INSTITUTO FEDERAL
Rio Grande do Sul

Programação Funcional – Lambdas

PROF. TIAGO MORAES

Roteiro

- Introdução
- Programação Funcional no Java – antes dos lambdas
- Lambdas
- Interfaces Funcionais predefinidas
- Referências de método
- Streams

Introdução

- Programação Funcional (PF)
  - Trabalhar com referencias a funções, funções como parâmetros
  - Declarativo X Imperativo
    - Diz o que quer X diz como fazer
    - Por exemplo: printar uma lista:

```
List<Integer> lista = Arrays.asList(2,6,8,9,5);
```

  - E a POO?
  - POO em conjunto com programação funcional
    - Aumento do poder de expressão da linguagem
    - Otimizam a maneira de construir certas estruturas:

Introdução

- Programação Funcional (PF)
  - Trabalhar com referencias a funções, funções como parâmetros
  - Declarativo X Imperativo
    - Diz o que quer X diz como fazer
    - Por exemplo: printar uma lista:

```
List<Integer> lista = Arrays.asList(2,6,8,9,5);
```

Forma Declativa

```
lista.stream().foreach(System.out::println);
```

Forma Imperativa

```
for(Integer num: lista ){
    System.out.println(num);
}
```

  - E a POO?
  - POO em conjunto com programação funcional
    - Aumento do poder de expressão da linguagem
    - Otimizam a maneira de construir certas estruturas:

PF no Java – antes dos lambdas

- Utiliza classes anônimas que implementam interfaces ou classes abstratas
  - Normalmente com um método abstrato
  - Normalmente Interface
    - Exemplo: criar uma thread() deve implementar a interface Runnable e o método run

```
public static void main(String[] args){
    Thread t;
    t=new Thread( new Runnable(){
         @Override
         public void run() {
              System.out.println("ola");
          }
    });
    t.run();
}
```

PF no Java – antes dos lambdas

- Exemplo 2:
  - Com uma interface própria
    - Apesar de `new Funcional`, não é instância da interface, mas do tipo da interface (polimorfismo)
    - Necessita dar uma implementação para doisParaUm

```
public interface Funcional<T>{
    public T doisParaUm(T x, T y);
}
```

```
public static void main(String[] args){
    Funcional<String> anonima = new Funcional<String>(){
        @Override
        public String doisParaUm(String x, String y) {
            return x+y;
        }
    };
    System.out.println(anonima.doisParaUm("ola", " mundo"));
}
```

Objeto do tipo Funcional Implementação de classe anônima

Lambdas

- Lambdas – lançadas no Java 8 (2014)
  - Aumentou o poder da programação funcional no Java
    - Pois se tornou menos “verboso” (menos código e mais simples)
    - Possibilitou API’s da linguagem como Streams
    - Futuramente devem surgir outras API’s que utilizem lambdas
  - A construção de uma expressão lambda é uma forma mais simples e intuitiva para obter mesmo resultado
  - Utiliza inferência de tipos
  - Operador ->
    - Forma geral:

```
(parâmetros) -> {
        implementação
}
```

Lambdas

- Exemplo:
  - Com o Java 8, interfaces criadas para utilização com lambdas, devem ter a annotation *@FuncionalInterface*

```
@FuncionalInterface
public interface Funcional<T>{
    public T doisParaUm(T x, T y);
}
```

  - Opção 1:

```
public static void main(String[] args){
    Funcional<String> anonima = (String x, String y) -> {
            return x+y;
        };
    System.out.println(anonima.doisParaUm("ola", " mundo");
}
```

  - Opção 2: mais simples possível (Recomendável)

```
public static void main(String[] args){
    Funcional<String> anonima = (x, y) -> x+y;
    System.out.println(anonima.doisParaUm("ola", " mundo");
}
```

Lambdas

- Comparando a melhora no código com lambdas:
  - Com lambda

```
Funcional<String> anonima = (x, y) -> x+y;
```

  - Sem lambda

```
Funcional<String> anonima = new Funcional<String>(){
    @Override
    public String doisParaUm(String x, String y) {
        return x+y;
    }
};
```

- Quando utilizar lambdas:
  - Foram criadas para serem curtas!
  - De preferência 1 linha para não precisar usar as {}
  - Boa prática fala em no máxima 3 linhas...

Interfaces Funcionais Predefinidas

- O Java criou junto um conjunto de interfaces genéricas:
  - Pacote java.util.function
  - Para que não precisemos criar interfaces para utilizar lambdas
  - São utilizadas pela API Streams por exemplo
  - Sempre que possível é mais indicado utilizá-las

Interfaces Funcionais Predefinidas

- Principais:

Interface	Método abstrato	Finalidade
UnaryOperator<T>	T apply(T t)	Aplica uma operação unária a t e retorna objeto do mesmo tipo
BinaryOperator<T>	T apply(T t1, T t2)	Equivalente a UnaryOperator mas para dois parâmetros do mesmo tipo
Predicate<T>	boolean test(T t)	Verifica se t satisfaz alguma condição
Function<T, R>	R apply(T t)	Usa t para retornar um objeto do tipo R
Supplier<T>	T get()	Crie e retorna um objeto do tipo T
Consumer<T>	void accept(T t)	Utiliza o parâmetro t para realizar algo

Referências de Métodos

- Junto com os lambdas veio também a possibilidade de referenciar um método
  - Operador **::**
  - Quando se pode utilizar, facilita ainda mais a escrita (em relação a lambdas)
  - Exemplo com lambdas:

```
Consumer<String> op = (x) -> System.out.println(x);
op.accept("ola");
```

  - Equivalente com referência de métodos

```
Consumer<String> op = System.out::println;
op.accept("ola");
```

  - As referências a métodos podem ser feitas para métodos:
    - estáticos, de instância ou construtores

Referências de Métodos

- Outras possibilidades para as Interfaces Funcionais Predefinidas

Interface	Método abstrato	Exemplo
UnaryOperator<T>	T apply(T t)	String::toLowerCase
BinaryOperator<T>	T apply(T t1, T t2)	BigInteger:add
Predicate<T>	boolean test(T t)	Collectioon::isEmpty
Function<T, R>	R apply(T t)	Arrays::asList
Supplier<T>	T get()	LocalDate::now
Consumer<T>	void accept(T t)	System.out::println

Streams

- API que utiliza PF - trouxe a programação funcional para o Java (Java 8)
  - Muito utilizada com o collections.
- Vantagens:
  - Forma mais Eficiente de programar
  - Utilizará lambda : tendência das linguagens de uma maneira geral
- Stream pipeline: suas operações
  - **Fonte**: cria Stream
  - **Intermediárias**: zero ou mais operações de processamento: retorna Stream
  - **Terminal**: retorna algo não stream

Streams

- Exemplo:

- **Fonte** : Criação feita a partir de collections: listas, arrays etc
  - Método stream() que retorna uma stream

```
List<Integer> lista = Arrays.asList(2,7,8,9,5);
lista.stream();
```

- **Intermediárias** : também retornam streams e podem ser encadeados
  - skip(long), limit(long)
  - Filter(Predicate), map(Function),

```
List<Integer> lista = Arrays.asList(2,6,8,9,5);
lista.stream()
    .skip(2)
    .filter(x -> x%2==0);
```

Streams

- Exemplo:

- **Terminais**: Finalizam obtendo valor não stream
  - forEach(Consumer) : void
  - collect(Collector)
  - max(Comparator): Optional
  - count():long

```
List<Integer> lista = Arrays.asList(2,6,8,9,5);
lista = lista.stream()
    .skip(2)
    .filter(x -> x%2==0)
    .collect(Collectors.toList());
```