

# Produtividade Java com biblioteca Google Guava

Guilherme de Cleva Farto  
Coffee and Code  
Agroindústria  
Assis, SP



TODOS OS DIREITOS RESERVADOS

○ Fevereiro  
2018



# Objetivo

Contextualizar sobre Google Guava

Apresentar pacotes e features essenciais

- Strings (Joiner, Splitter e CharMatcher)
- Preconditions
- Collections
  - Utilities (Type Inference)
  - Ordering
  - New types (Multimap e Multiset)
  - Iterable e FluentIterable

Exemplificar uso com código



# O QUE É **PRODUTIVIDADE?**

Entregar mais com maior qualidade?

Escrever menos código?

Código mais reusável?

Reducir ou eliminar redundâncias?

Consumir tempo com o que realmente importa?



# Testar se uma String possui valor

```
public boolean validate(String value) {  
    if (value != null && !"".equals(value)) { // ...  
  
    if (value != null && !value.isEmpty()) { // ...  
  
    if (value != null && !value.trim().isEmpty()) { // ...  
  
    if (value != null && value.length() > 0) { // ...  
  
}
```



# Testar se uma String possui valor

```
public boolean validate(String value) {  
  
    // Com Guava  
  
    if (!Strings.isNullOrEmpty(value)) { // ...  
  
}
```



## Filtrar listagem de fazendas pela quantidade de talhões

```
public static List<FazendaVO> filtrar(List<FazendaVO> fazendas,  
    int quantidadeTalhoes) {  
    List<FazendaVO> fazendasFiltradas =  
        new LinkedList<FazendaVO>();  
  
    if (ListUtil.isNotNullOrEmpty(fazendas)) {  
        for (FazendaVO fazenda : fazendas) {  
            if (ListUtil.isNotNullOrEmpty(fazenda.getTalhoes())  
                && fazenda.getTalhoes().size() >= quantidadeTalhoes) {  
                fazendasFiltradas.add(fazenda);  
            }  
        }  
    }  
  
    return fazendasFiltradas;  
}
```



# Filtrar listagem de fazendas pela quantidade de talhões

```
public static List<FazendaVO> filtrarIterator(List<FazendaVO> fazendas,  
    int quantidadeTalhoes) {  
    if (ListUtil.isNotNullOrEmpty(fazendas)) {  
        for (Iterator<FazendaVO> iFazenda = fazendas.iterator(); iFazenda.hasNext();) {  
            FazendaVO fazenda = iFazenda.next();  
  
            if (fazenda.getTalhoes().size() < quantidadeTalhoes) {  
                iFazenda.remove();  
            }  
        }  
  
        return fazendas;  
    }  
}
```



## Filtrar listagem de fazendas pela quantidade de talhões

```
public static List<FazendaVO> filtrar(List<FazendaVO> fazendas,  
    final int quantidadeTalhoes) {  
    return FluentIterable.from(fazendas)  
        .filter(new Predicate<FazendaVO>() {  
  
            @Override  
            public boolean apply(FazendaVO fazenda) {  
                return ListUtil.isNotNullOrEmpty(fazenda.getTalhoes())  
                    && fazenda.getTalhoes().size() >= quantidadeTalhoes;  
            }  
        }).toList();  
}
```



## Filtrar listagem de fazendas pela quantidade de talhões

```
public static List<FazendaVO> filtrar(List<FazendaVO> fazendas,  
    final int quantidadeTalhoes) {  
    return FluentIterable.from(fazendas)  
        .filter(fazenda -> ListUtil.isNotNullOrEmpty(fazenda.getTalhoes())  
            && fazenda.getTalhoes().size() >= quantidadeTalhoes)  
        .toList();  
}
```



**“Google Guava é um conjunto open source de bibliotecas e recursos criado e mantido por engenheiros e desenvolvedores da Google”**

- Bibliotecas e utilitários que são usados diariamente pela Google
- Testado constantemente, em produção, pela Google

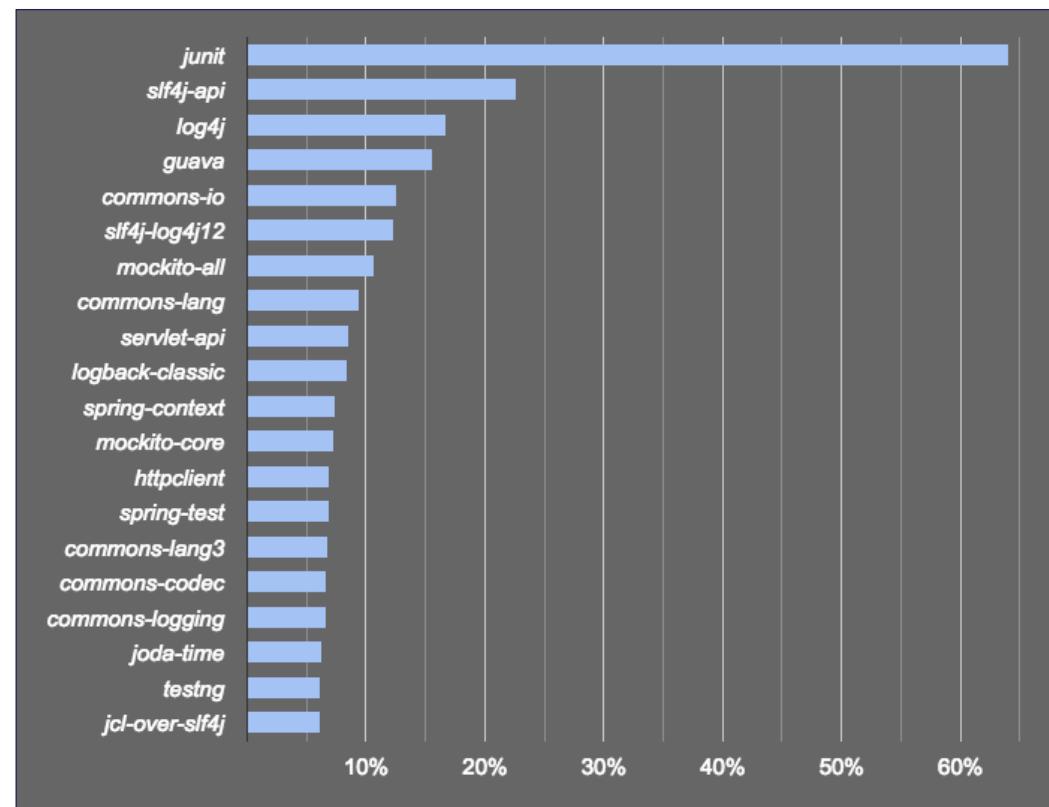
**“Em Julho de ’12, contava com + de 286.000 testes unitários”**

- Gerados, de maneira automatizada, com extensa cobertura de testes



**“Em ‘15, [Takipi](#) analisou 60.678 imports de 11.939 bibliotecas (unique) Java adotadas em 5.216 projetos Java no GitHub”**

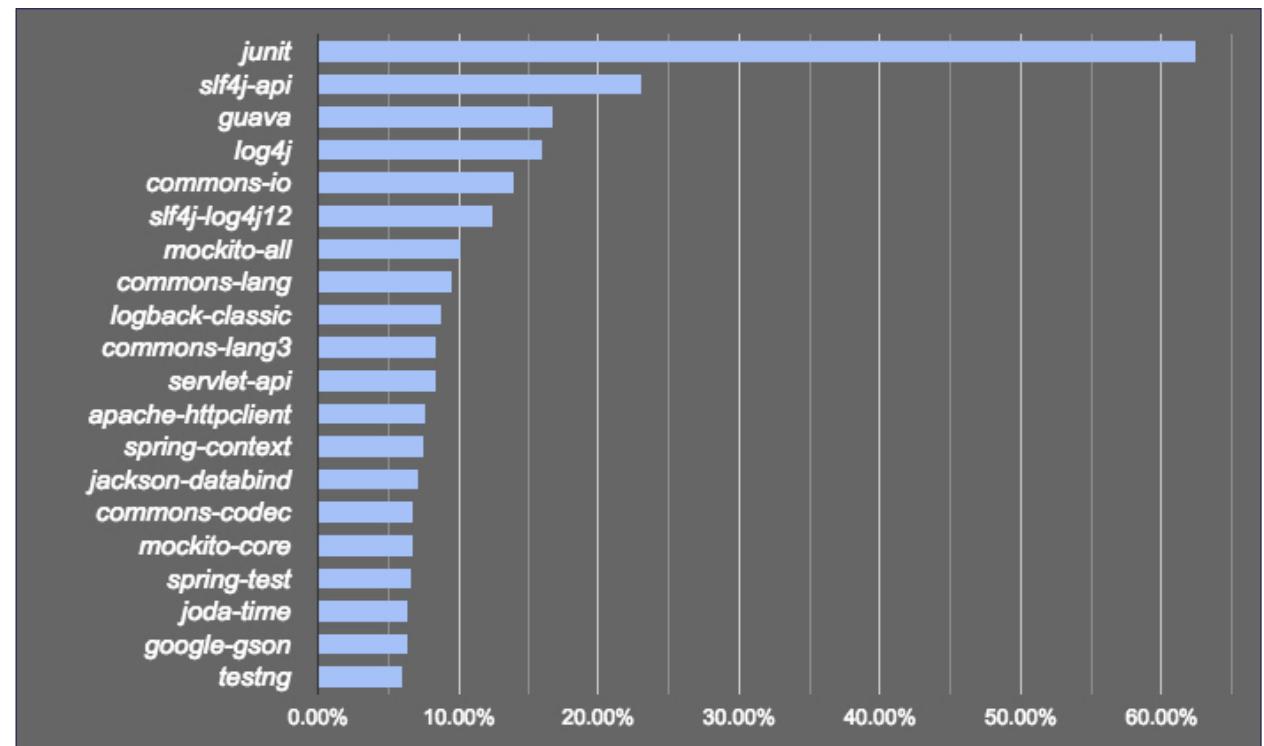
- Uso em 15,6% dos projetos
- Apache Commons com ~12%





**“Em ‘16, [Takipi](#) analisou 47.251 imports de 12.059 bibliotecas (unique) Java adotadas em 3.862 projetos Java no GitHub”**

- Guava sobe para 3<sup>a</sup> posição





## “Guava é composto por diversas outras *core libraries*”

- String utilities
- Basic utilities: Preconditions, Ordering, ...
- Collections: Immutable collections, New types, Collection utilities, ...
- Functional utilities: Uses of functional programming
  
- Optional<?>, Graphs, Caches, Concurrency,
- Primitives, Ranges, I/O e Files
- Hashing, Math, Reflection, ...
  
- Mais informações: <https://github.com/google/guava/wiki>



# Dependência Maven



```
<dependency>
    <groupId>com.google.guava</groupId>
    <artifactId>guava</artifactId>
    <version>20.0</version> <!-- for <= jdk 6 -->
</dependency>

.
.
.
<dependency>
    <groupId>com.google.guava</groupId>
    <artifactId>guava</artifactId>
    <version>21.0</version> <!-- if >= 21.0, requires jdk 8 -->
</dependency>

<!-- starting with 22.0, has an Android flavor -->
```



# Strings

**Joiner**

**Splitter**

**CharMatcher**



## STRINGS JOINER

```
String[] culturas = { "Cana-de-Açúcar", null, "Soja", null, "Algodão" };

String info = Joiner.on(", ").skipNulls().join(culturas);

System.out.println(info);

/*
    .join( Object[] parts      )
    .join( Iterable<?> parts  )
    .join( Iterator<?> parts  )
    .join( Object... parts    )
*/
```

Cana-de-Açúcar, Soja, Algodão



## STRINGS JOINER

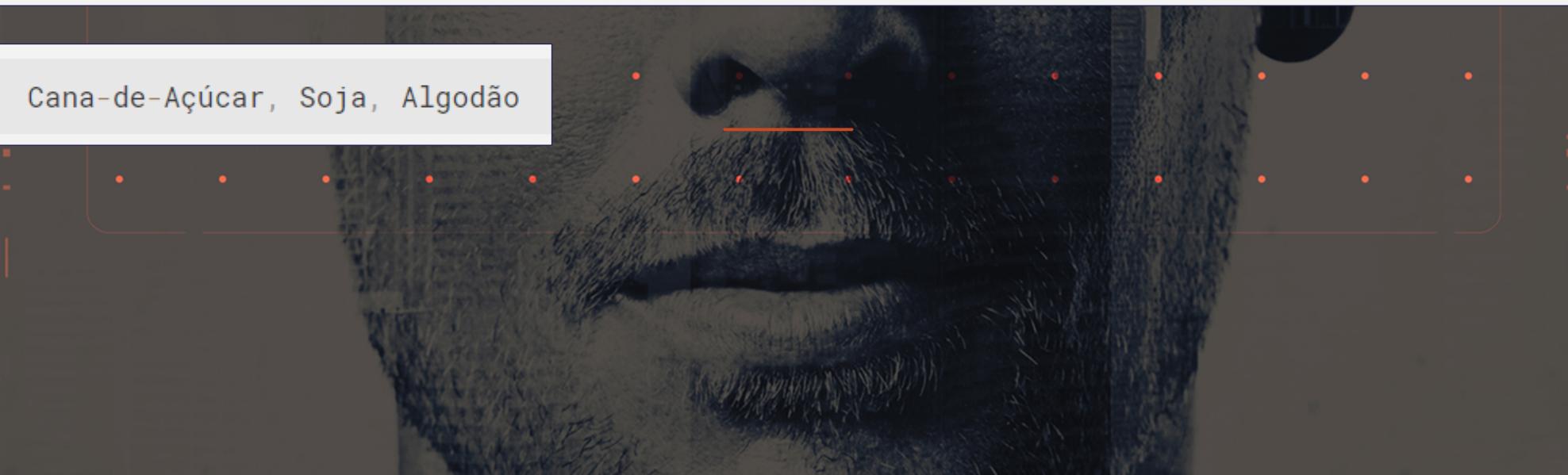
18



```
List<String> culturas = Arrays.asList("Cana-de-Açúcar", "Soja", "Algodão");

String info = Joiner.on(", ").skipNulls().join(culturas);

System.out.println(info);
```



Cana-de-Açúcar, Soja, Algodão



## STRINGS JOINER

19



```
List<String> culturas = Arrays.asList("Cana-de-Açúcar", "Soja", "Algodão");

StringBuilder result = new StringBuilder("");

result.append("Culturas: ");

Joiner.on(", ").skipNulls().appendTo(result, culturas);

System.out.println(result);
```

Culturas: Cana-de-Açúcar, Soja, Algodão



## STRINGS JOINER

20



```
Map<String, String> data = new LinkedHashMap<String, String>();
data.put("Fazenda #1", "Cana-de-Açúcar");
data.put("Fazenda #2", "Soja");
data.put("Fazenda #3", "Algodão");

String info = Joiner.on(", ").withKeyValueSeparator(" = ").join(data);

System.out.println(info);
```

```
Fazenda #1 = Cana-de-Açúcar, Fazenda #2 = Soja, Fazenda #3 = Algodão
```



## STRINGS JOINER

```
String[] operacoes = { "1000", "1002", null, "1005", "1008" };

String where = Joiner.on(", ").skipNulls().join(operacoes);

String sql = "... WHERE CD_OPERACAO IN ( " + where + " ) ...";

System.out.println(sql);
```

```
... WHERE CD_OPERACAO IN ( 1000, 1002, 1005, 1008 ) ...
```



## STRINGS JOINER

22



```
List<String> operacoes = Arrays.asList("1000", "1002", null, "1005", "1008");

String where = Joiner.on(" OR ").skipNulls()
    .join(Iterables.transform(operacoes,
        new Function<String, String>() {

            public String apply(String value) {
                return "CD_OPERACAO = " + value;
            }
        }));
    });

String sql = "... WHERE " + where + "...";

System.out.println(sql);
```

... WHERE CD\_OPERACAO = 1000 OR CD\_OPERACAO = 1002 OR CD\_OPERACAO = null OR



## STRINGS SPLITTER

```
String info = "Cana-de-Açúcar, , Soja, , Algodão";  
  
Iterable<String> culturas = Splitter.on(" , ").omitEmptyStrings().split(info);  
  
System.out.println(culturas);
```

[Cana-de-Açúcar, Soja, Algodão]



## STRINGS SPLITTER

```
String info = "Cana-de-Açúcar, Soja. Algodão|Milho^Café";  
  
List<String> culturas = Splitter.onPattern(", . | ^").omitEmptyStrings()  
    .splitToList(info);  
  
System.out.println(culturas.size());  
System.out.println(culturas);
```

```
5
```

```
[Cana-de-Açúcar, Soja, Algodão, Milho, Café]
```



## STRINGS SPLITTER

25



```
String info = "Fazenda #1 = Cana-de-Açúcar, Fazenda #2 = Soja, Fazenda #3 = Algodão";  
  
Map<String, String> fazendas = Splitter.on(" ", ).withKeyValueSeparator(" = ")  
    .split(info);  
  
System.out.println(fazendas.size());  
System.out.println(fazendas.keySet());  
System.out.println(fazendas.values());  
System.out.println(fazendas);
```

3

```
[Fazenda #1, Fazenda #2, Fazenda #3]  
[Cana-de-Açúcar, Soja, Algodão]  
{Fazenda #1=Cana-de-Açúcar, Fazenda #2=Soja, Fazenda #3=Algodão}
```



## STRINGS SPLITTER

```
String info = "Guava Workshop - TOTVS Coffee and Code";  
  
List<String> result = Splitter.fixedLength(5).splitToList(info);  
  
System.out.println(result.size());  
System.out.println(result);
```

8

[Guava, Work, shop , - TOT, VS Co, ffee , and C, ode]



## STRINGS SPLITTER

```
String info = "Cana-de-Açúcar, Soja, Algodão, Milho, Café";  
  
List<String> culturas = Splitter.on(", ").limit(3).splitToList(info);  
  
System.out.println(culturas.size());  
  
for (String cultura : culturas) {  
    System.out.println(cultura);  
}
```

3  
Cana-de-Açúcar  
Soja  
Algodão, Milho, Café



## // STRINGS CHARMATCHER

```
String COFFEE_AND_CODE = "    Coffee and Code    ";

String result = CharMatcher.javaLowerCase().retainFrom(COFFEE_AND_CODE);

System.out.println("[ " + result + " ]");

result = CharMatcher.javaLowerCase().removeFrom(COFFEE_AND_CODE);

System.out.println("[ " + result + " ]");
```

```
[offeeandode]
[   c   c   ]
```



## // STRINGS CHARMATCHER

```
String COFFEE_AND_CODE = "    Coffee and Code    ";

String result = CharMatcher.javaUpperCase().retainFrom(COFFEE_AND_CODE);

System.out.println("[ " + result + " ]");

result = CharMatcher.javaUpperCase().removeFrom(COFFEE_AND_CODE);

System.out.println("[ " + result + " ]");
```

```
[cc]
[    offee and ode    ]
```



## // STRINGS CHARMATCHER

```
String COFFEE_AND_CODE = "    Coffee and Code    ";

String result = CharMatcher.whitespace().trimLeadingFrom(COFFEE_AND_CODE);

System.out.println("[ " + result + " ]");

result = CharMatcher.whitespace().trimTrailingFrom(COFFEE_AND_CODE);

System.out.println("[ " + result + " ]");

result = CharMatcher.whitespace().trimFrom(COFFEE_AND_CODE);

System.out.println("[ " + result + " ]");
```

```
[Coffee and Code    ]
[    Coffee and Code]
[Coffee and Code]
```



## // STRINGS CHARMATCHER

```
String COFFEE_AND_CODE = "    Coffee and Code    ";

String result = CharMatcher.whitespace().collapseFrom(COFFEE_AND_CODE, '-');

System.out.println("[ " + result + " ]");

result = CharMatcher.whitespace().trimAndCollapseFrom(COFFEE_AND_CODE, '-');

System.out.println("[ " + result + " ]");
```

```
[-Coffee-and-Code-]
[Coffee-and-Code]
```



## STRINGS CHARMATCHER

```
String COFFEE_AND_CODE_2018 = " * Coffee and Code 2018 * ";

System.out.println(CharMatcher.digit().countIn(COFFEE_AND_CODE_2018));
System.out.println(CharMatcher.javaLetter().countIn(COFFEE_AND_CODE_2018));
System.out.println(CharMatcher.javaLetterOrDigit().countIn(COFFEE_AND_CODE_2018));
System.out.println(CharMatcher.ascii().countIn(COFFEE_AND_CODE_2018));
System.out.println(CharMatcher.is('*').countIn(COFFEE_AND_CODE_2018));

4   System.out.println(CharMatcher.digit().retainFrom(COFFEE_AND_CODE_2018));
13  System.out.println(CharMatcher.javaLetter().retainFrom(COFFEE_AND_CODE_2018));
17  System.out.println(CharMatcher.javaLetterOrDigit().retainFrom(COFFEE_AND_CODE_2018));
26  System.out.println(CharMatcher.ascii().retainFrom(COFFEE_AND_CODE_2018));
2    
```

4  
13  
17  
26  
2  
2018

CoffeeandCode  
CoffeeandCode2018  
\* Coffee and Code 2018 \*



## STRINGS CHARMATCHER

```
String COFFEE_AND_CODE_2018 = " * Coffee and Code 2018 * ";

String result = CharMatcher.is('*').or(CharMatcher.is(' ')).or(CharMatcher.is('C')))
    .removeFrom(COFFEE_AND_CODE_2018);

System.out.println(result);

result = CharMatcher.forPredicate(new Predicate<Character>() {

    @Override
    public boolean apply(Character c) {
        return c == '*' || c == ' ' || c == 'C';
    }
}).removeFrom(COFFEE_AND_CODE_2018);

System.out.println(result);
```

offeeandode2018  
offeeandode2018



## STRINGS CHARMATCHER

```
String COFFEE_AND_CODE_2018 = "coffee@and^code";  
  
String result = CharMatcher.is('@').replaceFrom(COFFEE_AND_CODE_2018, " ");  
  
System.out.println(result);  
  
result = CharMatcher.isNot('@').replaceFrom(COFFEE_AND_CODE_2018, " ");  
  
System.out.println(result);  
  
result = CharMatcher.is('@').or(CharMatcher.is('^')).replaceFrom(COFFEE_AND_CODE_2018, " ");  
  
System.out.println(result);  
  
result = CharMatcher.anyOf("@^").replaceFrom(COFFEE_AND_CODE_2018, " ");  
  
System.out.println(result);  
  
result = CharMatcher.noneOf("@^").replaceFrom(COFFEE_AND_CODE_2018, ".");  
  
System.out.println(result);
```

```
coffee and^code  
 @  
coffee and code  
coffee and code  
.....@....^....
```



## // STRINGS CHARMATCHER

```
String COFFEE_AND_CODE_2018 = "coffee@and^code";  
  
int result = CharMatcher.is('@').indexIn(COFFEE_AND_CODE_2018);  
  
System.out.println(result);  
  
result = CharMatcher.is('c').lastIndexIn(COFFEE_AND_CODE_2018);  
  
System.out.println(result);
```

6

11



# Preconditions



## PRECONDITIONS

```
public static void inserir(EquipamentoVO equipamento) {  
    Preconditions.checkNotNull(equipamento);  
    Preconditions.checkNotNull(equipamento, "Equipamento é obrigatório.");  
  
    Preconditions.checkNotNull(equipamento.getCodigo(),  
        "Código do equipamento é obrigatório.");  
  
    // throws IllegalArgumentException  
    Preconditions.checkArgument(TipoEquipmentoEnum.PROPRIO.equals(equipamento.getTipo()),  
        "Equipamento %s não é do tipo Próprio.", equipamento.getCodigo());  
  
    // throws IllegalStateException  
    Preconditions.checkState(equipamento.isDisponivel(),  
        "Equipamento %s não está Disponível.", equipamento.getCodigo());  
  
    // Preconditions.checkElementIndex(index, 10);  
    // Preconditions.checkPositionIndex(index, 10);  
  
    EquipamentoDAO.inserir(equipamento);  
}
```



# Collections Utilities

## // COLLECTIONS UTILITIES

39

```
// Without Guava (< JDK 7)
List<EquipamentoVO> equipamentos1 = new LinkedList<EquipamentoVO>();

// Without Guava (>= JDK 7)
List<EquipamentoVO> equipamentos2 = new LinkedList<>();

// List with Guava
// Lists.newArrayList, Lists.newLinkedList
List<EquipamentoVO> equipamentosListGuava = Lists.newLinkedList();

// Map with Guava
// Maps.newHashMap, Maps.newLinkedHashMap, Maps.newTreeMap
Map<EquipamentoVO, Date> equipamentosMapGuava = Maps.newLinkedHashMap();
```



# Functional Programming

**Uso em**

- Ordering
- Collections
- FluentIterable



### Function<F, T>

- “como transformar F em T”
  - `T apply(F input)`
- Usado, e.g., na transformação de instâncias e coleções (listas)

### Predicate<F>

- “validar se `true` ou `false` dado F”
  - `boolean apply(F input)`
- Usado, e.g., no filtro (pesquisa) de coleções (listas)



# Ordering

## ORDERING

43

```
Ordering<FazendaV0> ordering = Ordering.natural().nullsFirst()
    .onResultOf(new Function<FazendaV0, Long>() {
        @Override
        public Long apply(FazendaV0 fazenda) {
            return NumberUtil.asLong(fazenda.getCodigo());
        }
    });
System.out.println(ordering.isOrdered(fazendas));
List<FazendaV0> fazendasOrdenadas = ordering.sortedCopy(fazendas);
```

```
true
10 - Fazenda #10
20 - Fazenda #20
30 - Fazenda #30
40 - Fazenda #40
50 - Fazenda #50
60 - Fazenda #60
70 - Fazenda #70
80 - Fazenda #80
90 - Fazenda #90
100 - Fazenda #100
```

# ORDERING

44

```
Ordering<FazendaVO> ordering = Ordering.natural().nullsFirst().reverse()  
    .onResultOf(new Function<FazendaVO, Long>() {
```

```
    @Override  
    public Long apply(FazendaVO fazenda) {  
        return NumberUtil.asLong(fazenda.getCodigo());  
    }  
});
```

```
System.out.println(ordering.isOrdered(fazendas));
```

```
List<FazendaVO> fazendasOrdenadas = ordering.sortedCopy(fazendas);
```

```
false  
100 - Fazenda #100  
90 - Fazenda #90  
80 - Fazenda #80  
70 - Fazenda #70  
60 - Fazenda #60  
50 - Fazenda #50  
40 - Fazenda #40  
30 - Fazenda #30  
20 - Fazenda #20  
10 - Fazenda #10
```



## ORDERING

45



```
Ordering<FazendaVO> ordering = Ordering.natural().nullsFirst()
    .onResultOf(new Function<FazendaVO, Long>() {
        @Override
        public Long apply(FazendaVO fazenda) {
            return NumberUtil.asLong(fazenda.getCodigo());
        }
    });

List<FazendaVO> fazendasOrdenadas = ordering.immutableSortedCopy(fazendas);

fazendasOrdenadas.remove(0); // Exception!
```

```
Exception in thread "main" java.lang.UnsupportedOperationException
    at com.google.common.collect.ImmutableList.remove(ImmutableList.java:466)
    at com.totvs.guavaworkshop.coffeeandcode.collections.ordering.examples.Example03.main(Example03.java:27)
```

# ORDERING

46

```
Ordering<FazendaV0> ordering = Ordering.from(new Comparator<FazendaV0>() {  
  
    @Override  
    public int compare(FazendaV0 f1, FazendaV0 f2) {  
        return Ints.compare(f1.getTalhoes().size(), f2.getTalhoes().size());  
    }  
}).reverse();  
  
List<FazendaV0> fazendasOrdenadas = ordering.sortedCopy(fazendas);
```

20 - Fazenda #20 - 8 talhões - 7 equipamentos  
30 - Fazenda #30 - 8 talhões - 10 equipamentos  
80 - Fazenda #80 - 8 talhões - 14 equipamentos  
50 - Fazenda #50 - 7 talhões - 11 equipamentos  
60 - Fazenda #60 - 7 talhões - 12 equipamentos  
70 - Fazenda #70 - 6 talhões - 7 equipamentos  
100 - Fazenda #100 - 5 talhões - 12 equipamentos  
10 - Fazenda #10 - 4 talhões - 10 equipamentos



## ORDERING

47



```
Ordering<FazendaVO> ordering = Ordering.from(new Comparator<FazendaVO>() {  
  
    @Override  
    public int compare(FazendaVO f1, FazendaVO f2) {  
        return Ints.compare(f1.getTalhoes().size(), f2.getTalhoes().size());  
    }  
});  
  
FazendaVO maiorFazenda = ordering.max(fazendas);  
  
FazendaVO menorFazenda = ordering.min(fazendas);
```

20 - Fazenda #20 - 8 talhões - 7 equipamentos  
40 - Fazenda #40 - 3 talhões - 8 equipamentos

# ORDERING

48

```
Ordering<FazendaV0> ordering = Ordering.from(new Comparator<FazendaV0>() {  
  
    @Override  
    public int compare(FazendaV0 f1, FazendaV0 f2) {  
        return Ints.compare(f1.getTalhoes().size(), f2.getTalhoes().size());  
    }  
});  
  
List<FazendaV0> maioresFazendas = ordering.greatestOf(fazendas, 3);  
  
List<FazendaV0> menoresFazendas = ordering.leastOf(fazendas, 3);
```

20 - Fazenda #20 - 8 talhões - 7 equipamentos  
30 - Fazenda #30 - 8 talhões - 10 equipamentos  
80 - Fazenda #80 - 8 talhões - 14 equipamentos  
40 - Fazenda #40 - 3 talhões - 8 equipamentos  
10 - Fazenda #10 - 4 talhões - 12 equipamentos  
90 - Fazenda #90 - 4 talhões - 7 equipamentos

# // ORDERING

49

```
Ordering<FazendaV0> ordering = Ordering.from(new Comparator<FazendaV0>() {  
  
    @Override  
    public int compare(FazendaV0 f1, FazendaV0 f2) {  
        return Ints.compare(f1.getTalhoes().size(), f2.getTalhoes().size());  
    }  
}).reverse();  
  
Ordering<FazendaV0> orderingEquipamentos = Ordering.from(new Comparator<FazendaV0>() {  
  
    @Override  
    public int compare(FazendaV0 f1, FazendaV0 f2) {  
        return Ints.compare(f1.getEquipamentos().size(), f2.getEquipamentos().size());  
    }  
}).reverse();  
  
ordering = ordering.compound(orderingEquipamentos);  
  
List<FazendaV0> fazendasOrdenadas = ordering.sortedCopy(fazendas);
```



## ORDERING

50

- •   •   80 - Fazenda #80 - 8 talhões - 14 equipamentos
- •   •   30 - Fazenda #30 - 8 talhões - 10 equipamentos
- •   •   20 - Fazenda #20 - 8 talhões - 7 equipamentos
- •   •   60 - Fazenda #60 - 7 talhões - 12 equipamentos
- •   •   50 - Fazenda #50 - 7 talhões - 11 equipamentos
- •   •   70 - Fazenda #70 - 6 talhões - 7 equipamentos
- •   •   100 - Fazenda #100 - 5 talhões - 12 equipamentos
- •   •   10 - Fazenda #10 - 4 talhões - 12 equipamentos
- •   •   90 - Fazenda #90 - 4 talhões - 7 equipamentos
- •   •   40 - Fazenda #40 - 3 talhões - 8 equipamentos



## ORDERING

51

```
Ordering<FazendaVO> ordering = Ordering.from(new Comparator<FazendaVO>() {  
  
    @Override  
    public int compare(FazendaVO f1, FazendaVO f2) {  
        ComparisonChain chain = ComparisonChain.start();  
  
        chain = chain.compare(f1.getTalhoes().size(), f2.getTalhoes().size());  
        chain = chain.compare(f1.getEquipamentos().size(), f2.getEquipamentos().size());  
  
        return chain.result();  
    }  
}).reverse();  
  
List<FazendaVO> fazendasOrdenadas = ordering.sortedCopy(fazendas);
```



## ORDERING

52

- • • 80 - Fazenda #80 - 8 talhões - 14 equipamentos
- • • 30 - Fazenda #30 - 8 talhões - 10 equipamentos
- • • 20 - Fazenda #20 - 8 talhões - 7 equipamentos
- • • 60 - Fazenda #60 - 7 talhões - 12 equipamentos
- • • 50 - Fazenda #50 - 7 talhões - 11 equipamentos
- • • 70 - Fazenda #70 - 6 talhões - 7 equipamentos
- • • 100 - Fazenda #100 - 5 talhões - 12 equipamentos
- • • 10 - Fazenda #10 - 4 talhões - 12 equipamentos
- • • 90 - Fazenda #90 - 4 talhões - 7 equipamentos
- • • 40 - Fazenda #40 - 3 talhões - 8 equipamentos

## // ORDERING

53

```
Ordering<FazendaVO> ordering = Ordering.from(new Comparator<FazendaVO>() {  
  
    @Override  
    public int compare(FazendaVO f1, FazendaVO f2) {  
        ComparisonChain chain = ComparisonChain.start();  
  
        chain = chain.compare(f1.getTalhoes().size(), f2.getTalhoes().size());  
        chain = chain.compare(f1.getEquipamentos().size(), f2.getEquipamentos().size(),  
            Ordering.natural().reverse());  
  
        return chain.result();  
    }  
}).reverse();  
  
List<FazendaVO> fazendasOrdenadas = ordering.sortedCopy(fazendas);
```

# ORDERING

54

- • • 20 - Fazenda #20 - 8 talhões - 7 equipamentos
- • • 30 - Fazenda #30 - 8 talhões - 10 equipamentos
- • • 80 - Fazenda #80 - 8 talhões - 14 equipamentos
- • • 50 - Fazenda #50 - 7 talhões - 11 equipamentos
- • • 60 - Fazenda #60 - 7 talhões - 12 equipamentos
- • • 70 - Fazenda #70 - 6 talhões - 7 equipamentos
- • • 100 - Fazenda #100 - 5 talhões - 12 equipamentos
- • • 90 - Fazenda #90 - 4 talhões - 7 equipamentos
- • • 10 - Fazenda #10 - 4 talhões - 12 equipamentos
- • • 40 - Fazenda #40 - 3 talhões - 8 equipamentos



# Collections New Types

**Multimap**

**Multiset**



## COLLECTIONS NEW TYPES MULTIMAP

```
public static void main(String[] args) {  
  
    Map<String, List<String>> fazendasCulturas = new LinkedHashMap<String, List<String>>();  
  
    fazendasCulturas = adicionar(fazendasCulturas, "1000", "Algodão");  
    fazendasCulturas = adicionar(fazendasCulturas, "1000", "Soja");  
    fazendasCulturas = adicionar(fazendasCulturas, "1000", "Milho");  
    fazendasCulturas = adicionar(fazendasCulturas, "2000", "Cana-de-Açúcar");  
    fazendasCulturas = adicionar(fazendasCulturas, "2000", "Café");  
  
    System.out.println(fazendasCulturas);  
  
}
```

```
{1000=[Algodão, Soja, Milho], 2000=[Cana-de-Açúcar, Café]}
```



## COLLECTIONS NEW TYPES MULTIMAP

```
public static Map<String, List<String>> adicionar(Map<String, List<String>> fazendasCulturas,
    String fazenda, String cultura) {
    if (fazendasCulturas == null) {
        fazendasCulturas = new LinkedHashMap<String, List<String>>();
    }

    List<String> culturas = null;

    if (fazendasCulturas.containsKey(fazenda)) {
        culturas = fazendasCulturas.get(fazenda);
    } else {
        culturas = new LinkedList<String>();
    }

    culturas.add(cultura);

    fazendasCulturas.put(fazenda, culturas);

    return fazendasCulturas;
}
```



## COLLECTIONS NEW TYPES

### MULTIMAP

```
public static void main(String[] args) {  
  
    // ArrayListMultimap, HashMultimap, LinkedListMultimap, LinkedHashMultimap,  
    // TreeMultimap, ImmutableListMultimap, ImmutableSetMultimap  
ListMultimap<String, String> fazendasCulturas =  
    LinkedListMultimap.create();  
  
    adicionar(fazendasCulturas, "1000", "Algodão");  
    adicionar(fazendasCulturas, "1000", "Soja");  
    adicionar(fazendasCulturas, "1000", "Milho");  
    adicionar(fazendasCulturas, "2000", "Cana-de-Açúcar");  
    adicionar(fazendasCulturas, "2000", "Café");  
  
    System.out.println(fazendasCulturas.size());  
    System.out.println(fazendasCulturas);  
    System.out.println(fazendasCulturas.get("1000"));  
    System.out.println(fazendasCulturas.keySet());  
  
    Map<String, List<String>> asMap = Multimaps.asMap(fazendasCulturas);  
  
    System.out.println(asMap);  
  
}
```



## COLLECTIONS NEW TYPES MULTIMAP

```
public static void adicionar(Multimap<String, String> fazendasCulturas,  
    String fazenda, String cultura) {  
    fazendasCulturas.put(fazenda, cultura);  
}
```

```
5  
{1000=[Algodão, Soja, Milho], 2000=[Cana-de-Açúcar, Café]}  
[Algodão, Soja, Milho]  
[1000, 2000]  
{1000=[Algodão, Soja, Milho], 2000=[Cana-de-Açúcar, Café]}
```



# COLLECTIONS NEW TYPES

## MULTISET

```
public static void main(String[] args) {  
  
    Map<String, Integer> pragas = new LinkedHashMap<String, Integer>();  
  
    • •  
    adicionar(pragas, "Broca-da-cana");  
    adicionar(pragas, "Carrapicho");  
    adicionar(pragas, "Carrapicho");  
    adicionar(pragas, "Broca-da-cana");  
    adicionar(pragas, "Cigarrinha-das-raizes");  
    adicionar(pragas, "Cupim");  
    adicionar(pragas, "Broca-da-cana");  
    adicionar(pragas, "Cupim");  
    • •  
    System.out.println(pragas);  
  
    System.out.println(pragas.get("Broca-da-cana")); // 3  
    // remover "Broca-da-cana" ???  
    System.out.println(pragas.get("Besouro")); // null  
  
    System.out.println(pragas.keySet());  
}
```

```
{Broca-da-cana=3, Carrapicho=2, Cigarrinha-das-raizes=1, Cupim=2}  
3  
null  
[Broca-da-cana, Carrapicho, Cigarrinha-das-raizes, Cupim]
```



## COLLECTIONS NEW TYPES MULTISET

```
public static void adicionar(Map<String, Integer> pragas, String praga) {  
    Integer contadorPragas = pragas.get(praga);  
  
    if (contadorPragas == null) {  
        pragas.put(praga, 1);  
    } else {  
        pragas.put(praga, contadorPragas + 1);  
    }  
}
```



## COLLECTIONS NEW TYPES MULTISET

```
public static void main(String[] args) {  
  
    // HashMultiset, TreeMultiset, LinkedHashMultiset,  
    // ConcurrentHashMultiset, ImmutableMultiset  
    Multiset<String> pragas = HashMultiset.create();  
  
    adicionar(pragas, "Broca-da-cana");  
    adicionar(pragas, "Carrapicho");  
    adicionar(pragas, "Carrapicho");  
    adicionar(pragas, "Broca-da-cana");  
    adicionar(pragas, "Cigarrinha-das-raízes");  
    adicionar(pragas, "Cupim");  
    adicionar(pragas, "Broca-da-cana");  
    adicionar(pragas, "Cupim");  
  
    System.out.println(pragas.size());  
    System.out.println(pragas);  
  
    System.out.println(pragas.count("Broca-da-cana")); // 3  
    pragas.remove("Broca-da-cana");  
    System.out.println(pragas.count("Broca-da-cana")); // 2
```



## COLLECTIONS NEW TYPES MULTISET

```
System.out.println(pragas.count("Besouro")); // 0
System.out.println(pragas.contains("Besouro")); // false

pragas.setCount("Besouro", 5);

System.out.println(pragas.count("Besouro")); // 0
System.out.println(pragas.contains("Besouro")); // false

System.out.println(pragas.elementSet()); // like keySet() for a Map

}

public static void adicionar(Multiset<String> pragas, String praga) {
    pragas.add(praga);
}
```



## COLLECTIONS NEW TYPES MULTISET

- 8 [Cupim x 2, Cigarrinha-das-raízes, Broca-da-cana x 3, Carrapicho x 2]
- 3
- 2
- 0
- false
- 5
- true
- [Besouro, Cupim, Cigarrinha-das-raízes, Broca-da-cana, Carrapicho]



# Iterable e FluentIterable

## ITERABLE

66

```
List<EquipamentoV0> equipamentos1 = Lists.newArrayList(  
    Arrays.asList(new EquipamentoV0("1000"), new EquipamentoV0("3000")));  
List<EquipamentoV0> equipamentos2 = Lists.newArrayList(  
    Arrays.asList(new EquipamentoV0("2000"), new EquipamentoV0("4000")));  
List<EquipamentoV0> equipamentos3 = Lists.newArrayList(  
    Arrays.asList(new EquipamentoV0("4000"), new EquipamentoV0("1000")));  
  
Iterable<EquipamentoV0> equipamentos = Iterables.concat(equipamentos1, equipamentos2,  
    equipamentos3);  
  
System.out.println(Iterables.size(equipamentos));  
System.out.println(equipamentos);  
  
// Requires equals() for EquipamentoV0  
System.out.println(Iterables.frequency(equipamentos, new EquipamentoV0("1000")));  
  
Iterable<List<EquipamentoV0>> equipamentosParticionados = Iterables.partition(equipamentos, 3);  
  
System.out.println(Iterables.size(equipamentosParticionados));
```

```
List<EquipamentoV0> equipamentosParticao = Iterables.get(equipamentosParticionados, 0);  
  
EquipamentoV0 equipamentoPrimeiro = Iterables.getFirst(equipamentos, null);  
EquipamentoV0 equipamentoUltimo = Iterables.getLast(equipamentos, null);  
  
Iterable<EquipamentoV0> equipamentosLimite = Iterables.limit(equipamentos, 4);  
  
// NoSuchElementException - if the iterable is empty  
// IllegalArgumentException - if the iterable contains multiple elements  
EquipamentoV0 equipamentoUnico = Iterables.getOnlyElement(equipamentos);
```

```
6  
[EquipamentoV0 [codigo=1000, modelo=null, grupoOperativo=null, tipo=null, ultimoHorimetro=null,  
2  
2  
Exception in thread "main" java.lang.IllegalArgumentException: expected one element but was: <E  
at com.google.common.collect.Iterators.getOnlyElement(Iterators.java:322)  
at com.google.common.collect.Iterables.getOnlyElement(Iterables.java:294)  
at com.totvs.guavaworkshop.coffeeandcode.collections.iterables.examples.Example01.main(Examp
```

## // FLUENTITERABLE

68

```
public static void main(String[] args) {  
    List<FazendaVO> fazendas = // ...  
  
    List<FazendaVO> fazendasFiltradas = FluentIterable.from(fazendas)  
        .filter(FILTER_FAZENDAS_MAIOR_IGUAL_7_TALHOES)  
        .filter(FILTER_FAZENDAS_MENOR_IGUAL_10_EQUIPAMENTOS).toList();  
}
```

## // FLUENTITERABLE

69

```
public static final Predicate<FazendaVO> FILTER_FAZENDAS_MAIOR_IGUAL_7_TALHOES =  
    new Predicate<FazendaVO>() {  
  
        @Override  
        public boolean apply(FazendaVO fazenda) {  
            return ListUtil.isNotNullOrEmpty(fazenda.getTalhoes())  
                && fazenda.getTalhoes().size() >= 7;  
        }  
    };
```

```
public static final Predicate<FazendaVO> FILTER_FAZENDAS_MENOR_IGUAL_10_EQUIPAMENTOS =  
    new Predicate<FazendaVO>() {  
  
        @Override  
        public boolean apply(FazendaVO fazenda) {  
            return ListUtil.isNotNullOrEmpty(fazenda.getEquipamentos())  
                && fazenda.getEquipamentos().size() <= 10;  
        }  
    };
```

2

[FazendaVO [codigo=20, descricao=Fazenda #20],  
 FazendaVO [codigo=30, descricao=Fazenda #30]]

# // FLUENTITERABLE

70

```
List<FazendaVO> fazendas = ...  
  
List<FazendaVO> fazendasFiltradas = FluentIterable.from(fazendas)  
    .filter(Predicates.and(  
        FILTER_FAZENDAS_MAIOR_IGUAL_7_TALHOES, FILTER_FAZENDAS_MENOR_IGUAL_10_EQUIPAMENTOS))  
    .toList();  
  
fazendasFiltradas = FluentIterable.from(fazendas)  
    .filter(Predicates.or(  
        FILTER_FAZENDAS_MAIOR_IGUAL_7_TALHOES, FILTER_FAZENDAS_MENOR_IGUAL_10_EQUIPAMENTOS))  
    .toList();
```

```
2  
[FazendaVO [codigo=20, descricao=Fazenda #20],  
 FazendaVO [codigo=30, descricao=Fazenda #30]]  
8  
[FazendaVO [codigo=20, descricao=Fazenda #20],  
 FazendaVO [codigo=30, descricao=Fazenda #30], ...]
```

## // FLUENTITERABLE

71

```
public static final Function<FazendaVO, String> FUNCTION_CODIGO_FAZENDA =
    new Function<FazendaVO, String>() {

    @Override
    public String apply(FazendaVO fazenda) {
        return fazenda.getCodigo();
    }
};

public static void main(String[] args) {
    List<FazendaVO> fazendas = // ...

    ImmutableListMultimap<String, FazendaVO> index = FluentIterable.from(fazendas)
        .filter(Predicates.and(
            FILTER_FAZENDAS_MAIOR_IGUAL_7_TALHOES, FILTER_FAZENDAS_MENOR_IGUAL_10_EQUIPAMENTOS))
        .index(FUNCTION_CODIGO_FAZENDA);

    System.out.println(index.size());
    System.out.println(index);
    System.out.println(index.keys());
}
```

2

{20=[FazendaVO [codigo=20, descricao=Fazenda #20]],  
30=[FazendaVO [codigo=30, descricao=Fazenda #30]]}  
[20, 30]

# // FLUENTITERABLE

72

```
public static final Predicate<FazendaV0> FILTER_FAZENDAS_TALHAO_ALGODAO =
    new Predicate<FazendaV0>() {

        @Override
        public boolean apply(FazendaV0 fazenda) {
            ImmutableListMultimap<String, TalhaoV0> talhoesCulturas =
                Multimaps.index(fazenda.getTalhoes(), new Function<TalhaoV0, String>() {

                    @Override
                    public String apply(TalhaoV0 talhao) {
                        return talhao.getCultura().getDescricao();
                    }
                });
            return talhoesCulturas.containsKey("Algodão");
            // return talhoesCulturas.containsKey("Algodão")
            // && talhoesCulturas.get("Algodão").size() >= 3;
        }
   };
```

9

```
[FazendaV0 [codigo=10, descricao=Fazenda #10],
 FazendaV0 [codigo=30, descricao=Fazenda #30], ...]
```

# // FLUENTITERABLE

73

```
List<FazendaV0> fazendasFiltradas = FluentIterable.from(fazendas)
    .filter(FILTER_FAZENDAS_TALHAO_ALGODAO).transform(new Function<FazendaV0, FazendaV0>() {

    @Override
    public FazendaV0 apply(FazendaV0 fazenda) {
        ImmutableList<TalhaoV0> talhoesFiltrados =
            FluentIterable.from(fazenda.getTalhoes())
                .filter(new Predicate<TalhaoV0>() {

                    @Override
                    public boolean apply(TalhaoV0 talhao) {
                        return "Algodão".equals(talhao.getCultura().getDescricao());
                    }
                }).toList();

        fazenda.clearTalhoes();
        fazenda.addTalhoes(talhoesFiltrados);

        return fazenda;
    }
}).toList();

System.out.println(fazendasFiltradas.size());
System.out.println(fazendasFiltradas);
```



## FLUENTITERABLE

```
9  
[FazendaVO [codigo=10, descricao=Fazenda #10], FazendaVO [codigo=30,  
10  
    1010 - CulturaVO [codigo=3000, descricao=Algodão] - 170.90  
30  
    1100 - CulturaVO [codigo=3000, descricao=Algodão] - 54.42  
    1110 - CulturaVO [codigo=3000, descricao=Algodão] - 101.58  
40  
    1060 - CulturaVO [codigo=3000, descricao=Algodão] - 31.22  
    1070 - CulturaVO [codigo=3000, descricao=Algodão] - 147.90  
50  
    1030 - CulturaVO [codigo=3000, descricao=Algodão] - 32.06  
    1130 - CulturaVO [codigo=3000, descricao=Algodão] - 62.26
```

# // FLUENTITERABLE

75

```
ImmutableList<TalhaoV0> talhoesFiltrados =
    FluentIterable.from(fazendas)
        .filter(FILTER_FAZENDAS_TALHAO_ALGODAO)
        .transformAndConcat(new Function<FazendaV0, Iterable<TalhaoV0>>() {

            @Override
            public Iterable<TalhaoV0> apply(FazendaV0 fazenda) {
                ImmutableList<TalhaoV0> talhoesFiltrados = FluentIterable
                    .from(fazenda.getTalhoes()).filter(new Predicate<TalhaoV0>() {

                        @Override
                        public boolean apply(TalhaoV0 talhao) {
                            return "Algodão".equals(talhao.getCultura().getDescricao());
                        }
                    })
                    .toList();

                return talhoesFiltrados;
            }
        }).toList();
```

```
20
[TalhaoV0 [codigo=1010, cultura=CulturaV0 [codigo=3000, descricao=Algodão],
1010 - CulturaV0 [codigo=3000, descricao=Algodão] - 170.90
1100 - CulturaV0 [codigo=3000, descricao=Algodão] - 54.42
1110 - CulturaV0 [codigo=3000, descricao=Algodão] - 101.58
1060 - CulturaV0 [codigo=3000, descricao=Algodão] - 31.22
1070 - CulturaV0 [codigo=3000, descricao=Algodão] - 147.90
1030 - CulturaV0 [codigo=3000, descricao=Algodão] - 32.06
1130 - CulturaV0 [codigo=3000, descricao=Algodão] - 62.26
```



```
ImmutableList<TalhaoVO> talhoesFiltrados = FluentIterable.from(fazendas)
    .filter(FILTER_FAZENDAS_TALHAO_ALGODAO)
    .transformAndConcat(new Function<FazendaVO, Iterable<TalhaoVO>>() {

        @Override
        public Iterable<TalhaoVO> apply(FazendaVO fazenda) {
            // ...
        }
    }).limit(3).toList();
```

3

```
[TalhaoVO [codigo=1010, cultura=CulturaVO [codigo=3000, descricao=Algodão],
 1010 - CulturaVO [codigo=3000, descricao=Algodão] - 170.90
 1100 - CulturaVO [codigo=3000, descricao=Algodão] - 54.42
 1110 - CulturaVO [codigo=3000, descricao=Algodão] - 101.58
```

## // FLUENTITERABLE

77

```
ImmutableList<TalhaoVO> talhoesFiltrados = FluentIterable.from(fazendas)
    .filter(FILTER_FAZENDAS_TALHAO_ALGODAO)
    .transformAndConcat(TRANSFORM_TALHAO_ALGODAO).toSortedList(new Comparator<TalhaoVO>() {
        @Override
        public int compare(TalhaoVO t1, TalhaoVO t2) {
            return t1.getAreaProdutiva().compareTo(t2.getAreaProdutiva());
        }
    }).reverse();
```

20

```
[TalhaoVO [codigo=1190, cultura=CulturaVO [codigo=3000, descricao=Algodão],  
1190 - CulturaVO [codigo=3000, descricao=Algodão] - 229.47  
1200 - CulturaVO [codigo=3000, descricao=Algodão] - 210.74  
1180 - CulturaVO [codigo=3000, descricao=Algodão] - 197.64  
1080 - CulturaVO [codigo=3000, descricao=Algodão] - 173.34  
1010 - CulturaVO [codigo=3000, descricao=Algodão] - 170.90  
1170 - CulturaVO [codigo=3000, descricao=Algodão] - 154.92  
1070 - CulturaVO [codigo=3000, descricao=Algodão] - 147.90
```

## // FLUENTITERABLE

78

```
ImmutableList<TalhaoVO> talhoesFiltrados = FluentIterable.from(fazendas)
    .filter(FILTER_FAZENDAS_TALHAO_ALGODAO)
    .transformAndConcat(TRANSFORM_TALHAO_ALGODAO).toSortedList(new Comparator<TalhaoVO>() {

        @Override
        public int compare(TalhaoVO t1, TalhaoVO t2) {
            return t1.getAreaProdutiva().compareTo(t2.getAreaProdutiva());
        }
    }).reverse().subList(0, 3);
```

3

```
[TalhaoVO [codigo=1190, cultura=CulturaVO [codigo=3000, descricao=Algodão],
 1190 - CulturaVO [codigo=3000, descricao=Algodão] - 229.47
 1200 - CulturaVO [codigo=3000, descricao=Algodão] - 210.74
 1180 - CulturaVO [codigo=3000, descricao=Algodão] - 197.64
```



## “Where are Fold, Reduce, Aggregator, ... ?”

- Não há pretensão em tornar o Guava uma biblioteca completa de programação funcional
- Java 8 já possui muitas features neste sentido e podem se complementar com Guava

## Projeto “AgriDataGenerator”

- Gerador de dados agrícolas, usado nos testes com Guava
- Versão atual: Fazenda, Talhão, Equipamento e Inspeção Fitossanitária
- Pode ser usado nos próximos C&C como gerador de datasets

## Próximos passos

- Praticar as conceitos apresentados com outras necessidades



# Dúvidas?



# Referências e Links

User Guide

<https://github.com/google/guava/wiki>

Philosophy Explained

<https://github.com/google/guava/wiki/PhilosophyExplained>

Guava Compatibility

<https://github.com/google/guava/wiki/Compatibility>

Apache Commons Collections Equivalents

<https://github.com/google/guava/wiki/ApacheCommonCollectionsEquivalents>

# OBRIGADO



**GUILHERME DE CLEVA FARTO**

Pesquisa e Inovação – TOTVS Agroindústria

guilherme.farto@totvs.com.br

[totvs.com](https://www.facebook.com/totvs)

[company/totvs](https://www.linkedin.com/company/totvs)

[@totvs](https://twitter.com/totvs)

[fluig.com](https://fluig.com)