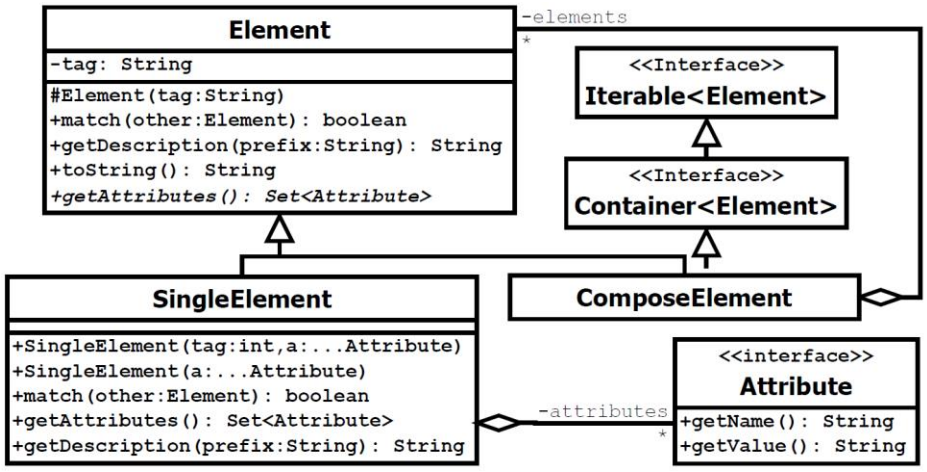
Instituto Superior de Engenharia de Lisboa
Programação III
1º Teste – Exame de época recurso - 30 de janeiro de 2023 (Duração: 1:00)

ISEL ADEETC

Pretende-se implementar uma solução para a representação de elementos que descrevem o formato de documentos, sendo este formato uma versão simplificada do formato XML (*XML Schema Definition*). Todos os elementos contidos num documento contêm uma marca (*tag*). Existem dois tipos de elementos; os simples e os compostos. Os elementos simples podem conter vários atributos e cada atributo é composto por um nome e por um valor. Os elementos compostos podem conter vários elementos. Para o efeito chegou-se ao seguinte diagrama estático de classes.



Tendo em conta o diagrama estático de classes e o output dos troços de código:

1. [2] Defina a classe **`A`**.

```
Attribute a1 = new A("name","Maria");
System.out.println( a1 );
Attribute a2 =  new A("name","Maria");

System.out.println( a1.equals(a2) );
System.out.println( a1.hashCode() == a2.hashCode() );
```

```
name="Maria"
true
true
```

2. [3] Defina a classe abstrata **`Element`**. No construtor é passada a marca (*tag*). Dois elementos correspondem (*match*) se tiverem a mesma *tag*. O método **`getAttributes`** é abstrato. O método **`getDescription`** retorna o prefixo recebido por parâmetro concatenado com o carácter '<' e com a *tag*. O método **`toString`** não pode ser redefinido, retorna o resultado da chamada ao método **`getDescription`** passando-lhe como parâmetro a *string* vazia.

3. [4] Defina a classe **`SingleElement`**. Tenha em conta que:

- No construtor com dois parâmetros é passado a *tag* e os atributos. No construtor com um parâmetro só são passados os atributos, a *tag* é "*element*".
- O método **`match`** retorna **`true`** se o elemento **`el`** tiver a mesma *tag* e os mesmos atributos.
- O método **`getAttributes`** retorna o conjunto imutável de atributos.
  Nota: O método estático **`of`** da interface **`Set`** recebendo por parâmetro um *array* de **`Attribute`** retorna um **`Set<Attribute>`** imutável.
- O método **`getDescription`** constrói uma *string* com: o resultado da chamada ao método **`getDescription`** da classe base; a lista de atributos; e a *string* "/>" (ver output do exemplo).

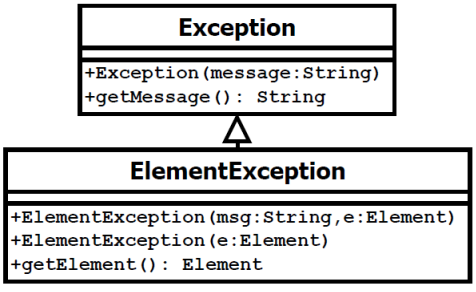
```
Element e1 = new SingleElement("fa", new A("min","10"), new A("max","20"));
System.out.println( e1 );

Element e2 = new SingleElement(new A("file","x.txt"));
System.out.println( e2 );
```

```
<fa min="10" max="20"/>
<element file="x.txt"/>
```

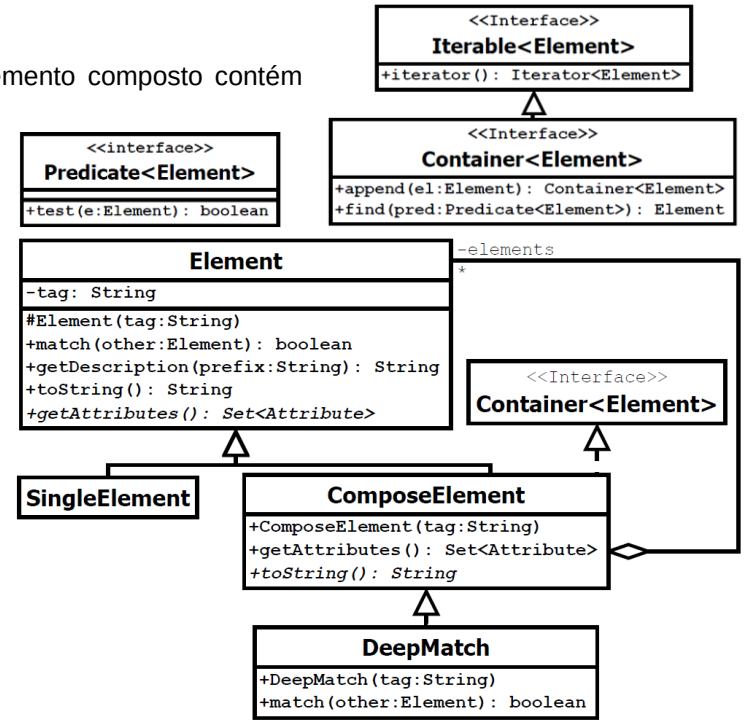
4. [2] Defina a classe **`ElementException`** para que o método **`getMessage`** herdado de **`Exception`** retorne:

- A *string* passada por parâmetro no construtor concatenada com a descrição do elemento, caso tenha sido instanciado com o construtor com dois parâmetros;
- A *string* "*Invalid element*: " seguida da descrição do elemento caso tenha sido instanciada com o construtor com um parâmetro.



5. [2] Defina a interface **`Container<Element>`**. O método **`append`** pode lançar a exceção **`ElementException`**.

6. [5] Defina a classe **`ComposeElement`**. Um elemento composto contém uma lista de elementos (simples ou compostos).



- O método **`find`** procura em **`elements`** um elemento que obedeça ao predicado.

  Retorna o primeiro que encontra ou **`null`** caso não encontre;
- O método **`append`** adiciona o elemento **`el`** caso não encontre um elemento em **`elements`** que faça correspondência (*match)*. Caso contrário, lança a exceção **`ElementException`** passando-lhe por parâmetro na instanciação a mensagem "*Already exists:* " e o elemento que encontrou. O método **`append`** retorna o próprio elemento composto.

  NOTA: Usar o método **`find`** para encontrar em **`elements`** um elemento que faça match;
- O método **`iterator`** retorna o **`Iterator`** para os elementos de **`elements`**;
- O método **`getAttributes`** retorna um conjunto vazio.

  NOTA: A classe **`Collections`** disponibiliza o método estático **`emptySet`** que retorna um conjunto vazio imutável;
- O método **`toString`** lista os elementos (ver exemplo de output).

```
try{
 Element e1= new SingleElement("age", new A("min","10"), new A("max","20"));
 Element e2= new SingleElement("id", new A("name","Maria"));
 ComposeElement ce1 = new ComposeElement("data");
 ce1.append(e1).append(e2);
 System.out.println( ce1 );
```

```
<data>
    <age min="10" max="20"/>
    <id name="maria"/>
<data/>
```

```
 Element e3= new SingleElement("ad", new A("street","Rua da Prata"));
 ComposeElement ce2=new ComposeElement("contact");
 ce2.append(e3).append(ce1);
 System.out.println( ce2 );

 ce2.append( e3 ); // Vai lançar exceção
} catch(ElementException ex) {
 System.out.println(ex.getMessage());
}
```

```
<contact>
  <ad street="Rua da Prata"/>
  <data>
    <age min="10" max="20"/>
    <id name="maria"/>
  <data/>
<contact/>
```

```
Already exist: <ad street="Rua da Prata"/>
```

7. [2] Defina a classe **`DeepMatch`**. Um **`DeepMatch`** é um **`ComposeElement`** em que o método **`match`** retorna **`true`** só e só se o elemento **`other`** for composto, tiver a mesma *tag* e para cada elemento de **`elements`** existir em **`other`** um elemento que correspondente. Usar o método **`find`** para encontrar em **`other`** um elemento que faça *match*.