Java 8 na prática

Bruno Yokio Tatsumi
Coffee and Code
Agroindústria
Assis, SP

Março
2018

Objetivos

1. Por que Java 8?
2. Nova API de Data
3. Optional
4. Functional Interface
5. Lambda
6. Method Reference
7. Melhorias nas API's de Collections e Maps
8. Stream

Por que Java 8?

- Produtividade (fazer mais com menos)
- Legibilidade = Qualidade
- Apresentação dos recursos e funcionalidades prevendo futuras migrações

02

java.time.*

Vantagens:

- Fabricação estática
- Fluent API / Builder
- Granularidade de objetos
- Métodos utilitários

Static factory & Fluent API / Builder

```
String dataHora = LocalDate
        .of(2018, 1, 1)
        .atStartOfDay()
        .format(DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss"))
        .toString();
```

Tipos de objetos

```java
/* Objeto com dia, mês e ano */
LocalDate dataAtual = LocalDate.now();

/* Objeto com dia, mês, ano e hora */
LocalDateTime dataHoraAtual = LocalDateTime.now();

/* Objeto com dia e mês */
MonthDay diaMesAtual = MonthDay.now();

/* Objeto com ano e mês */
YearMonth anoMesAtual = YearMonth.now();

/* Objeto com ano */
Year anoAtual = Year.now();
```

Operações úteis

```
public boolean isDataColheitaPermitidaAposAplicacaoInsumos(
        AplicacaoInsumoVO aplicacaoInsumos,
        LocalDate dataColheita) {

    LocalDate dataLimite = aplicacaoInsumos.getDataOperacao()
            .plusDays(DIAS_APOS_APLICACAO_PERMITIDO_COLHER);

    return dataColheita.isAfter(dataLimite);
}
```

Operações úteis

```java
/* Objeto com dia, mês e ano */
LocalDate dataAtual = LocalDate.now();
LocalDate dataPosterior = dataAtual.plusDays(10L);

/* Objeto com dia, mês, ano e hora */
LocalDateTime dataHoraAtual = LocalDateTime.now();
LocalDateTime dataHoraPosterior = dataHoraAtual.plus(5L, ChronoUnit.HOURS);

/* Objeto com dia e mês */
MonthDay diaMesAtual = MonthDay.now();
LocalDate diaMesAtualAno2018 = diaMesAtual.atYear(2018);

/* Objeto com ano e mês */
YearMonth anoMesAtual = YearMonth.now();
LocalDate finalDoMes = anoMesAtual.atEndOfMonth();

/* Objeto com ano */
Year anoAtual = Year.now();
LocalDate natal2018 = anoAtual.atMonthDay(MonthDay.of(12, 25));
```

03

java.util.Optional

- Similar ao Optional do Guava
- Evitar comparações com null
- Encadeamento de chamadas nullSafe
- Métodos utilitários

Optional


```
// code smell
if (talhao != null && talhao.getFazenda() != null) {
    return talhao.getFazenda().getCodigo() != null ? talhao.getFazenda().getCodigo() : "";
}
return "";
```



```
/* Sem comparação com nulos */
```

```
/* Encadeamento de chamadas nullSafe */
```

```
/* Fluent API */
```

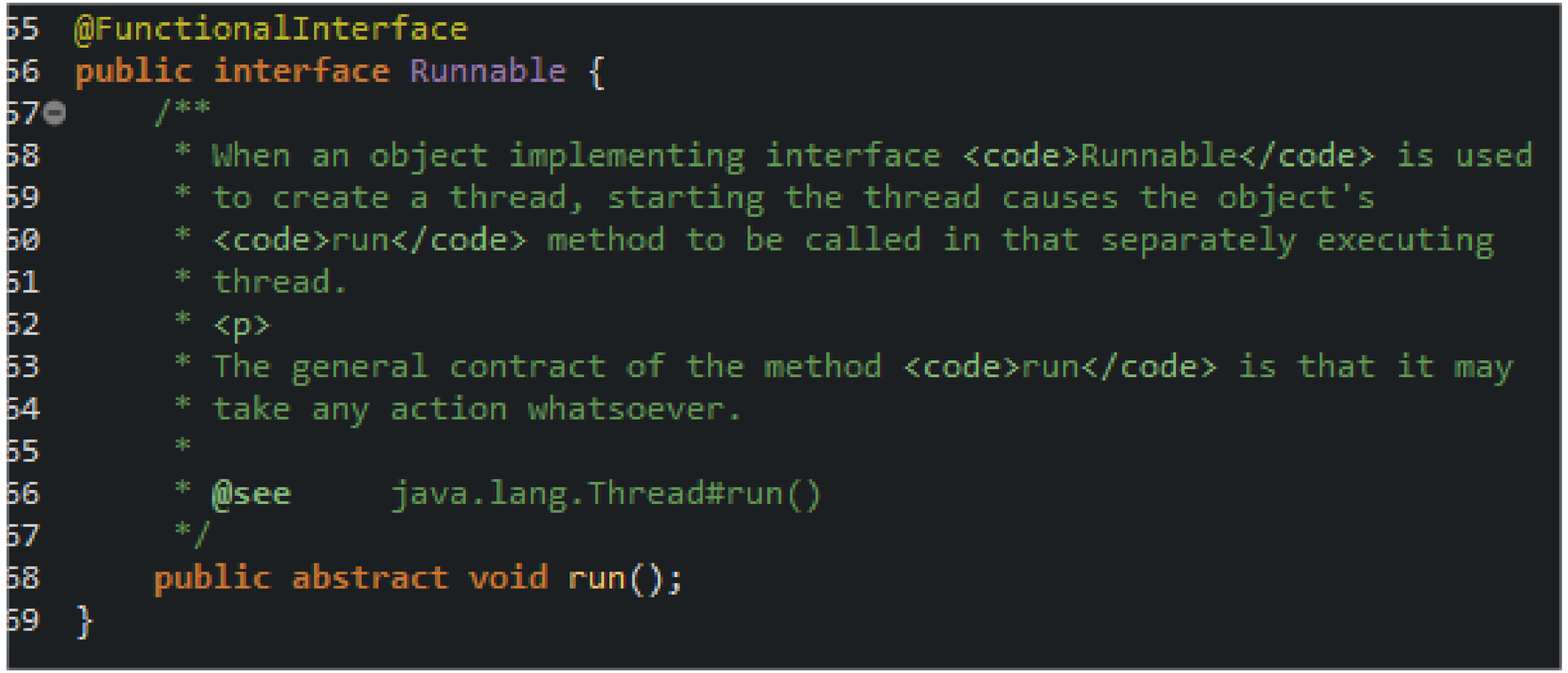
```
    return Optional.ofNullable(talhao)
                .map(TalhaoVO::getFazenda)
                .map(FazendaVO::getCodigo)
                .orElse("");
```

04

Functional Interface

Interface com apenas um método

Functional Interface



```java
@FunctionalInterface
public interface Runnable {
    /**
     * When an object implementing interface <code>Runnable</code> is used
     * to create a thread, starting the thread causes the object's
     * <code>run</code> method to be called in that separately executing
     * thread.
     * <p>
     * The general contract of the method <code>run</code> is that it may
     * take any action whatsoever.
     *
     * @see     java.lang.Thread#run()
     */
    public abstract void run();
}
```

Functional Interface


```
Runnable runnable = new Runnable() {

    @Override
    public void run() {
        System.out.println("Coffee & Code Agro!");
    }
};
```

```
Runnable runnable = () -> System.out.println("Coffee & Code Agro!");
```

05

λ Lambda

Lambda – Function (dado uma entrada, gera uma saída)


```java
/**
 * Represents a function that accepts one argument and produces a result.
 *
 * <p>This is a <a href="package-summary.html">functional interface</a>
 * whose functional method is {@link #apply(Object)}.
 *
 * @param <T> the type of the input to the function
 * @param <R> the type of the result of the function
 *
 * @since 1.8
 */
@FunctionalInterface
public interface Function<T, R> {

    /**
     * Applies this function to the given argument.
     *
     * @param t the function argument
     * @return the function result
     */
    R apply(T t);
```

Lambda – Predicate (presente no Guava)

```java
/**
 * Represents a predicate (boolean-valued function) of one argument.
 *
 * <p>This is a <a href="package-summary.html">functional interface</a>
 * whose functional method is {@link #test(Object)}.
 *
 * @param <T> the type of the input to the predicate
 *
 * @since 1.8
 */
@FunctionalInterface
public interface Predicate<T> {

    /**
     * Evaluates this predicate on the given argument.
     *
     * @param t the input argument
     * @return {@code true} if the input argument matches the predicate,
     * otherwise {@code false}
     */
    boolean test(T t);
```

Lambda – Exemplo de Function


```
//Lambda expression que converte TalhaoVO em FazendaVO
Function<TalhaoVO, FazendaVO> talhaoParaFazenda = talhao -> talhao.getFazenda();

FazendaVO fazenda = new FazendaVO("1", "Fazenda 1");
TalhaoVO talhao = new TalhaoVO("1", fazenda, BigDecimal.TEN);

FazendaVO fazendaDoTalhao = talhaoParaFazenda.apply(talhao);
```

Lambda – Exemplo de Predicate

```java
private static void exemploPredicate() {
    Predicate<TalhaoVO> talhaoPossuiAreaDisponivel = talhao -> !talhao.isSemAreaDisponivel();
    Predicate<TalhaoVO> talhaoPossuiFazenda = talhao -> Objects.nonNull(talhao.getFazenda());

    FazendaVO fazenda = new FazendaVO("1", "Fazenda 1");
    TalhaoVO talhao = new TalhaoVO("1", fazenda, BigDecimal.TEN);

    boolean talhaoPossuiAreaEFazenda = talhaoPossuiAreaDisponivel
            .and(talhaoPossuiFazenda)
            .test(talhao);

    System.out.println(talhaoPossuiAreaEFazenda);
}
```

06

Method Reference

Method Reference

```
static Function<TalhaoVO, Long> FUNCTION_ID_TALHAO_LAMBDA = talhao -> talhao.getId();
```

```
static Function<TalhaoVO, Long> FUNCTION_ID_TALHAO = TalhaoVO::getId;
```

07

Collections e Maps

- forEach
- removeIf
- merge

forEach

```
/**
 * Performs the given action for each element of the {@code Iterable}
 * until all elements have been processed or the action throws an
 * exception.  Unless otherwise specified by the implementing class,
 * actions are performed in the order of iteration (if an iteration order
 * is specified).  Exceptions thrown by the action are relayed to the
 * caller.
 *
 * @implSpec
 * <p>The default implementation behaves as if:
 * <pre>{@code
 *     for (T t : this)
 *         action.accept(t);
 * }</pre>
 *
 * @param action The action to be performed for each element
 * @throws NullPointerException if the specified action is null
 * @since 1.8
 */
default void forEach(Consumer<? super T> action) {
    Objects.requireNonNull(action);
    for (T t : this) {
        action.accept(t);
    }
}
```

Collection.forEach

```java
public static void insertTalhoesLambda(Collection<TalhaoVO> talhoes) {
    TalhaoDAO talhaoDAO = new TalhaoDAO();
    //iteração utilizando lambda
    talhoes.forEach(t -> talhaoDAO.insert(t));
}
```

```java
public static void insertTalhoesMethodReference(Collection<TalhaoVO> talhoes) {
    TalhaoDAO talhaoDAO = new TalhaoDAO();
    //utilizando method reference
    talhoes.forEach(talhaoDAO::insert);
}
```

Collection.removeIf

```java
/**
 * Removes all of the elements of this collection that satisfy the given
 * predicate.  Errors or runtime exceptions thrown during iteration or by
 * the predicate are relayed to the caller.
 *
 * @implSpec
 * The default implementation traverses all elements of the collection using
 * its {@link #iterator}.  Each matching element is removed using
 * {@link Iterator#remove()}.  If the collection's iterator does not
 * support removal then an {@code UnsupportedOperationException} will be
 * thrown on the first matching element.
 *
 * @param filter a predicate which returns {@code true} for elements to be
 *        removed
 * @return {@code true} if any elements were removed
 * @throws NullPointerException if the specified filter is null
 * @throws UnsupportedOperationException if elements cannot be removed
 *         from this collection.  Implementations may throw this exception if a
 *         matching element cannot be removed or if, in general, removal is not
 *         supported.
 * @since 1.8
 */
default boolean removeIf(Predicate<? super E> filter) {
    Objects.requireNonNull(filter);
    boolean removed = false;
    final Iterator<E> each = iterator();
    while (each.hasNext()) {
        if (filter.test(each.next())) {
            each.remove();
            removed = true;
        }
    }
    return removed;
}
```

Collection.removeIf

```
talhoes.removeIf(t -> t.getArea().compareTo(BigDecimal.ZERO) == 0);
TalhaoDAO talhaoDAO = new TalhaoDAO();
talhoes.forEach(talhaoDAO::insert);
```

```
//remove talhões sem área disponível
talhoes.removeIf(TalhaoVO::isSemAreaDisponivel);
TalhaoDAO talhaoDAO = new TalhaoDAO();
//insere no banco de dados os talhões que sobrarams
talhoes.forEach(talhaoDAO::insert);
```

Map.merge

```java
public static Map<TalhaoVO, BigDecimal> agruparAreasApontadasPorTalhoes(Collection<AplicacaoInsumoVO> apontamentos) {
    Map<TalhaoVO, BigDecimal> areasApontamentosPorTalhao = new HashMap<>();

    for (AplicacaoInsumoVO aplicacaoInsumo : apontamentos) {
        areasApontamentosPorTalhao.merge(
                aplicacaoInsumo.getTalhao(),
                aplicacaoInsumo.getAreaAplicada(),
                BigDecimal::add);
    }

    return areasApontamentosPorTalhao;
}
```

08

Stream

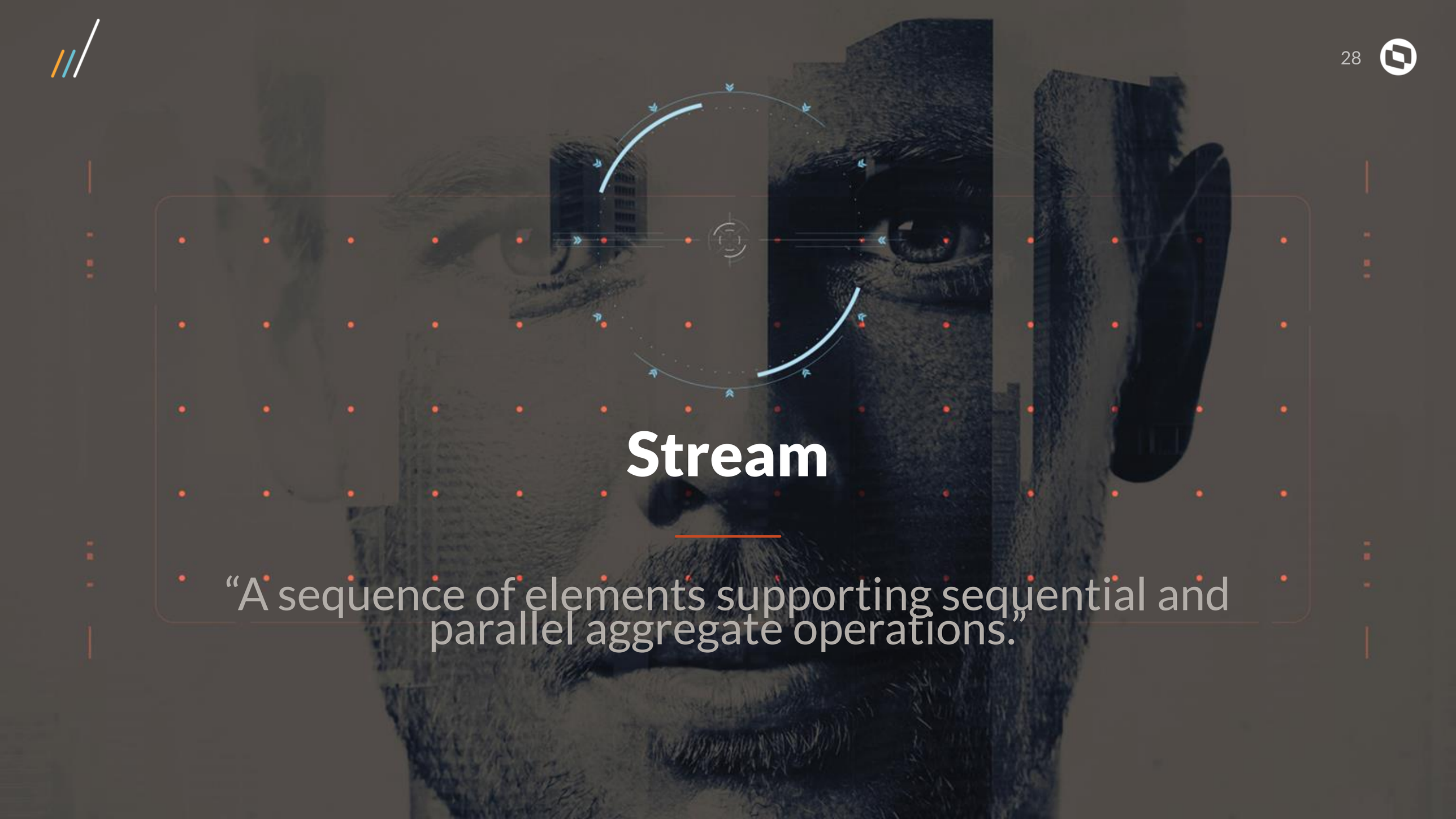
filter

map

collect

reduce

...



Stream

"A sequence of elements supporting sequential and parallel aggregate operations."

Stream.filter

```
public static Stream<AplicacaoInsumoVO> filtrarApontamentosPorTalhao(
        Collection<AplicacaoInsumoVO> apontamentos,
        TalhaoVO talhao) {

    return apontamentos.stream()
            .filter(apontamento -> apontamento.getTalhao().equals(talhao));
}
```

```
public static Stream<AplicacaoInsumoVO> filtrarApontamentosPorTalhoesComAreaDisponivel(
        Collection<AplicacaoInsumoVO> apontamentos) {

    return apontamentos.stream()
            .filter(apontamento -> apontamento.getTalhao().possuiAreaDisponivel());
}
```

Stream.map

```java
public static Stream<TalhaoVO> getTalhoesStream(Collection<AplicacaoInsumoVO> apontamentos) {
    /* Transforma uma Stream de apontamentos em uma nova Stream de talhões */
    Stream<TalhaoVO> talhoesStream = apontamentos.stream()
            .map(AplicacaoInsumoVO::getTalhao);
    return talhoesStream;
}
```

Stream.collect

```java
public static List<TalhaoVO> getTalhoesApontamentos(Collection<AplicacaoInsumoVO> apontamentos) {
    List<TalhaoVO> talhoes = apontamentos.stream()
            .map(AplicacaoInsumoVO::getTalhao)
            .collect(Collectors.toList());
    return talhoes;
}
```

Stream – encadeamento de chamadas


```
public static List<TalhaoVO> getTalhoesApontamentosSemAreaDisponivel(Collection<AplicacaoInsumoVO> apontamentos) {
    return apontamentos.stream()
            .filter(apontamento -> apontamento.getTalhao().isSemAreaDisponivel())
            .map(AplicacaoInsumoVO::getTalhao)
            .distinct()
            .collect(Collectors.toList());
}
```

Stream.reduce

```
public static BigDecimal getSomaAreasApontadasPorTalhao(
        Collection<AplicacaoInsumoVO> apontamentos,
        TalhaoVO talhao) {

    return apontamentos.stream()
            .filter(apontamento -> apontamento.getTalhao().equals(talhao))
            .map(AplicacaoInsumoVO::getAreaAplicada)
            .reduce(BigDecimal::add)
            .orElse(BigDecimal.ZERO);
}
```

Stream.anyMatch


```java
public static boolean isAlgumTalhaoSemAreaDisponivel(Collection<AplicacaoInsumoVO> apontamentos) {
    return apontamentos.stream().anyMatch(apontamento -> apontamento.getTalhao().isSemAreaDisponivel());
}
```

Stream.allMatch


```
public static boolean isTodosTalhoesSemAreaDisponivel(Collection<AplicacaoInsumoVO> apontamentos) {
    return apontamentos.stream().allMatch(apontamento -> apontamento.getTalhao().isSemAreaDisponivel());
}
```

Stream.noneMatch

```java
public static boolean isNenhumTalhaoSemAreaDisponivel(Collection<AplicacaoInsumoVO> apontamentos) {
    return apontamentos.stream().noneMatch(apontamento -> apontamento.getTalhao().isSemAreaDisponivel());
}
```

Stream – mais collectors

```java
public static Map<LocalDate, List<AplicacaoInsumoVO>> agruparApontamentosPorData(Collection<AplicacaoInsumoVO> apontamentos) {
    return apontamentos.stream()
            .collect(Collectors.groupingBy(AplicacaoInsumoVO::getDataOperacao));
}
```

Stream – mais collectors


```java
public static Map<LocalDate, Long> contarApontamentosPorData(Collection<AplicacaoInsumoVO> apontamentos) {
    return apontamentos.stream()
            .collect(Collectors.groupingBy(AplicacaoInsumoVO::getDataOperacao, Collectors.counting()));
}
```

Stream – mais collectors


```java
public static Map<LocalDate, Double> agruparMediaAreaAplicadaPorData(Collection<AplicacaoInsumoVO> apontamentos) {
    return apontamentos.stream()
            .collect(groupingBy(AplicacaoInsumoVO::getDataOperacao, averagingDouble(AplicacaoInsumoVO::getAreaAplicadaAsDouble)));
}
```

Java 8 – antes e depois (códigos reais)

```
if (!listTalhoes.isEmpty() && listTalhoes.size() > 0){
    String talhoes = "";
    int numTalhoes = listTalhoes.size();
    for(int i=0; i< listTalhoes.size(); i++){
        talhoes += listTalhoes.get(i).getIdUPNivel3() ;
        if(i+1 < numTalhoes){
            talhoes += ", ";
        }
    }
    preencheCamposSelect(talhoes);
}
```

Java 8 – antes e depois (códigos reais)


```
String talhoes = listTalhoes.stream()
    .map(ColheitaLocalVO::getIdUPNivel3)
    .map(String::valueOf)
    .collect(Collectors.joining(", "));
preencheCamposSelect(talhoes);
```

Java 8 – antes e depois (códigos reais)

```
BigDecimal qtRequisitada = BigDecimal.ZERO;

for (OrdemServicoCampoInsumoLocalDetalheVO detalhe : insLocal.getDetalhes()) {
    if (detalhe.getQtdeTotal() != null) {
        qtRequisitada = qtRequisitada.add(detalhe.getQtdeTotal());
    }
}
```

Java 8 – antes e depois (códigos reais)

```
BigDecimal qtRequisitada = insLocal.getDetalhes().stream()
    .map(o -> o.getQtdeTotal())
    .filter(Objects::nonNull)
    .reduce(BigDecimal::add)
    .orElse(BigDecimal.ZERO);
```

Java 8 – antes e depois (códigos reais)


```
Iterator it = list.iterator();
while (it.hasNext()) {
    TipoApontamentoVO vo = (TipoApontamentoVO) it.next();

    Collection<CamposTipoApontamentoVO> collectionCamposTipoApontamento = Validate.validate(
            vo.getCamposTipoApontamentoVO(), vo, con, termos, termosGerais);

    if (collectionCamposTipoApontamento != null) {
        CamposTipoApontamentoDAO.Insert.insert(collectionCamposTipoApontamento, con, vo.getId(), termos);
    }
}
```

Java 8 – antes e depois (códigos reais)

```
list.stream()
    .map(tipoApto -> Validate.validate(tipoApto.getCamposTipoApontamentoVO(), vo, con, termos, termosGerais))
    .filter(Objects::nonNull)
    .forEach(camposTipoApto -> CamposTipoApontamentoDAO.Insert.insert(camposTipoApto, con, vo.getId(), termos))
```

OBRIGADO

Bruno Yokio Tatsumi

Desenvolvimento Agroindústria

bruno.tatsumi@totvs.com.br

totvs.com

company/totvs

@totvs

fluig.com