PUC Minas
A B A K Ó S
Instituto de Ciências Exatas e Informática

Expressões Lambda e interfaces funcionais

Hand-out em PDF. Vídeo e código nos respectivos links
https://drive.google.com/drive/folders/1yKfz6fnG63HVjIeQU66xxy3uPVsRJiNL?usp=sharing
https://github.com/ProgMod/trabTeorico/tree/master/codigo

Daniel Henrique Rodrigues Costa
João Américo Martins Caiafa de Andrade
Pedro Henrique Magalhães Silva

1 INTERFACES FUNCIONAIS, EXPRESSÕES LAMBDA E REFERÊNCIAS DE MÉTODOS

A interface funcional: É aquela que possui somente um método abstrato.Esse método geralmente informa a finalidade da interface. A interface funcional costuma representar uma única ação. A interface funcional define o tipo de destino de uma expressão lambda.Uma expressão lambda só pode ser usada quando o tipo de destino é especificado. Esse recurso foi adicionado na versão JDK-8 e tem como papel fundamental otimizar como certas estruturas comuns são implementadas.

Expressão lambda: é um método anônimo (não nomeado) usado para implementar um método definido pela interface funcional. A expressão lambda é como uma classe anônima.

A expressão lambda introduz uma nova sintaxe "->". Essa sintaxe divide a expressão lambda em duas partes: O lado esquerdo especifica qualquer parâmetro requerido pela expressão lambda e o lado direito é o corpo lambda, no qual especifica as ações da expressão lambda. Uma expressão lambda não é executada por conta própria: ela serve como meio de implementar a interface funcional. Há dois tipos de corpos lambda: Por uma única expressão:

Exemplo: ( )-> 98,6; Esta expressão não tem parâmetros, sendo assim vazia.Ela retorna o valor constante 98,6.O tipo de retorno é um double;

Por bloco: Uma lambda de bloco expande as operações possíveis em uma operação lambda, pois permite que o corpo da expressão tenha várias instruções. Para criar uma lambda de bloco basta colocar chaves no corpo do bloco, como se faz em qualquer bloco de instruções.

Na lambda de bloco deve-se colocar a instrução return explicitamente, pois o bloco lambda não representa um única expressão.

Exemplos:

Lambda de expressão:

Interface numericTest{

boolean Test (int n,int n);

}

NumericTest isFactor= (n,d)-> (n%d)==0;

Interfaces funcionais: Define-se como método abstrato aquele que não possui implementação.Uma interface pode ter métodos padrões e static principalmente a partir do JDK8.

Código-exemplo:

Interface numericTest{

boolean Test (int n,int n);

}

A expressão lambda não é executada por conta própria.Em vez disso, ela forma a implementação do método abstrato definido pela interface funcional que especifica o seu destino.A expressão lambda só pode ser especificada em um contexto em que um tipo de destino seja definido.Um desses contextos é quando a expressão lambda é atribuída a uma referência de interface funcional:Outros tipos são: inicialização de variáveis, instruções return e argumentos de métodos.

1.1 Interfaces funcionais genéricas

As interfaces funcionais genéricas que consistem em um tipo genérico que pode ser modificado durante a implementação da interface funcional. Esse recurso é apropriado em casos onde há a necessidade de criar duas interface funcionais que possuam métodos iguais. Porém, é necessário colocar dentro dos parâmetros o tipo genérico e declarado na interface funcional.

Exemplo:

Interface SomeTest<T>{

Boolean test (Tn,Tm);

}

Neste caso, o T especifica o tipo dos dois parâmetros de test().

1.2 Passar expressão lambda como argumento

Fornece uma maneira de passar um código executável como argumento de um método.

1.3 Expressão lambda e a captura de variáveis

Uma variável da classe pode ser usada dentro de uma expressão lambda. O método da classe pode também ser usado dentro de uma expressão lambda. A variável da classe não pode ter i valor alterado dentro de uma expressão lambda, se não ocorrer um erro.

1.4 Exceção dentro de uma expressão lambda

Uma expressão lambda pode lançar uma exceção desde que essa seja compatível com a do método da interface funcional; Referencia de método: É um recurso da expressão lambda. Ema referencia de método serve para referenciá-lo sem executá-lo. Há 3 tipos de referência de métodos que são:

Referência a métodos static: Para declarar é necessário o nome da classe com o nome do método separados por dois pontos como no exemplo abaixo:

Nome Classe:: nome Metodo;

Referência de métodos de instância: A sintaxe é semelhante, mas nesse caso deve-se utilizar a referencia da classe e não o nome como no exemplo anterior.Exemplo:

Nome Classe referencia :: nome Metodo;

Referencia de construtor: É possível criar referencias a construtores desde que esta referencia passa ter o método compatível com a de interface funcional.

2 INTERFACES FUNCIONAIS PREDEFINIDAS

Na maioria dos casos não é necessário definir a própria interface funcional porque o JDK 8 adicionou um novo pacote chamado Java.util.function que fornece várias interfaces predefinidas.Principais interfaces:

UnaryOperator<T>: Aplica uma operação a um objeto de tipo T e retorna o resultado, que também é do tipo T. Seu método se chama apply( ).

BinaryOperator<T>: Aplica uma operação a dois objetos de tipo T e retorna o resultado, que também é de tipo T. Seu método se chama apply( ).

Consumer <T>: Aplica uma operação em um objeto de tipo T. Seu método se chama accept( );

Supplier<T>: Retorna um objeto do tipo T.Seu método se chama get();

Function<T,R>:Aplica uma operação a um objeto de tipo T e retorna o resultado como um objeto de tipo R.Seu método se chama apply();

Predicate<T>: Determina se um objeto de tipo T se enquadra em alguma restrição. Retorna um valor boolean que indica o resultado. Seu método se chama test( ).

3 JAVA COLLECTIONS FRAMEWORK COMBINADO COM LAMBDA

Java Collections Framework, introduzido no Java 2, teve sua utilidade ampliada no Java 5 - com Generics e sua grande possibilidade de tipos-, com a possibilidade de implementar classes de estruturas de dados já prontas, como list e arrayList, por exemplo.

Com o surgimento do lambda no Java 8, puderam ser efetuados métodos mais eficientes como foreach, iterator dentre outras que simplificam o código e motivam o uso dessa combinação para obter a eficiência desejada.

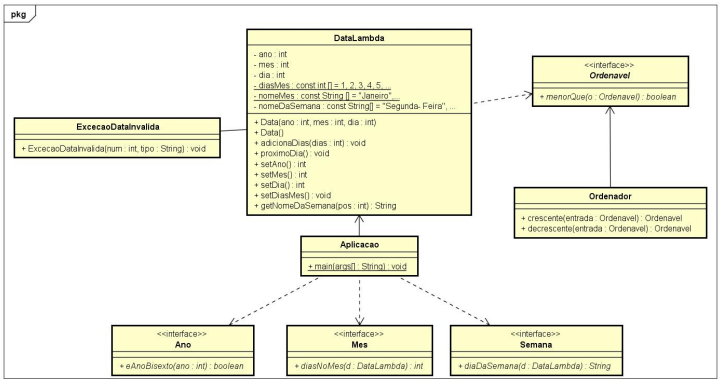
4 STREAMS API

A utilização de Streams se dá, predominantemente, no contexto da Java Collections Framework, sendo seu poder uma simplificação em termos de filtro, ordenação de dados, inserção e seleção por comparação, por exemplo, são life-savers em implementação por trazerem uma maior otimização em termos de funcionalidade por tamanho do código (por exemplo).

5 POR QUE UTILIZAR LAMBDA EM PROGRAMAÇÃO MODULAR E UML DO CÓDIGO DO TRABALHO

Lambda em si exige uma interface de um método externo a classe principal (aplicação ou etc), sendo a modularização, portanto, uma obrigação ao implementar lambda. Além desse fato, traz enormes vantagens, pois as interfaces apenas interagem com a main do código, portanto, a solução já modularizada.

Figura 1 – UML - código de exemplo



Link da UML:
$https://github.com/ProgMod/trabTeorico/blob/master/UML$

REFERÊNCIAS

SCHILDT, Herbert. Java para Iniciantes. 2015. 6ª ed. Porto Alegre. Bookman Editora.

Silva, C. (2016). Java 8: Iniciando o desenvolvimento com a Streams API. [online] Oracle.com. Disponível: http://www.oracle.com/technetwork/pt/articles/java/streams-api-java-8-3410098-ptb.html [acesso 2 de maio 2018].

ARHIPOV, A. Java 8: Lambdas & Java Collections. [online] zeroturnaround.com. Disponível: https://zeroturnaround.com/rebellabs/java-8-explained-applying-lambdas-to-java-collections/ [acesso 2 de maio 2018].