comparing(<Function<? super T, ? extends U> keyExtractor)

- Permite que você passe qualquer campo para ordenação por meio de uma expressão lambda ou method reference.
- Versões primitivas de Functions também são suportadas pelo método

thenComparing(Comparator<? super T> Other)

- Adiciona mais campos para ordenação da Collection

reversed()

- Inverte a ordenação

```
1:  public class TesteOrdenacao02 {
2:     public static void main(String[] args) {
3:        List<Produto> produtos = new ArrayList<>();
4:        produtos = BDProduto.getProdutos();
5:
6:        Function<Produto, Double> comparaValor = p -> p.getValor();
7:        Function<Produto, String> comparaTipo = p -> p.getTipo();
8:
9:        Comparator<Produto> comparaValorTipo = Comparator
10:                                    .comparing(comparaTipo)
11:                                    .thenComparing(comparaValor);
12:
13:       produtos.stream()
14:          .sorted(comparaValorTipo)
15:          .forEach(p -> tipoDescricaoValor(p));
16:
17:    }
18:
19:    private static void tipoDescricaoValor(Produto p) {
20:       System.out.printf("Tipo: %s - descrição: %s - valor: %s\n",
21:             p.getTipo(),
22:             p.getDescricao(),
23:             moeda(p.getValor()));
24:    }
25: }
```

Coleta de dados do pipeline:

averagingDouble(ToDoubleFunction<? super T> mapper)

- Retorna a média aritmética dos valores definidos pela função

groupingBy(Function<? super T, ? extends K> classifier)

- Retorna um **Map<K,List>** onde K conterá o valor do campo definido para o agrupamento na função e List conterá a lista de elementos que possuem esse valor. Será gerada uma lista para cada variação em K.

joining(CharSequence delimiter)

- Concatena em uma String os dados separando-os pelo(s) caractere(s) definido(s) em delimiter.

partitionBy(Predicate<? super T> predicate)

- Retorna um **Map<Boolean,List>** onde na chave **true**, estará a lista de todos os elementos que retornaram verdadeiro para o Predicate e na chave false, os elementos que retornaram falso.

```
1:  public class TesteCollect01 {
2:    public static void main(String[] args) {
3:       List<Produto> produtos = new ArrayList<>();
4:       produtos = BDProduto.getProdutos();
5:
6:       double mediaValorBike = produtos.stream()
7:                           .filter(p -> p instanceof Bicicleta)
8:                           .collect(Collectors.averagingDouble(p -> p.getValor()));
9:       System.out.printf("Média valor bicicleta: %s \n", moeda(mediaValorBike));
10:   }
11: }
```

```
1:  public class TesteCollect02 {
2:    public static void main(String[] args) {
3:       List<Produto> produtos = new ArrayList<>();
4:       produtos = BDProduto.getProdutos();
5:
6:       Map<String, List<Produto>> produtosPorTipo = produtos.stream()
7:             .collect(Collectors.groupingBy(p -> p.getTipo()));
8:
9:       produtosPorTipo.forEach((tipo, lista) -> {
10:         System.out.printf("\nTipo: %s \n", tipo);
11:         lista.stream()
12:               .forEach(p -> System.out.printf("\t%s\n", p.getDescricao()));
13:      });
14:   }
15: }
```

```
1:  public class TesteCollect03 {
2:    public static void main(String[] args) {
3:       List<Produto> produtos = new ArrayList<>();
4:       produtos = BDProduto.getProdutos();
5:
6:       var tipoProduto = produtos.stream()
7:             .map(p -> p.getTipo())
8:             .distinct()
9:             .collect(Collectors.joining(", "));
10:      System.out.printf("Tipos de produto: %s \n", tipoProduto);
11:   }
12: }
```

```
1:  public class TesteCollect04 {
2:    public static void main(String[] args) {
3:       List<Produto> produtos = new ArrayList<>();
4:       produtos = BDProduto.getProdutos();
5:       Map<Boolean, List<Produto>> acimaDe2000 = produtos.stream()
6:                .collect(Collectors.partitioningBy(p -> p.getValor() >= 1_000.00));
7:       acimaDe2000.forEach((tipo, lista) -> {
8:          System.out.println(tipo ? "acima de R$ 1000,00:" : "abaixo de R$ 1000,00:");
9:          lista.forEach(p -> System.out.printf("\t%s - %s\n",
10:               p.getDescricao(), moeda(p.getValor())));
11:      });
12:   }
13: }
```