Biblioteca de Manipulação de XML

Projeto de Programação Avançada 2023/2024

Pretende-se uma biblioteca para geração e manipulação de XML em Kotlin, cobrindo os elementos essenciais:

- Documento
- Entidades (*tags*)
- Texto aninhado em entidade
- Entidades aninhadas
- Atributos

O seguinte exemplo ilustra os elementos acima.

```
<?xml version="1.0" encoding="UTF-8"?>
<plano>
	<curso>Mestrado em Engenharia Informática</curso>
	<fuc codigo="M4310">
		<nome>Programação Avançada</nome>
		<ects>6.0</ects>
		<avaliacao>
			<componente nome="Quizzes" peso="20%"/>
			<componente nome="Projeto" peso="80%"/>
		</avaliacao>
	</fuc>
	<fuc codigo="03782">
		<nome>Dissertação</nome>
		<ects>42.0</ects>
		<avaliacao>
			<componente nome="Dissertação" peso="60%"/>
			<componente nome="Apresentação" peso="20%"/>
			<componente nome="Discussão" peso="20%"/>
		</avaliacao>
	</fuc>
</plano>
```

O objetivo é assumir o papel de quem está a desenvolver algo para terceiros utilizarem (biblioteca). Assim sendo, aspetos de API (forma e documentação) deverão ser tidos em especial atenção. Por outro lado, a biblioteca deverá oferecer alguma flexibilidade na sua utilização para permitir aplicá-la num leque alargado de situações.

Atenção: A biblioteca não tratará de *parsing* de XML (embora isso fizesse sentido numa situação real). Apenas é exigida a geração de XML a partir de modelos em memória.

Documentação: A API da biblioteca deverá estar documentada utilizando KDoc.

Testes: Todas as funcionalidades deverão ter os respectivos testes JUnit.

Fase 1 - Modelo

Objetivo: Desenvolver as classes para representar XML em memória (modelo), operações de manipulação, e testes.

API

As classes deverão permitir:

1. **adicionar e remover** entidades
2. **adicionar, remover, e alterar** atributos em entidades
3. aceder à **entidade mãe** e **entidades filhas** de uma entidade
4. ***pretty print*** em formato de String, e **escrita para ficheiro**
5. varrimento do documento com objetos **visitantes** (*Visitor*)
6. **adicionar atributos** globalmente ao documento
   (fornecendo *nome* da entidade, *nome* e *valor* do atributo)
7. **renomeação de entidades** globalmente ao documento
   (fornecendo *nome antigo* e *nome novo*)
8. **renomeação de atributos** globalmente ao documento
   (fornecendo *nome* da entidade, e *nome antigo* e *nome novo* do atributo)
9. **remoção de entidades** globalmente ao documento
   (fornecendo *nome*)
10. **remoçao de atributos** globalmente ao documento
    (fornecendo *nome* da entidade, e *nome* de atributo)

Micro-XPath

Com vista a ter uma forma de obter fragmentos de XML, pretende-se permitir interrogar o documento com expressões XPath *simples*. Entenda-se, expressões apenas compostas por uma sequência de nomes de entidades. Por exemplo a expressão seguinte resultaria na seguinte lista de entidades:

fuc/avaliacao/componente

```
<componente nome="Quizzes" peso="20%"/>
<componente nome="Projeto" peso="80%"/>
<componente nome="Dissertação" peso="60%"/>
<componente nome="Apresentação" peso="20%"/>
<componente nome="Discussão" peso="20%"/>
```

Fase 2 - Mapeamento Classes-XML

Objetivo: Desenvolver uma forma de obtenção automática de entidades XML a partir de objetos, com base na estrutura das respectivas classes.
Atenção: não se pretende a geração direta de texto XML a partir dos objetos, mas sim a instanciação das classes do modelo. O facto de trabalhar com o modelo permite uma fácil manipulação posterior à obtenção automática, quando comparado com texto.

O modelo desenvolvido na Fase 1 permite a manipulação de documentos XML em memória, contudo só por via de instanciação manual. Nesta fase a ideia é que as classes do modelo possam ser instanciadas automaticamente a partir de objetos, oferecendo alguma flexibilidade de personalização por via de anotações nas classes, permitindo:

- alterar nomes de identificadores na tradução para XML
- determinar de que forma são traduzidos os atributos os objetos (atributo XML ou entidade)
- excluir atributos de objetos

Os exemplos seguintes demonstram a flexibilidade desejada na obtenção de XML (omitindo as anotações a desenvolver no projeto).

Classe	`class ComponenteAvaliacao(val nome: String, val peso: Int)`
Objeto	`val c = ComponenteAvaliacao("Quizzes", "20")`
XML	`<componente nome="Quizzes" peso="20"/>`

Classe	`class FUC(`
	`  val codigo: String,`
	`  val nome: String,`
	`  val ects: Double,`
	`  val observacoes: String,`
	`  val avaliacao: List<ComponenteAvaliacao>`
	`)`

Objeto	`val f = FUC("M4310", "Programação Avançada", 6.0, "la la...",`
	`  listOf(`
	`  ComponenteAvaliacao("Quizzes", "20"),`
	`  ComponenteAvaliacao("Projeto", "80")`
	`  )`
	`)`

XML	`<fuc codigo="M4310">`
	`  <nome>Programação Avançada</nome>`
	`  <ects>6.0</ects>`
	`  <avaliacao>`
	`  <componente nome="Quizzes" peso="20"/>`
	`  <componente nome="Projeto" peso="80"/>`
	`  </avaliacao>`
	`</fuc>`

Embora os exemplos anteriores ilustrem algumas possibilidades de personalização da tradução de objetos em XML, nunca será possível antever todas as formas possíveis de tradução. Por exemplo, podia ser um objetivo traduzir os objetos FUC da seguinte forma.

```
<fuc codigo="M4310">
      <ects>6.0</ects>
      <nome>Programação Avançada</nome>
      <componente nome="Quizzes" peso="20%"/>
      <componente nome="Projeto" peso="80%"/>
</fuc>
```

Note que a ordem de atributos não corresponde à da classe, a lista de componentes de avaliação deixou de ter uma entidade, e que o valor textual do peso não simplesmente o inteiro em String.

Por forma a ter uma forma de personalização do texto que é inserido no XML resultante de valores de atributos dos objetos, pretende-se uma anotação para indicar uma classe que implementa a transformação a fazer à string por omissão (p.e. acrescentar “%”).

```
class ComponenteAvaliacao(
    val nome: String,
    @XmlString(AddPercentage::class)
    val peso: Int
)
```

Por forma a ter uma forma de personalização pós-mapeamento, pretende-se uma anotação que associe um adaptador que faz alterações livres na entidade XML após mapeamento automático (p.e., alterar a ordem dos atributos XML).

```
@XmlAdapter(FUCAdapter::class)
class FUC(...) {
   ...
}
```

Fase 3 - DSL Interna (opcional)

Objetivo: Disponibilizar uma API que facilite a instanciação de modelos XML por via de uma DSL interna em Kotlin (estilo livre).

Fase 4 - Publicação

Objetivo: Disponibilizar a biblioteca publicamente com documentação e tutoriais.

Deverá ser criado um repositório no github (público na entrega), contendo o código do projeto, casos de teste, e tutoriais de utilização da biblioteca em ficheiros Markdown (.md).