

Programação III

Semestre de Inverno de 2022-2023

2º Trabalho prático

Data de Entrega: 14 de dezembro de 2022

OBJETIVO: Praticar a utilização de *streams* de texto, do mecanismo de tratamento de exceções, da construção de tipos e métodos genéricos, as interfaces funcionais e expressões lambda, e os contentores da *framework* de coleções do JAVA.

NOTA: Para os métodos assinalados com [T] têm que constar os testes unitários que permitem validar a sua correção.

Grupo 1

Realize os seguintes métodos na classe **StreamsUtils**

1. [T] Realize o método público estático, com a seguinte assinatura:

```
public static boolean validate( Reader in ) throws IOException
```

que recebendo por parâmetro um *stream* de texto, contendo código fonte na linguagem Java, retorna **true** se para todo o carácter ‘}’ existiu anteriormente um carácter ‘{’ e se para todo o carácter ‘{’ existe posteriormente um carácter ‘}’, caso contrário **false**. Considere que os literais do tipo **String** e os comentários não contêm chavetas.

2. [T] Realize um método com a seguinte assinatura:

```
public static void copyCom( BufferedReader in, PrintWriter out ) throws IOException
```

que recebendo por parâmetro um *stream* de texto, contendo código fonte na linguagem Java, copia para o *stream* de texto **out** os comentários de linha indicando em que linha ocorrem. Assume que nem os literais do tipo *string* nem os comentários de bloco contêm a sequência de caracteres que define o início de um comentário de linha (`"/"`).

3. Realize os seguintes métodos estáticos públicos:

```
a) <R> void mapper( BufferedReader in,
                    Function<String, R> mapping,
                    BiConsumer<String, R> action ) throws IOException
```

[T] Que por cada linha lida do *stream in* execute a ação binária **action** passando por parâmetro a linha e o resultado da operação **mapping** sobre a linha. Para obter o mapeamento duma linha deve ser chamado o método **apply** sobre o objeto função **mapping** passando-lhe por parâmetro a linha. Executar a ação binária é chamar o método **accept** passando dois parâmetros: a linha lida e *string* que resultou do mapeamento.

b) **Integer evaluate(String expression)**

[T] Que retorna o resultado da avaliação da expressão aritmética recebida por parâmetro. Uma expressão aritmética é constituída, por um carácter dígito seguido de uma sequência de zero ou mais pares (operador, carácter dígito), seguidos do carácter ‘=’. Os operadores são o ‘+’ ou ‘-’. Caso a expressão esteja incorreta retorna **null**;

```
<expression> ::= <digit> {<operator> <digit>}* '='
<operator> ::= '+' | '-'
```

c) **void evaluate(String filenameIn, BiConsumer<String,Integer> action) throws IOException**

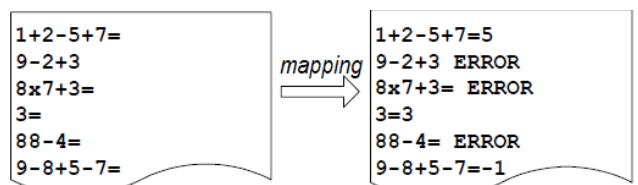
Que recebendo por parâmetro um ficheiro de texto com nome **filenameIn**, em que cada linha contém uma expressão aritmética, execute a **action** sobre cada linha do ficheiro e o resultado do cálculo da expressão.

Utilize o método da alínea a) e implemente um **Function<String, Integer>**.

d) **void copyEvaluate(String filenameIn, String filenameOut) throws IOException**

Que recebendo por parâmetro um ficheiro de texto com nome **filenameIn**, em que cada linha contém uma expressão aritmética, escreva no ficheiro de texto com nome **filenameOut** as expressões com o resultado do cálculo.

Utilize o método da alínea c) e implemente um **BiConsumer<String,Integer>**.



e) `void copyEvaluate(BufferedReader in, Writer out) throws IOException`

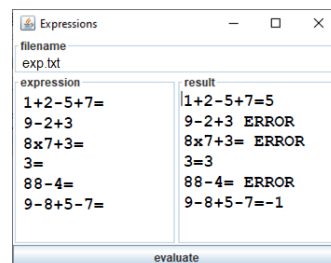
Que recebendo por parâmetro o *stream* de texto *in* em que cada linha contém uma expressão aritmética, escreva no *stream* de texto *out* a expressão com o resultado do cálculo.

Utilize o método da alínea a) e implemente uma `Function<String, Integer>` e um `BiConsumer<String, Integer>`

f) `String stringEvaluate(String expression)`

[T] Que recebendo por parâmetro uma *string* contendo uma expressão retorne uma *string* com a expressão seguida do resultado do cálculo. Utilize o método da alínea e).

4. Faça uma aplicação com o aspeto da figura. Quando for premido o botão “*evaluate*” chama o método `evaluate` da alínea b), para atualizar as duas áreas de texto “*expression*” com as linhas do ficheiro, e “*result*” com a expressão e o resultado do cálculo. Em caso de existirem exceções deve ser escrito na área de texto “*result*” a mensagem associada à exceção lançada.



Grupo 2

1. Realize, na classe `AlgorithmUtils`, os seguintes métodos estáticos públicos

a) `<E> boolean isOrdered(Collection<E> seq, Comparator<E> compareValue)`

[T] Que retorna `true` se a série estiver ordenada de forma crescente ou decrescente segundo o comparador `compareValue`, caso contrário `false`.

b) `<E> List<E> getSubSequences(Collection<E> seq, int n, Comparator<E> cmp)`

[T] Que produza uma lista com os valores da *n*ésima subsequência ordenada, de forma crescente segundo o comparador `cmp`, existentes na sequência `seq`. Retorna uma lista vazia caso a sequência não contenha `n` subsequências. Lança a exceção de *runtime* `IllegalArgumentException` caso `n` seja menor do que 1.

Exemplo para uma sequência de inteiros e comparação pela ordem natural:

Sequência iterada por `seq` => 10, 20, 30, 12, 13, 8, 1, 2, 3

n	Lista produzida
1	[10, 20 30]
2	[12, 13]
3	[8]
4	[1, 2, 3]

c) `<K,V,C extends Collection<V>> void addAll(BufferedReader in, Map<K,C> m, Function<String, V> getValue, Function<V, K> getKey, Supplier<C> supC) throws IOException`

Que por cada linha lida do *stream in*, adicione ao contentor associativo `m` um determinado valor `v` na coleção associada a determinada chave `k`. No contentor associativo os dados associados à chave são coleções de valores do tipo `V`. O valor `v` é obtido por aplicação da função `getValue` à linha lida. A chave `k` é obtida por aplicação da função `getKey` ao valor `v`. O fornecedor `supC` permite obter novas instâncias de coleções `C` quando a chave ainda não existe no contentor associativo `m`.

d) `<K,V,C extends Collection<V>> void forEachIf(Map<K,C> m, Predicate<K> pred, Consumer<V> action)`

Que recebendo por parâmetro o contentor associativo `m`, executa a ação para cada objeto do tipo `V` existente em `m` caso a chave obedeça ao predicado `pred`.

e) `<K,V,C extends Collection<V>> void forEachIf(Map<K,C> m, BiPredicate<K,C> p, BiConsumer<K,C> action)`

Que recebendo por parâmetro o contentor associativo `m`, executa a ação para cada par chave, coleção que obedece ao predicado `p`.

2. Realize a classe **Families<C extends Collection<String>>** tendo em conta que:

- O campo **families** é um contentor associativo em que chave é o apelido da família e o valor associado é o conjunto de nomes cujo apelido é igual ao da chave (membros da família).
- O construtor recebe por parâmetro os fornecedores: **supMap** para obter o contentor associativo que armazena as famílias; e **supC** para obter novas instâncias de coleções **C** quando a chave ainda não existe no contentor associativo **families**.

- **String surname(String name)**

Método estático que recebendo por parâmetro um nome retorna o apelido.

- **Set<String> getSurnames()**

Retorna o conjunto de apelidos existentes em **families**.

- **C getNames(String surname)**

Retorna o conjunto de nomes que existem em **families** que têm como apelido **surname**.

- **void addNames(File names) throws IOException**

Lê o ficheiro **names**, onde cada linha contém nome completo de uma pessoa, e adicione os nomes ao contentor associativo. A chave é o apelido da família e o valor associado é o conjunto de nomes cujo apelido é igual ao da chave (membros da família). Na implementação usar o método **addAll** da alínea 1.c.

- **void addName(String name)**

Adiciona o nome ao contentor associativo. Na implementação usar o método **addAll** da alínea 1.c.

- **void forEachName(Consumer<String> action) throws IOException**

Que execute a ação sobre cada nome. Na implementação usar o método **forEachIf** da alínea 1.d.

- **void forEach(BiConsumer<String, c> action) throws IOException**

Que execute a ação sobre cada par (apelido, coleção de nomes).

- **void printFamilies(PrintWriter out, Set<String> except) throws IOException**

Escreve no *stream* as famílias cujo apelido não consta no conjunto **except**. Por cada família a primeira linha contém o apelido e o número de membros, as linhas seguintes contêm os nomes dos membros anteceditos do carácter **tab**. Na implementação usar o método **forEachIf** da alínea 1.e.

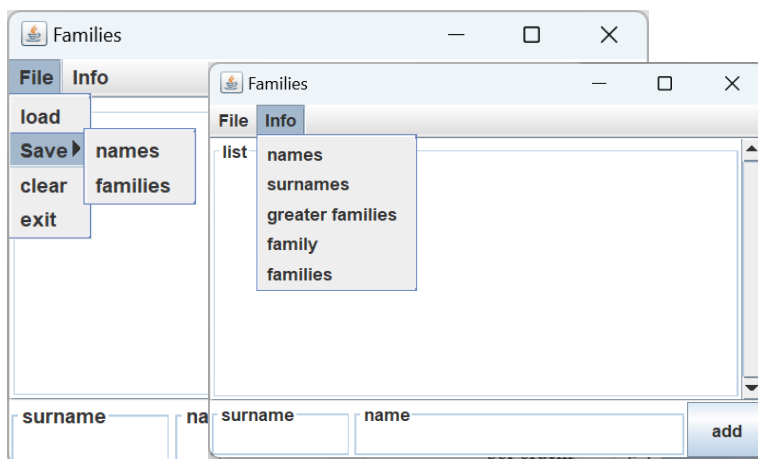
- **Set<String> getGreaterFamilies()**

Produz e retorna o conjunto dos apelidos das famílias mais numerosa. O conjunto de apelidos deve ficar ordenado por ordem decrescente. Na implementação usar o método estático **max** da classe **Collections** para obter o número de membros das famílias mais numerosa e usar o método **forEachIf** da alínea 1.e para produzir o conjunto dos apelidos.

3. Implemente uma interface gráfica para visualizar e introduzir informação sobre famílias. A aplicação deve permitir:

- Adicionar nomes através da interface ou da leitura de ficheiros.
- Guardar os nomes num ficheiro, por famílias ou unicamente os nomes.
- Listar os nomes de uma determinada família ou todos os nomes por família.
- Listar todos os apelidos.
- Obter os apelidos das famílias mais numerosas.
- Iniciar a estrutura de dados.

Families<C extends Collection<String>>
-families: Map<String, C>
+Families(supMap:Supplier<Map<String, C>>, supC:Supplier<C>)
+addName(name:String)
+addNames(names:File)
+getSurnames(): Set<String>
+getNames(surname:String): C
+getGreaterFamilies(): Set<String>
+printFamilies(out:PrintWriter,except:Set<String>)
+toString(): String
+forEachName(action:Consumer<String>)
+forEach(action:BiConsumer<String, C>)
+surname(name:String): String



Bom trabalho