INE5416 - Paradigmas da Programação (2015/2)

Relatório 6: Classes e Tipos

Caique Rodrigues Marques 13204303

Embora Haskell também possa ser orientado a objetos, o conceito de classes em Haskell tem a função de parametrizar o polimorfismo.

Polimorfismo

Nas linguagens de programação, o polimorfismo é definição de uma interface para diferentes arquivos, ou seja, uma interface pode ter várias formas. Um exemplo, dada uma interface genérica e dois arquivos distintos que implementam esta interface, portanto, de um olhar diferente, a interface possui duas formas de operar. Há dois tipos de polimorfismo:

- **Polimorfismo Paramétrico:** Quando uma função não possui um tipo especificado, então pode ser usado transparentemente por qualquer tipo, ou seja, a função pode retornar sempre um mesmo valor independente do tipo que foi aplicada a ela. Em orientação a objetos isto é conhecido como programação genérica, como, por exemplo, o uso de templates.
- **Polimorfismo Ad Hoc:** Ao contrário do paramétrico, o polimorfismo ad hoc se trata de funções que podem se comportar e retornar valores diferentes dependendo do tipo em que foi usado, por exemplo, especificando o uso em um inteiro pode retornar diferente se usando em um char. Ad-hoc é popular em orientação a objetos e também é conhecido como sobrecarregamento de funções (functions overloading) onde as funções têm a mesma assinatura, mas comportamentos diferentes.

A linguagem Haskell, cujo paradigma é o funcional, implementa os dois tipos de polimorfismo. Em C++ e Java, cujos paradigmas são a orientação a objetos e como já dito antes, o polimorfismo ad hoc pode ser implementado para complementar o conceito de herança. A seguir um exemplo de function overloading em C++ (semelhante pode ser aplicado em Java).

```
#include <string>

int add(int a, int b)
{
    return a + b;
}

std::string add(const char *a, const char *b)
{
    std::string result(a);
    result += b;
    return result;
}
```

Note o método add(), ele foi aplicado a dois tipos: inteiros e strings. Ambas as funções têm comportamentos semelhantes, mas em inteiros o valor retornado é a soma de dois inteiros, enquanto em strings, o valor retornado é a concatenação de duas strings.

Em Haskell, o funcionamento de polimorfismo ad hoc é um pouco mais além, ele não se limita apenas a funções, mas também a tipos, inclusive, Haskell não suporta a possibilidade de duas funções com a mesma assinatura como em C++ ou Java, portanto, ele pode mudar o tipo de retorno em uma função. A seguir um exemplo:

```
f a1 a2 = case (typeOf a1) of
        Int  -> a1 + a2
        Bool -> a1 && a2
        _    -> a1
```

C++ e Java têm implementações semelhantes de polimorfismo paramétrico, como exemplificado antes, com o uso de templates, o método sempre retorna a mesma resposta independente do tipo usado. A seguir, um exemplo de uma implementação de um método de uma lista, em C++ (semelhante pode ser aplicado em Java). Como se pode ver, independente do tipo que for especificado em "T" (seja inteiros, char, strings, booleanos), o método sempre irá adicionar um dado ao final da lista de elementos "T".

```
template <typename T>
void Lista<T>::adiciona(T obj)
{
    this->topo += 1;
    if (this->topo >= this->tamanhoMaximo) {
        throw "Tamanho maximo excedido";
    }
    this->arranjo[this->topo] = obj;
}
```

Em Haskell, a implementação de polimorfismo paramétrico é bem semelhante às implementações em orientação a objetos. A seguir um exemplo simples, o parâmetro "[a]" pode ser uma lista de qualquer tipo e a função sempre vai retornar o tamanho, que é definido em inteiros.

```
length :: [a] -> Int
```

Classes

O conceito de classes em Haskell é diferente do conceito de classes em C++, neste, é definido para implementar métodos e interfaces, no primeiro, é definido para implementar interfaces genéricas. Segue um exemplo de uma classe em Haskell onde, dado um número "a", ele o retorna na sua forma real.

```
class (Ord a, Num a) => Real a where
    toRational :: a -> Rational
```

Haskell também possui um outro conceito de classes, as Type Classes, é uma interface, mas com algum comportamento mais específico. Há confusões com implementação de type classes e classes em orientação a objetos, mas são bem distintos como se dá para notar. Como Haskell é uma linguagem fortemente tipada, ele usa variáveis de tipagem para definir as classes que são construídas, como são bastantes usadas as variáveis Eq, Ord, etc..