Programação I

Expressões Lambda e Interfaces Funcionais

Samuel da Silva Feitosa

Aula 17

Expressões Lambda

- Conceito do paradigma funcional incluído no Java 8, criando uma maneira concisa de definir métodos diretamente no local do seu uso.
  - Evita o emprego de classe anônimas e simplifica o uso de interfaces funcionais.
- Cria a possibilidade de definir funções de primeira classe.
  - Permite que uma função seja atribuída a uma variável, passada por parâmetro ou retornada em um método.

Expressões Lambda em Java

- É uma função anônima, ou seja, um método sem declaração do próprio nome.
- Sintaxe básica:
  - (lista parâmetros) -> expressão
  - O operador -> é lido como “que retorna” ou “que produz”.
- Exemplos:
  - (a) -> 2*a*a - 0.5*a - 1.3
  - (int n) -> { int s = 0;
    for (int i=1; i<n;i++) s+=i;
    return soma;
    }

Target types das expressões lambda

- Toda expressão lambda possui um *target type*.
  - Isso que permite diferenciar as entidades de primeira classe tipadas.
- Por exemplo: (unidade) -> unidade * 0.453
  - É possível deduzir o tipo de retorno double devido a presença de operadores aritméticos e literais double.
  - Enquanto que o tipo de unidade poderia ser *int*.
  - Logo, a assinatura seria double (int) .
- Outros Exemplos:
  - (a, b) -> a > b ? a : b
  - (n) -> (Math.log(n)/Math.log(2)) % 2 == 0
  - () -> Calendar.getInstance().get(Calendar.Month() + 1)

Aplicação das expressões lambda

- Considere um sistema de software projetado para efetuar um cálculo customizável.
  - double efetuarCalculo(Calculavel calc, double a, double b)
- O acionamento do método efetuarCalculo requer um objeto que implementa a interface Calculavel.
  - Vamos definir esta interface com apenas um método.
  - Interfaces com apenas um método são chamadas de *interfaces funcionais*.
  - É por meio do objeto que implementa a interface Calculavel que o método efetuarCalculo realiza o cálculo desejado.

Interface Calculável

- A interface Calculavel possui apenas um método e a anotação *@FunctionalInterface*.

```java
@FunctionalInterface
public interface Calculavel {
    double calcular(double a, double b);
}
```

Classe CalculavelImpl

- Podemos implementar uma classe que realiza a interface *Calculavel*.

```
public class CalculavelImpl implements Calculavel {
    @Override
    public double calcular(double a, double b) {
        return 0.4*a + 0.6*b;
    }
}
```

- Para suprir um objeto do tipo Calculavel para o método efetuarCalculo, podemos escrever:

```
CalculavelImpl obj = new CalculavelImpl();
double res = efetuarCalculo(obj, 15, 10);
```

Usando classe anônima

- Como é muito provável que a classe CalculavelImpl seja usada somente uma vez, uma classe anônima seria uma alternativa para suprir um objeto Calculavel.

```
double res = efetuarCalculo(new Calculavel() {
    @Override
    public double calcular(double a, double b) {
        return 0.4*a + 0.6*b;
    }
} , 15, 10);
```

- Embora evite a implementação de uma classe específica, sua legibilidade é questionável.

Usando Expressão Lambda

- Neste ponto surgem as vantagens das expressões lambda.
- Podemos escrever:

```
double res = efetuarCalculo((a, b) -> 0.4*a + 0.6*b, 15, 10);
```

- Isto funciona porque o compilador pode determinar a assinatura da expressão lambda fornecida como sendo double (double, double).
  - Esta assinatura é compatível com o único método requerido pela interface Calculavel.

Interfaces funcionais predefinidas

- Para facilitar o uso das expressões lambda, foi introduzido no Java 8 o pacote *java.util.function*.
  - Contém um número razoável de interfaces funcionais de propósito geral.
  - Estas interfaces podem ser agrupadas em funções (*Function*), ações (*Consumer* e *Supplier*) e predicados (*Predicate*).
- Essas interfaces funcionais podem ser usadas como *target type* de expressões lambda.
- Mais detalhes podem ser encontrados na documentação do Java.
  - https://docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html

Referências para métodos

- A introdução das expressões lambda no Java 8 também possibilitou a criação de referências para métodos ou construtores.
  - Desde que seus target types sejam compatíveis com alguma interface funcional.
  - Uma referência para método sempre pode ser usada onde uma expressão lambda é aceita e vice-versa.

Tipo de referência	Sintaxe
Método estático	NomeClasse::metodoEstatico
Método de instância de objeto específico	objeto::metodoInstancia
Método de instância de objeto de tipo arbitrário	NomeClasse::metodoInstancia
Construtor	nomeClasse::new

Exemplo - Referência para método

- Vejamos o mesmo exemplo definido anteriormente, agora utilizando uma referência para um método.

```
double res = efetuarCalculo(Math::pow, 2, 4);
```

- Podemos usar qualquer método que respeite a assinatura.

```
double res = efetuarCalculo(Math::max, 2, 4);
```

```
double res = efetuarCalculo(Math::min, 2, 4);
```

Considerações Finais

- Nesta aula estudamos o funcionamento da nova funcionalidade do Java, conhecida como expressões lambda.
  - Conceito vindo das linguagens funcionais e introduzido na versão 8 do Java.
- Esta funcionalidade permite a definição de funções anônimas.
  - Otimiza a escrita de métodos usados uma única vez.
  - Permite atribuir funções/métodos a variáveis, passar por parâmetro e retornar como resultado numa função.
- Será muito útil para trabalhar com as Collections.