Category Theory Arrows e Monads em Java

Mozart L. Siqueira
Ciências da Computação
Centro Universitário La Salle - Unilasalle
Email: mozarts@unilasalle.edu.br

Pablo M. Parada
Ciências da Computação
Centro Universitário La Salle - Unilasalle
Email: pablo.paradabol@gmail.com

***Resumo*—<escrever>**
***Index Terms*—<escrever>**

I. INTRODUÇÃO

<escrever>

II. SOBRE O PARADIGMA FUNCIONAL

Influenciado principalmente pelo desenvolvimento do *lambuda calculus* [1], compondo o grupo da programação declarativa, o paradigma funcional utiliza-se da idéia de expressar computações através de funções combinadas em expressões. Neste, funções expressam o que deverá ser computado, ao invés de como sera computado [2]. Programas são construídos através da composição, tal que funções triviais (ou *building blocks*) são combinadas dando origem a novas funções que descrevem computações mais complexas.

Building blocks não devem fazer uso de variáveis que dependam de estado, isso significa que a computação deve ser pura e sem efeitos indesejados (ou *side-effects*). Também destaca-se o princípio de imutabilidade, onde o valor é de uma variável é determinado em sua criação, não permitindo novas atribuições posteriormente.

É possível afirmar que ao expressar um programa em uma linguagem funcional, obtem-se uma maneira concisa de solucionar problemas, dado que este constitui-se de operações e objetos atômicos e regras gerais para sua composição [3]. Estas qualidades são apreciadas nos tempos atuais, onde há necessidade de tratar os problemas oriundos do não-determinismo. Assim, o paradigma funcional mostra-se capaz, inclusive de influenciar outras linguagens como *Java* [4].

A. Lambda Expressions e Anonymous Inner Classes

Ao fornecer funções de primeira classe (também lambda expressions ou closures), a linguagem Java habilita a substituição de annonymous inner classes (AIC) por lambda expressions. Contudo, apesar destas serem transparentes a nível de código, ambas funcionalidades possuem diferentes implementações sob a Máquina Virtual Java (JVM).

AIC são compiladas pela JVM, dando origem a um novo arquivo contendo sua declaração. Além do mais, ao utilizarmos a palavra reservada *this* estamos referenciando a própria instância anônima. Como representam instâncias de uma classe, estas devem ser carregadas pelo *class loader* e seus construtores invocados pela máquina virtual. Ambas etapas consomem memória, tanto *heap* para alocação de objetos, quanto *permgem*, utilizada para guardar metadados, definições de classes e métodos [5].

Após a declaração do array de inteiros,

```
Integer[] integers = new Integer[]{1, 2, 3, 4, 5};
```

Listing 1. Array de Inteiros

é possível ordena-lo utilizando o método *sort* da classe *Arrays*.

```
Arrays.sort(integers, new Comparator<Integer>() {
    public int compare(Integer a, Integer b) {
        return a.compareTo(b);
    }
});
```

Listing 2. Sort - Anonymous Inner Class

Este recebe como primeiro argumento o array que deseja-se ordernar e como segundo, uma AIC da interface *Comparator* implementando o método *compare*.

Diferentemente de AIC, lambdas postergam a estratégia de compilação para em tempo de execução, utilizando a instrução *invokedynamic* [6]. Funções são traduzidas para métodos estáticos vinculados ao arquivo da classe correspondente a sua declaração, eliminando o consumo de memória. Agora, ao referir-se a *this*, a classe que delimita a lamda expression é acessada, ao contrário de AIC que acessa sua própria instância. Por fim, closures fornecem formas mais expressivas de representar comportamentos.

Assim, o mesmo código listado em 2 pode ser transformado em

```
Arrays.sort(integers, (a, b) -> a.compareTo(b));
```

Listing 3. Lambda Expressions

uma lambda que descreve a mesma expressão sem os encargos impostos por AIC.

REFERÊNCIAS

[1] P. Hudak, "Conception, evolution, and application of functional programming languages," *ACM Computing Surveys (CSUR)*, vol. 21, no. 3, pp. 359–411, 1989.

[2] K. Louden *et al.*, *Programming Languages: Principles and Practices*. Cengage Learning, 2011.

[3] G. Michaelson, *An Introduction to Functional Programming Through Lambda Calculus*. Courier Corporation, 2011.

[4] B. Goetz, R. Forax, D. Lea and B. Lee, "State of the lambda," Sept 2013, White Paper. [Online]. Available: http://cr.openjdk.java.net/ briangoetz/lambda/lambda-state-final.html

[5] C. Hunt and B. John, *Java Performance*. Prentice Hall Press, 2011.

[6] B. Goetz, "Translation of lambda expressions," Apri 2012, White Paper. [Online]. Available: http://cr.openjdk.java.net/ briangoetz/lambda/lambda-translation.html