

Grupo I

Considere os seguintes tipos da biblioteca do Java:

```
public interface Comparable<E> {  
    int compareTo(E e);  
}
```

```
public interface Comparator<E> {  
    int compare(E e1, E e2);  
}
```

```
public interface Predicate<E> {  
    //Retorna true se pt satisfaz o predicado  
    boolean test(E pt);  
}
```

```
public interface Supplier<S> {  
    // Retorna um S  
    S get();  
}
```

- a) [3] Realize o método com a seguinte assinatura:

```
public static <V, S extends Collection<V>>
```

```
    S elementsInRange(Iterable<V> seq, V min, V max, Comparator<V> cmp, Supplier<S> ss)
```

Que produz e retorna uma coleção com os elementos da sequência `seq` que são maiores ou iguais a `min` e menores ou iguais que `max`. A relação de ordem é definida pelo comparador `cmp`. A coleção a produzir é obtida com o `Supplier` `ss`.

- b) [3] Utilizando o método da alínea anterior e implementando o `Comparator` e o `Supplier` realize o método com a seguinte assinatura:

```
public static Set<String> wordsStartingWith(List<String> words, char letter)
```

que retorna o conjunto de *strings* ordenados lexicograficamente sem distinguir maiúsculas de minúsculas com as *strings* contidas na lista `words` que começam com a letra `letter`.

Nota: O método `compareToIgnoreCase` de `String` compara sem distinguir maiúsculas de minúsculas.

- c) [3] Realize o método com a seguinte assinatura:

```
public static <E> List<E> isSublist( ArrayList<E> l1, ArrayList<E> l2,  
                                   Comparator<E> cmp )
```

Que recebendo por parâmetro duas listas ordenadas segundo o comparador `cmp`, retorna `true` se a segunda lista for uma sublista da primeira.

Nota: No algoritmo tenha em conta que as sequências estão ordenadas.

Exemplo:

Seja: `l1 = List.of(1, 5, 7, 8)` e `l2=List.of(7, 8)` -> `l2` é sublista de `l1`

`l1 = List.of(1, 5, 7, 8)` e `l2=List.of(5, 8)` -> `l2` **não** é sublista de `l1`

Grupo II

```
public interface BiConsumer<S,I>{
    void accept(S nm, I nt);
}
```

- a) [3] Realize o método estático público com a seguinte assinatura:

```
void forEachStudent(BufferedReader in, BiConsumer<String, Integer> process)
    throws IOException
```

Cada linha do *stream* in contém o nome e a nota de um aluno, por esta ordem, separados por ‘:’. Por cada linha lida do *stream* deve ser passado ao **BiConsumer process** o nome e a nota.

Nota: O método estático **Integer.parseInt(String str)** retorna o valor inteiro que a *string* recebida por parâmetro representa.

```
Ana Vieira: 20
Carla Vaz: 18
David Pais: 20
Maria Silva: 18
Pedro Morais: 13
...
```

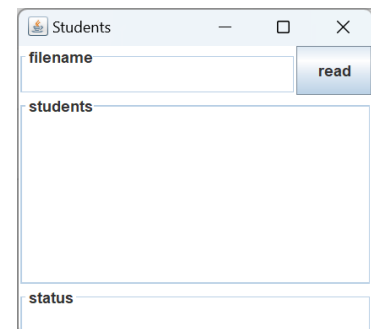
- b) [3] Utilizando o método da alínea anterior e implementando o **BiConsumer<String,Integer>** realize o método público estático com a seguinte assinatura:

```
void updateNotes( Map<String, Integer> students,
    String filename ) throws IOException
```

Que atualiza o contentor associativo **students** com a informação contida no ficheiro de texto com nome **filename** em que cada linha que contém o nome de um aluno seguido da nota que obteve. Na implementação do método **accept** do **BiConsumer** por cada par (nome/nota) caso o nome não exista como chave de **students** deve ser acrescentada uma nova entrada, caso contrário deve ser atualizado o valor associado se e só se for inferior ao da nota recebida por parâmetro.

- c) [5] Faça uma aplicação com o aspeto da figura.

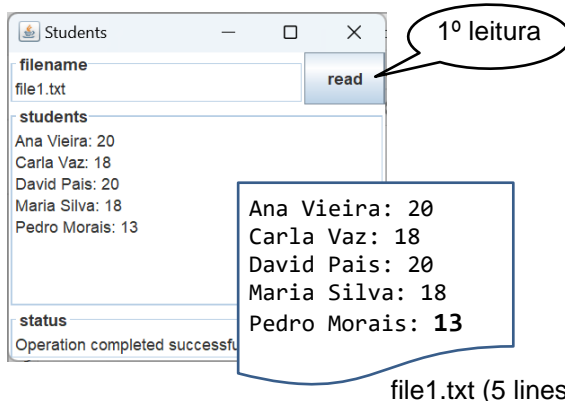
- Quando for premido o botão “**read**” deve ser chamado o método **updateNotes** para atualizar uma estrutura de dados **students** do tipo **Map<String, Integer>** com os dados lidos do ficheiro cujo nome está presente na caixa de texto “**filename**”. A área de texto “**students**” deve refletir a atualização.
- Caso a operação ocorra com sucesso deve ser escrito na caixa de texto “**status**” a informação “*Operation completed successfully*”, caso exista erro no acesso aos ficheiros deve ser escrito na caixa de texto “**Erro:** ” seguido da mensagem de erro.



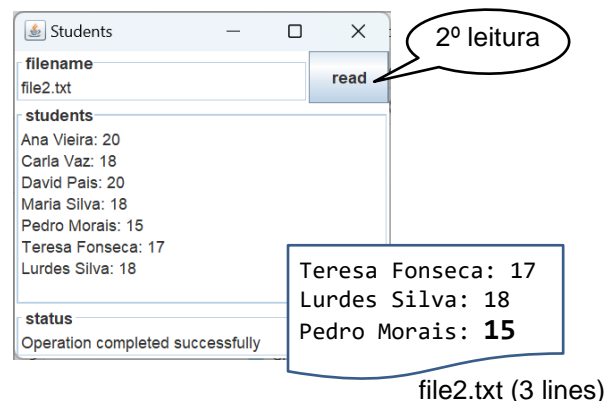
Para a instânciação das caixas e areas de texto use os seguintes métodos:

```
JTextField newJTextField( String title ) {
    JTextField tf = new JTextField( 18 ); tf.setBorder( new TitledBorder( title ) );
    return tf;
}
```

```
JTextArea newJTextArea( String title ) {
    JTextArea ta = new JTextArea(8, 30); ta.setBorder( new TitledBorder( title ) );
    return ta;
}
```



file1.txt (5 lines)



file2.txt (3 lines)