Linguagem Java

StringTokenizer, useDelimiter e split

Gestão de arrays

ArrayList

Classes genéricas

Exercício

- Resolução do exercício 11 da ficha

11. Defina uma classe para representar a informação acerca de um relatório. Um relatório (instância da classe `Report`) é definido através dos seguintes campos:

- *Titulo* (`String`)
- *Conjunto de autores* (1ª versão: array de *strings*; 2ª versão: `ArrayList` de *strings*)
- *Texto* (`StringBuilder` ou `StringBuffer`)

A classe deve disponibilizar as seguintes funcionalidades:

- Acrescentar um autor ao relatório garantindo que não existem repetidos.
- Remover um autor, garantido a manutenção da ordem dos restantes.
- Acrescentar texto. O texto será concatenado ao final do texto já existente.
- Substituir por letras maiúsculas as primeiras letras das palavras depois de pontos finais.
- Contar as palavras do texto (as palavras podem estar separadas por mais do que um separador: espaços, tabs, mudanças de linha, vírgulas e pontos). A função deverá retornar o número de palavras.
- Contar as ocorrências de uma dada palavra. A função deverá retornar o número de ocorrências ou 0 (zero) caso a palavra não exista no texto.
- Gerar `String` com toda a informação do relatório (método `toString()`).

Implemente uma aplicação Java que permita testar todas as funcionalidades da classe `Report`.

Divisão de uma *string* em *tokens*

- Para dividir o texto de um objeto `String` podem-se usar várias técnicas
  - Análise caracter a caracter
    - `String.charAt`, `String.subString`, …
  - Usar um objeto `Scanner` para iterar sobre a *string*

```
Scanner sc = new Scanner(strOriginal);
sc.useDelimeter(<regex com separadores pretendidos>);
while (sc.hasNext()) { String str = sc.next();...}
```

  - Usar o método `split` da classe `String`

```
String [] tokens = strOriginal.split(<regex_separadores>);
```

  - Usar um objeto `StringTokenizer`

```
StringTokenizer st = new
StringTokenizer(strOriginal,<str_delimitadores>);
```

Regular expressions

- Sequência de caracteres que permitem descrever padrões em textos
  - https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html
  - A classe `Pattern` permite criar e compilar expressões regulares
  - A classe `Matcher` permite verificar se um determinado padrão é respeitado
  - Existem outras classes e métodos que usam indiretamente estas classes, como é o caso do método `split` e funcionalidades obtidas pelo `useDelimiter` do `Scanner`
- Exemplo para representar separadores de palavras (de acordo com o pretendido para o exercício)
  - `"[\\s,.]+"`
    - o \s representa os separadores por omissão: espaço, tab, mudança de linha e mudança de página
    - o '+' permite indicar que se pretende uma ou mais ocorrências (por exemplo, podem existir vários espaços)

- Exemplo de um expressão regular para verificar um endereço IP:

```
private static final String IPV4_PATTERN =
   "^([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\\." +
   "([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\\." +
   "([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\\." +
   "([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])$";
```

Alteração da dimensão de um *array*

- Após a criação de um *array* não se pode alterar o seu tamanho
- Caso seja necessário um *array* com mais elementos então devemos criar um novo *array* com a dimensão pretendida e copiar os elementos do *array* original
- Exemplo:

```
int [] array = {1,2,3};
System.out.println(Arrays.toString(array));
   //output: [1, 2, 3]
int [] newArray = new int[array.length+2];
System.arraycopy(array,0,newArray,0,array.length);
array = newArray;
System.out.println(Arrays.toString(array));
   //output: [1, 2, 3, 0, 0]
array = Arrays.copyOf(array,array.length+2);
System.out.println(Arrays.toString(array));
   //output: [1, 2, 3, 0, 0, 0, 0]
```

ArrayList

- A linguagem Java disponibiliza um conjunto de classes que permitem facilitar o processamento de coleções de dados, usando diferentes paradigmas (listas, conjuntos, pares atributo-valor, ...)

- A classe `ArrayList` permite gerir uma lista dinâmica de objetos, através da disponibilização de um conjunto de funções adequadas, por exemplo:
  - `get, size`
  - `add, addAll, remove, removeAll, removeIf, clear`
  - `indexOf, contains, lastIndexOf`
  - `iterator, forEach`
  - `sort`
  - `toArray, toString`
  - `...`

ArrayList

- A criação de uma instância de `ArrayList` deve ser realizada especificando o tipo mais genérico de objetos que irá conter
  - A classe `ArrayList` está definida recorrendo ao conceito de "*classes genéricas*"

```
ArrayList<String> list1 = new ArrayList<>();
list1.add("DEIS-ISEC");

ArrayList<Object> list2 = new ArrayList<>();
list2.add("Prog. Avançada");
list2.add(1234);

for( int i = 0 ; i < list2.size(); i++){
      // …  list2.get(i)
}

for(String s: list1){
      // …  s;
}
```

Classes genéricas - exemplo

```
package pt.isec.pa.genericos;

class MeuTipo<T> {
    T valor;

    public MeuTipo(T valor) {
        this.valor = valor;
    }

    @Override
    public String toString() {
        return "MeuTipo{valor=" + valor + '}';
    }
}

public class App {
    public static void main(String[] args) {
        MeuTipo<Integer> vi = new MeuTipo<>(45);
        MeuTipo<Double> vd = new MeuTipo<>(4.5);
        MeuTipo<String> vs = new MeuTipo<>("quarenta e cinco");
        for(var mt : Arrays.asList(vi,vd,vs))
            System.out.println("> "+mt);

        System.out.println(vd.getClass() == vs.getClass());
        MeuTipo m1 = vd;
        MeuTipo m2 = vs;
        System.out.println(m1.valor.getClass() == m2.valor.getClass());
        System.out.println(m1.getClass().toString());
        System.out.println(m2.getClass().toString());
        System.out.println(m1.valor.getClass().toString());
        System.out.println(m2.valor.getClass().toString());
    }
}
```

Output:

```
> MeuTipo{valor=45}
> MeuTipo{valor=4.5}
> MeuTipo{valor=quarenta e cinco}
true
false
class pt.isec.pa.genericos.MeuTipo
class pt.isec.pa.genericos.MeuTipo
class java.lang.Double
class java.lang.String
```