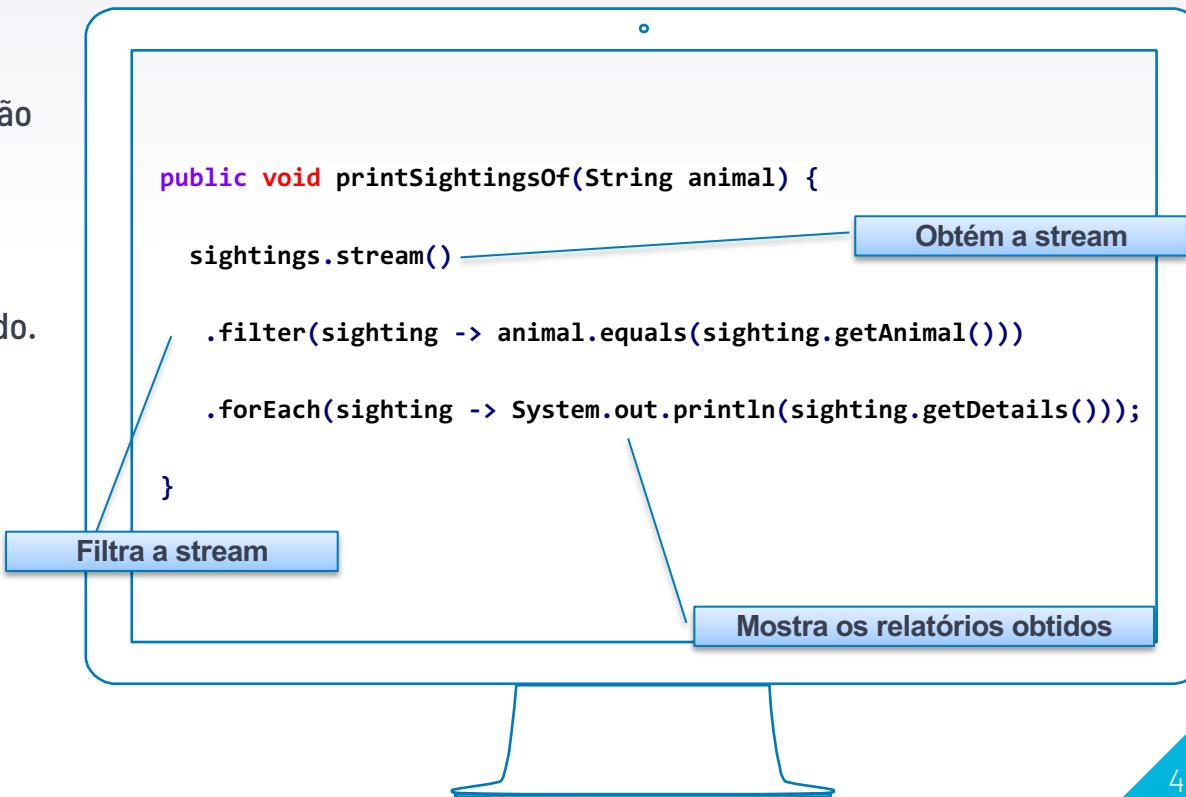
- ▶ Quantos avistamentos de elefantes foram registados:
 1. Filtramos a coleção de relatórios para obtermos apenas os relatórios respeitantes a elefantes.
 2. Mapeamos cada relatório obtido (um objeto da classe **Sighting**) no número de animais que está nesse relatório.
 3. Reduzimos a um único valor que corresponde à soma de todos os números de animais obtidos anteriormente.



Streams: o método filter

Método `filter`

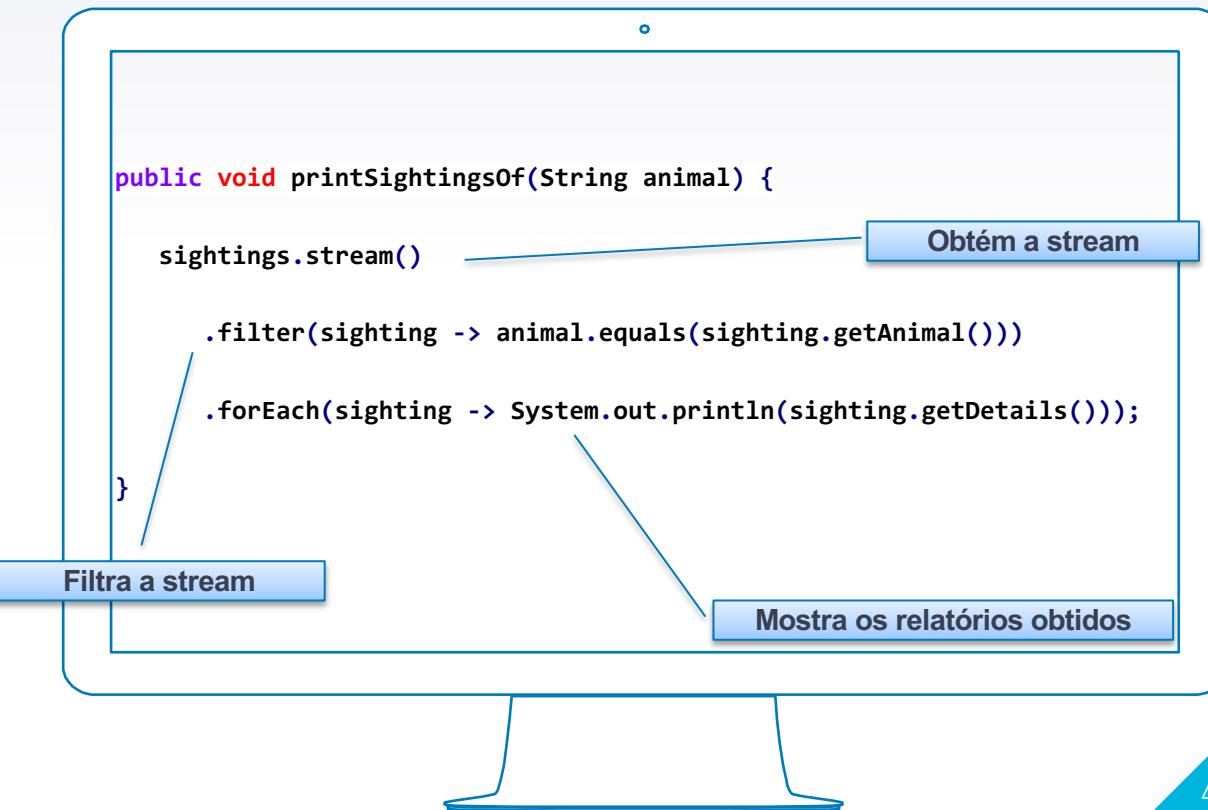
- ▶ Cria um subconjunto (filtrado) da coleção inicial.
- ▶ Normalmente existe uma condição que deve ser verdadeira para um dado elemento para que este seja selecionado. Esta condição é dada na forma de uma expressão lambda.
 - ▶ Exemplo: Mostrar todos os relatórios de um determinado animal



Streams: o método filter

Método **filter**

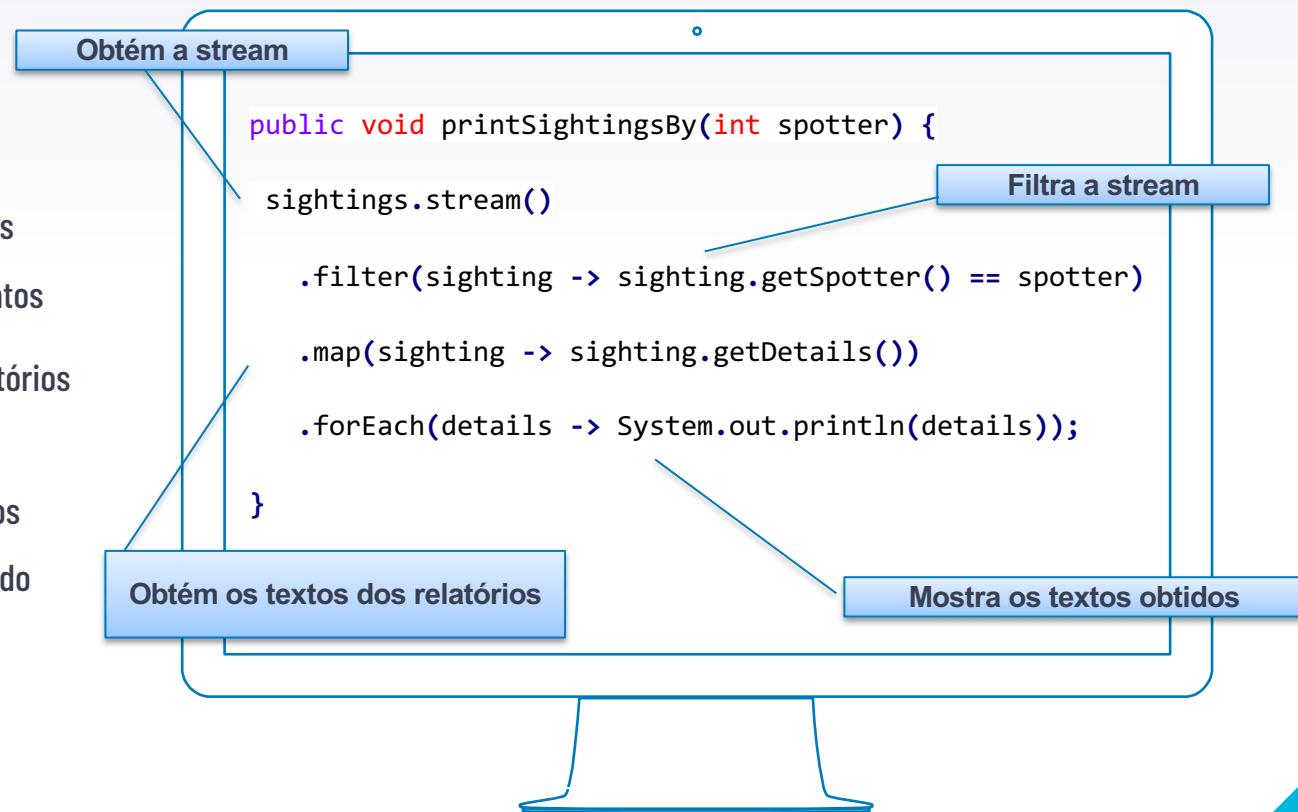
- ▶ Cria um subconjunto (filtrado) da coleção inicial.
- ▶ Normalmente existe uma condição que deve ser verdadeira para um dado elemento para que este seja selecionado. Esta condição é dada na forma de uma expressão lambda.
 - ▶ Exemplo: Mostrar todos os relatórios de um determinado animal



Streams: o método map

O método map

- ▶ Cria uma nova **stream** onde os elementos iniciais são mapeados (transformados) nouros elementos
 - ▶ Exemplo mostrar os relatórios de um dado observador obtendo primeiro todos os relatórios (mapeamento do relatório no seu texto descritivo.



Streams: o método reduce

O método `reduce`

- ▶ Recebe uma **stream** e reduz todos os elementos a um único valor.
 - ▶ Exemplo. Contar todos os animais de um determinado tipo.

Filtre a stream

```
public int getCount(String animal) {  
    return sightings.stream()  
        .filter(sighting -> animal.equals(sighting.getAnimal()))  
        .map(sighting -> sighting.getCount())  
        .reduce(0, (runningSum, count) -> runningSum + count);  
}
```

Obtém a stream

Obtém o número de avistamentos por relatório

Soma os todos os números de avistamentos

Streams: o método reduce

O método **reduce**

- ▶ Recebe uma **stream** e reduz todos os elementos a um único valor.
 - ▶ Exemplo. Contar todos os animais de um determinado tipo.

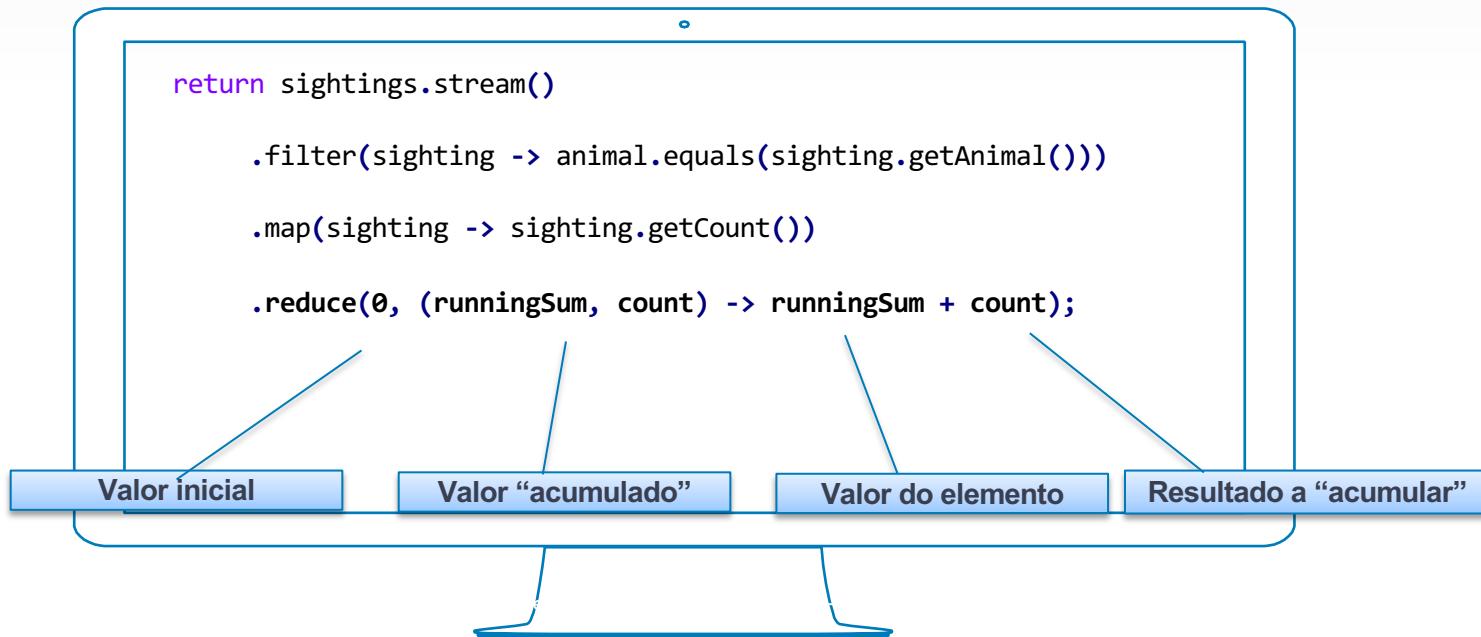
Os parâmetros do método **reduce** são um pouco mais complicados!

```
public int getCount(String animal) {  
  
    return sightings.stream()  
  
        .filter(sighting -> animal.equals(sighting.getAnimal()))  
  
        .map(sighting -> sighting.getCount())  
  
        .reduce(0, (runningSum, count) -> runningSum + count);  
  
}
```

Streams: o método reduce

O método **reduce**

- ▶ Recebe uma **stream** e reduz todos os elementos a um único valor.
 - ▶ Exemplo. Contar todos os animais de um determinado tipo.



Exemplo – Monitorização de animais

- ▶ Processamento funcional das observações - nova classe **AnimalMonitor**

```
public class AnimalMonitor {  
    private ArrayList<Sighting> sightings;  
    public AnimalMonitor() {  
        this.sightings = new ArrayList<>();  
    }  
    public void addSightings(String filename) {  
        SightingReader reader = new SightingReader();  
        sightings.addAll(reader.getSightings(filename));  
    }  
    // restantes método  
}
```

Exemplo – Monitorização de animais

Classe AnimalMonitor – método
printList

```
public void printList() {  
    for(Sighting record : sightings) {  
        System.out.println(record.getDetails());  
    }  
}
```

Nova classe AnimalMonitor – método
printList

```
public void printList() {  
    sightings.forEach(record -> System.out.println(record.getDetails()));  
}
```

Exemplo – Monitorização de animais

Classe **AnimalMonitor** - método
printSightingsOf

Nova classe **AnimalMonitor** – método
printSightingsOf

```
public void printSightingsOf(String animal) {  
    for(Sighting record : sightings) {  
        if(animal.equals(record.getAnimal())) {  
            System.out.println(record.getDetails());  
        }  
    }  
}  
  
public void printSightingsOf(String animal) {  
    sightings.stream()  
        .filter(sighting -> animal.equals(sighting.getAnimal()))  
        .forEach(sighting -> System.out.println(sighting.getDetails()));  
}
```

Exemplo – Monitorização de animais

Classe AnimalMonitor – método
printSightingsBy

```
public void printSightingsBy(int spotter) {  
    for(Sighting record : sightings) {  
        if(record.getSpotter() == spotter) {  
            System.out.println(record.getDetails());  
        }  
    }  
}
```

Nova classe AnimalMonitor – método
printSightingsBy

```
public void printSightingsBy(int spotter) {  
    sightings.stream()  
        .filter(sighting -> sighting.getSpotter() == spotter)  
        .map(sighting -> sighting.getDetails())  
        .forEach(details -> System.out.println(details));  
}
```

Exemplo – Monitorização de animais

Classe AnimalMonitor –
método **getCount**

```
public int getCount(String animal) {  
    int total = 0;  
    for(Sighting sighting : sightings) {  
        if(animal.equals(sighting.getAnimal())) {  
            total = total + sighting.getCount();  
        }  
    }  
    return total;  
}  
  
public int getCount(String animal) {  
    return sightings.stream()  
        .filter(sighting -> animal.equals(sighting.getAnimal()))  
        .map(sighting -> sighting.getCount())  
        .reduce(0, (runningSum, count) -> runningSum + count);  
}
```

Exemplo – Monitorização de animais

- ▶ Classe AnimalMonitor – método **removeZeroCounts**

Também é possível com coleções usando o método `removeIf` parecido com o `forEach`.

```
public void removeZeroCounts() {  
    Iterator<Sighting> it = sightings.iterator();  
    while(it.hasNext()) {  
        Sighting record = it.next();  
        if(record.getCount() == 0) {  
            it.remove();  
        }  
    }  
}
```

Exemplo – Monitorização de animais

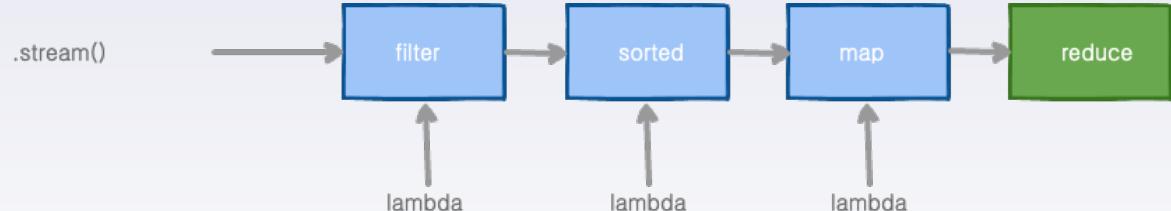
- ▶ Classe **AnimalMonitor**
 - método
removeZeroCounts



Condição de remoção do elemento

```
public void removeZeroCounts() {  
    sightings.removeIf(sighting -> sighting.getCount() == 0);  
}
```

Streams



▶ Streams

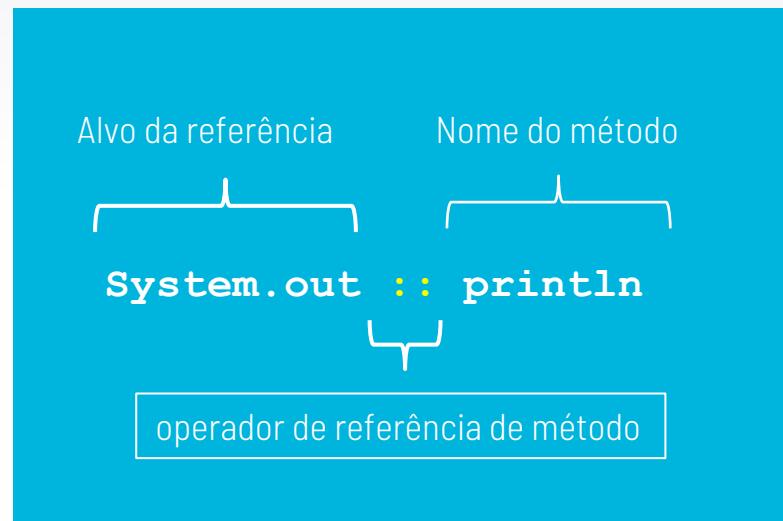
- ▶ Alguns dos métodos usados pelas streams em pipelines são considerados **intermediários**, enquanto outros são **terminais**.
 - ▶ Os métodos intermediários devolvem streams e podem ser seguidos por outros métodos.
 - ▶ Por exemplo map e filter.
 - ▶ Os métodos terminais não podem ser seguidos por outros métodos de stream porque não devolvem streams ou porque não devolvem nada (são void).
 - ▶ Por exemplo reduce.
- ▶ As streams têm mais métodos para além dos estudados aqui (`filter`, `map` e `reduce`) que podem ser consultados na documentação oficial e que permitem outras operações.
 - ▶ Por exemplo `count`, `limit`, `findFirst`, `skip`, etc.:

Operador de resolução de escopo em Java

- ▶ O operador (:) em Java é conhecido como operador de referência de método ou operador de dois pontos.
- ▶ É usado para chamar um método fazendo referência a ele com a ajuda de sua classe diretamente. Podemos usar o operador de referência de método em vez de expressões lambda porque ele se comporta da mesma forma que as expressões lambda.
- ▶ A única diferença entre a expressão lambda e o operador de referência de método é que, em vez de fornecer um delegado ao método, ele usa uma referência direta ao método pelo nome.

Operador de referência de método em java

- ▶ O alvo da referência é colocado antes do operador (::) e o nome do método é escrito depois.



Operador de referência de método: exemplo

```
import java.util.stream.*;
public class LambdaExpressionExample {
    public static void main(String[] args) {
        Stream<String> st = Stream.of("Lisboa", "Porto", "Évora", "Faro", "Aveiro", "Braga");
        st.forEach(city -> System.out.println(city));
    }
}
```

► Alternativa

```
public static void main(String[] args) {
    Stream<String> st = Stream.of("Lisboa", "Porto", "Évora", "Faro", "Aveiro", "Braga");
    st.forEach(System.out::println);
}
```

Tipos de Referências de Método

- Existem quatro tipos de referências de método em Java:

Tipos	Descrição	Sintaxe	Exemplo
Referência a um método estático	É usado para referenciar um método estático de uma classe.	<code>ContainingClass::staticMethodName</code>	<code>Math::floor</code> é equivalente a <code>Math.floor(x)</code>
Referência a um método de instância de um objeto específico	Refere-se a um método de instância usando uma referência a um objeto fornecido.	<code>containingObject::instanceMethodName</code>	<code>System.out::println</code> é equivalente a <code>System.out.println(x)</code>
Referência a um método de instância de um objeto arbitrário de um determinado tipo	Chama o método de instância numa referência a um objeto fornecido pelo contexto.	<code>ContainingType::methodName</code>	<code>String::indexOf</code> é equivalente a <code>str.indexOf()</code>
Referência a um construtor	Faz referência a um construtor.	<code>ClassName::new</code>	<code>LinkedList::new</code> é equivalente a <code>new LinkedList()</code>

Bibliografia

- ▶ Objects First with Java (6th Edition), David Barnes & Michael Kölling, Pearson Education Limited, 2016
 - ▶ Capítulo 5

