PADRÕES DE PROJETO COMPORTAMENTAIS

Leandro Duarte Novaes

leandroduarte2012@hotmail.com

Resumo

O Java 8 trouxe uma série de novas funcionalidades. Entre elas, há a implementação nessa versão de funções “lambdas” que ajudam a tornar a legibilidade de código ainda maior. Expressões lambdas também conseguem diminuir em muito as linhas de código e isso torna o seu uso extremamente recomendado, principalmente em sistemas voltado ao uso de Threads.

Palavras-chave: Orientação a Objetos, Lambda, Função Anônima.

1. Introdução

É natural a associação de funções lambdas do Java à funções anônimas do JavaScript ou mesmo as funções lambdas do Python. Em tese elas possuem muito em comum, mas na prática cada linguagem escolheu e definiu o seu uso. No java, elas não são tão anônimas assim, uma vez que podem ser associadas a variáveis e reutilizadas posteriormente pelo código em outra parte do sistema. Na verdade, esse é o objetivo! E indo além: com uma proposta de diminuição de trabalho em criação de funções ou métodos complexos.

Como foi supracitado, seu uso é extremamente visível em aplicações com uso de Threads, uma vez que a Lambda nesse caso exerce seu maior papel até então: LEGIBILIDADE. Essa tal legíbilidade pode ser vista no próximo tópico, com exemplos e demonstrações.

2. Seu uso no Java (Com Exemplos)

Imagine que em nosso programa precisemos de algo assim para criar uma Thread:

```
Runnable r = new Runnable() {
     public void run() {
          System.out.println("Thread com classer interna!");
     }
};
new Thread(r).start();
```

Foram necessárias cerca de 6 linhas de código para uma Thread que fizesse uma impressão no Console. Imagine que você precise ocupar menos espaço, uma vez que sua Classe esteja extremamente complexa e já se há problemas de legibilidade. Sua opção poderia ser transformar 6 em 2:

```
Runnable r = () -> System.out.println("Thread com função lambda!");
new Thread(r).start();
```

Outro uso comum de expressões lambdas são quando se está trabalhando com *Collections*. Para percorrer uma Lista do tipo *ArrayList<Integer>* com lambda, basta apenas:

```
List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7);
list.forEach(n -> System.out.println(n));
```

É preciso condicional? Basta inserir a condicional dentro da lambda:

```
list.forEach(n -> {
       if (n % 2 == 0) {
            System.out.println(n);
       }
});
```

Conclusão

É comum perceber que o uso da lambda ajuda em muito em deixar o código cada vez mais limpo. Entretanto, o uso de lambda, apesar de extremamente importante para atender ao princípio de legibilidade deve ser efetuado com cautela. Antes, deve ser feito uma avaliação se o trecho do código é algo que será realizado especificadamente uma única vez, ou se não terá nenhum uso posterior em outra parte da aplicação. Se por mínima chance que seja houver a necessidade de reutilizar esse trecho de código em outro local com outros argumentos ,pode ser necessário levar em conta que a criação de um método tornará o código mais reutilizável! Há muito o que se analisar na construção um software e o uso de lambda deve ser efetuado em casos excepcionais onde não há realmente a necessidade de criação de um método. Para uso em *Collections* por exemplo, deve ser utilizada em casos mais simples, uma vez que o método *.stream()* resolve muitos problemas que teriam que ser reinventados através de criação de uma função lambda