Trabalho 3 de Tópicos Avançados em Linguagens de Programação
Prof. José de Oliveira Guimarães
Veja a data de entrega no AVA

Este trabalho possui partes A, B e C. A parte seguinte envolve toda a parte anterior. Entregue apenas a parte mais abrangente que você conseguir fazer. A parte C vale 10, a B vale 8,5 e a A vale 7,5.

Parte A

Faça um protótipo `XML_Action` que possui um método

```
func toXML: Any any -> String
```

que produz código em XML, uma string, representando `any`. Em execução, o tipo do objeto `any` não pode ser um tipo básico (`Nil`, `Int`, ...). Esta string retornada deve ser o código XML com um elemento XML representando `any` cujo nome é dado por uma *feature* chamada `xmlRoot` associada ao protótipo de `any` com um atributo `prototype` com o nome do protótipo de `any`. O conteúdo deste elemento são elementos associados a variáveis de instância anotados com a *feature* `xmlElement`. O valor da variável de instância com nome `varName` deve ser obtido chamando-se um método `getVarName` (assuma que este método exista). Para uma variável de instância de um tipo básico anotada como

```
@feature(xmlElement, "personAge")
var Int age
```

deve ser gerado um elemento

```
<personAge>
24
</personAge>
```

se o valor da variável em execução for `"Isaac Newton"`. Assuma que nenhum tipo é `Nil` ou alguma união.

Mostraremos um exemplo do que deve ser gerado usando os protótipos abaixo.

```
@feature(xmlRoot, "CityOfResidence")
object City
    func init: String cityName { self.cityName = cityName }
    func getCityName -> String = cityName;
    @feature(xmlElement, "nameOfCity")
    String cityName;
end

@feature(xmlRoot, "A_person")
object Person
    @init(name, age, city)
    func getName -> String = name;
    func getAge -> Int = age;
    func getCity -> City = city;
    @feature(xmlElement, "PersonName")
    String name
```

```
    @feature(xmlElement, "PersonAge")
    Int age
    @feature(xmlElement, "PersonCity")
    City city
end
```

O envio de mensagem

```
    XML_Action toXML: Person("Isaac Newton", 24, City("Cambridge"))
```

deve retornar

```
<A_person prototype="Person">

    <PersonName>
    "Isaac Newton"
    </PersonName>

    <PersonAge>
    24
    </PersonAge>

    <CityOfResidence prototype = "City">
        <nameOfCity>
        "Cambridge"
        </nameOfCity>
    </CityOfResidence>

</A_person>
```

Nesta parte A não se preocupe com uniões e referências circulares. Assuma que todos os métodos `get` existam, que não há variáveis do tipo `Nil` e que toda variável anotada com `xmlElement` cujo tipo em execução não é um tipo básico pertence a um protótipo anotado com `xmlRoot`. No exemplo, o objeto de `Person` possui variável `city` anotada com `xmlElement`. Então em execução necessariamente (assuma isto) o protótipo do objeto referenciado por `city` em execução, `City` no exemplo, é anotado com `xmlRoot`.

O método `toXML:` deveria sinalizar um erro se `City` não fosse anotada com `xmlRoot` pois este método não saberia como gerar o código XML do objeto `City` referenciado por `city`. Então assuma que este erro não acontecerá.

Parte B

Confira agora que o objeto passado como parâmetro a `toXML:` pertence a um protótipo anotado com `xmlRoot`. Considere que uniões não etiquetadas possam ser o tipo de variáveis de instância. Uniões como

```
    Union<wattHour, Double, calorie, Double, joule, Double>
```

continuam proibidas (não é necessário fazer esta conferência). O valor de alguma variável pode ser `Nil` — neste caso o tipo da variável é `Nil` ou uma união com um dos tipos `Nil`.

Nesta parte B, assuma que não há referências circulares e que métodos `get` existam para todas as variáveis.

Parte C

Lance uma exceção `ExceptionCircularReference` se há uma referência circular entre os objetos e uma exceção `ExceptionGetMethodDoesNotExist` se algum método `get` não existe. No primeiro caso, a exceção deve ter um parâmetro do tipo `String` que informa o protótipo do objeto que causou a circularidade. A exceção `ExceptionGetMethodDoesNotExist` deve ter como parâmetro uma `String` com o nome da variável para a qual não foi encontrado o método `get`. Se `toXML:` receber um objeto de tipo básico como parâmetro ou um objeto cujo protótipo não seja anotado com `xmlRoot` , lance a exceção `ExceptionToXMLIllegalParameter` que deve tomar como parâmetro o nome do protótipo do parâmetro.

O método que chama `toXML:` deve tratar adequadamente estas exceções e emitir uma mensagem de erro adequada.