Curso Programação Orientada a Objetos com Java

Capítulo: Programação Funcional e Expressões Lambda

http://educandoweb.com.br

Prof. Dr. Nelio Alves

Uma experiência com Comparator

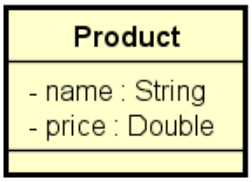
http://educandoweb.com.br

Prof. Dr. Nelio Alves

Problema

- Suponha uma classe Product com os atributos name e price.
- Podemos implementar a comparação de produtos por meio da implementação da interface Comparable<Product>
- Entretanto, desta forma nossa classe não fica fechada para alteração: se o critério de comparação mudar, precisaremos alterar a classe Product.
- Podemos então usar o default method "sort" da interface List:

```
default void sort(Comparator<? super E> c)
```



Comparator

https://docs.oracle.com/javase/10/docs/api/java/util/Comparator.html

Veja o método sort na interface List:
https://docs.oracle.com/javase/10/docs/api/java/util/List.html

Resumo da aula

- Comparator objeto de classe separada
- Comparator objeto de classe anônima
- Comparator objeto de expressão lambda com chaves
- Comparator objeto de expressão lambda sem chaves
- Comparator expressão lambda "direto no argumento"

https://github.com/acenelio/lambda1-java

Programação funcional e cálculo lambda

http://educandoweb.com.br

Prof. Dr. Nelio Alves

Paradigmas de programação

- **Imperativo** (C, Pascal, Fortran, Cobol)
- **Orientado a objetos** (C++, Object Pascal, Java (< 8), C# (< 3))
- **Funcional** (Haskell, Closure, Clean, Erlang)
- **Lógico** (Prolog)
- Multiparadigma (JavaScript, Java (8+), C# (3+), Ruby, Python, Go)

Paradigma funcional de programação

Baseado no formalismo matemático Cálculo Lambda (Church 1930)

	Programação Imperativa	**Programação Funcional**
Como se descreve algo a ser computado (*)	comandos ("como" - imperativa)	expressões ("o quê" - declarativa)
Funções possuem transparência referencial (ausência de efeitos colaterais)	fraco	forte
Objetos imutáveis (*)	raro	comum
Funções são objetos de primeira ordem	não	sim
Expressividade / código conciso	baixa	alta
Tipagem dinâmica / inferência de tipos	raro	comum

Transparência referencial

Uma função possui transparência referencial se seu resultado for sempre o mesmo para os mesmos dados de entrada. Benefícios: simplicidade e previsibilidade.

```
package application;

import java.util.Arrays;

public class Program {

     public static int globalValue = 3;

     public static void main(String[] args) {

          int[] vect = new int[] {3, 4, 5};
          changeOddValues(vect);
          System.out.println(Arrays.toString(vect));
     }

     public static void changeOddValues(int[] numbers) {
          for (int i=0; i<numbers.length; i++) {
               if (numbers[i] % 2 != 0) {
                    numbers[i] += globalValue;
               }
          }
     }
}
```

Exemplo de função que não é referencialmente transparente

Funções são objetos de primeira ordem (ou primeira classe)

Isso significa que funções podem, por exemplo, serem passadas como parâmetros de métodos, bem como retornadas como resultado de métodos.

```
public class Program {

     public static int compareProducts(Product p1, Product p2) {
          return p1.getPrice().compareTo(p2.getPrice());
     }

     public static void main(String[] args) {

          List<Product> list = new ArrayList<>();

          list.add(new Product("TV", 900.00));
          list.add(new Product("Notebook", 1200.00));
          list.add(new Product("Tablet", 450.00));

          list.sort(Program::compareProducts);

          list.forEach(System.out::println);
     }
}
```

Utilizamos aqui "method references"

Operador ::

Sintaxe:
`Classe::método`

Tipagem dinâmica / inferência de tipos

```
public static void main(String[] args) {

    List<Product> list = new ArrayList<>();

    list.add(new Product("TV", 900.00));
    list.add(new Product("Notebook", 1200.00));
    list.add(new Product("Tablet", 450.00));

    list.sort((p1, p2) -> p1.getPrice().compareTo(p2.getPrice()));

    list.forEach(System.out::println);
}
```

Expressividade / código conciso

```
Integer sum = 0;
for (Integer x : list) {
    sum += x;
}
```

vs.

```
Integer sum = list.stream().reduce(0, Integer::sum);
```

O que são "expressões lambda"?

Em programação funcional, expressão lambda corresponde a uma função anônima de primeira classe.

```
public class Program {

    public static int compareProducts(Product p1, Product p2) {
        return p1.getPrice().compareTo(p2.getPrice());
    }

    public static void main(String[] args) {

        (...)

        list.sort(Program::compareProducts);

        list.sort((p1, p2) -> p1.getPrice().compareTo(p2.getPrice()));

        (...)
```

Resumo da aula

	Programação Imperativa	**Programação Funcional**
Como se descreve algo a ser computado (*)	comandos ("como" - imperativa)	expressões ("o quê" - declarativa)
Funções possuem transparência referencial (ausência de efeitos colaterais)	fraco	forte
Objetos imutáveis (*)	raro	comum
Funções são objetos de primeira ordem	não	sim
Expressividade / código conciso	baixa	alta
Tipagem dinâmica / inferência de tipos	raro	comum

Cálculo Lambda = formalismo matemático base da programação funcional

Expressão lambda = função anônima de primeira classe

Interface funcional

http://educandoweb.com.br

Prof. Dr. Nelio Alves

Interface funcional

É uma interface que possui um único método abstrato. Suas implementações serão tratadas como expressões lambda.

```
public class MyComparator implements Comparator<Product> {

    @Override
    public int compare(Product p1, Product p2) {
        return p1.getName().toUpperCase().compareTo(p2.getName().toUpperCase());
    }
}
```

```
public static void main(String[] args) {

    (...)

    list.sort(new MyComparator());
```

Algumas outras interfaces funcionais comuns

- Predicate
  - https://docs.oracle.com/javase/8/docs/api/java/util/function/Predicate.html
- Function
  - https://docs.oracle.com/javase/8/docs/api/java/util/function/Function.html
- Consumer
  - https://docs.oracle.com/javase/8/docs/api/java/util/function/Consumer.html
  - Nota: ao contrário das outras interfaces funcionais, no caso do Consumer, é esperado ele possa gerar efeitos colaterais

Predicate (exemplo com removelf)

http://educandoweb.com.br

Prof. Dr. Nelio Alves

Predicate

https://docs.oracle.com/javase/10/docs/api/java/util/function/Predicate.html

```
public interface Predicate<T> {

    boolean test(T t);
}
```

Problema exemplo

Fazer um programa que, a partir de uma lista de produtos, remova da lista somente aqueles cujo preço mínimo seja 100.

```
List<Product> list = new ArrayList<>();

list.add(new Product("Tv", 900.00));
list.add(new Product("Mouse", 50.00));
list.add(new Product("Tablet", 350.50));
list.add(new Product("HD Case", 80.90));
```

https://github.com/acenelio/lambda2-java

Versões:

- Implementação da interface
- Reference method com método estático
- Reference method com método não estático
- Expressão lambda declarada
- Expressão lambda inline

Consumer (exemplo com forEach)

http://educandoweb.com.br

Prof. Dr. Nelio Alves

Consumer

https://docs.oracle.com/javase/10/docs/api/java/util/function/Consumer.html

```
public interface Consumer<T> {

    void accept(T t);
}
```

Problema exemplo

Fazer um programa que, a partir de uma lista de produtos, aumente o preço dos produtos em 10%.

```
List<Product> list = new ArrayList<>();

list.add(new Product("Tv", 900.00));
list.add(new Product("Mouse", 50.00));
list.add(new Product("Tablet", 350.50));
list.add(new Product("HD Case", 80.90));
```

https://github.com/acenelio/lambda3-java

Function (exemplo com map)

http://educandoweb.com.br

Prof. Dr. Nelio Alves

Function

https://docs.oracle.com/javase/10/docs/api/java/util/function/Function.html

```
public interface Function<T, R> {

    R apply(T t);
}
```

Problema exemplo

Fazer um programa que, a partir de uma lista de produtos, gere uma nova lista contendo os nomes dos produtos em caixa alta.

```
List<Product> list = new ArrayList<>();

list.add(new Product("Tv", 900.00));
list.add(new Product("Mouse", 50.00));
list.add(new Product("Tablet", 350.50));
list.add(new Product("HD Case", 80.90));
```

https://github.com/acenelio/lambda4-java

Nota sobre a função map

- A função "map" (não confunda com a estrutura de dados Map) é uma função que aplica uma função a todos elementos de uma stream.
- Conversões:
  - List para stream: `.stream()`
  - Stream para List: `.collect(Collectors.toList())`

Criando funções que recebem funções como argumento

http://educandoweb.com.br

Prof. Dr. Nelio Alves

Recordando

- removeIf(Predicate)
- foreach(Consumer)
- map(Function)

Problema exemplo

Fazer um programa que, a partir de uma lista de produtos, calcule a soma dos preços somente dos produtos cujo nome começa com "T".

```
List<Product> list = new ArrayList<>();

list.add(new Product("Tv", 900.00));
list.add(new Product("Mouse", 50.00));
list.add(new Product("Tablet", 350.50));
list.add(new Product("HD Case", 80.90));
```

1250.50

https://github.com/acenelio/lambda5-java

Stream

http://educandoweb.com.br

Prof. Dr. Nelio Alves

Stream

- É uma sequencia de elementos advinda de uma fonte de dados que oferece suporte a "operações agregadas".
  - Fonte de dados: coleção, array, função de iteração, recurso de E/S
- Sugestão de leitura:
  http://www.oracle.com/technetwork/pt/articles/java/streams-api-java-8-3410098-ptb.html

Características

- Stream é uma solução para processar sequências de dados de forma:
  - Declarativa (iteração interna: escondida do programador)
  - Parallel-friendly (imutável -> thread safe)
  - Sem efeitos colaterais
  - Sob demanda (lazy evaluation)
- Acesso sequencial (não há índices)
- Single-use: só pode ser "usada" uma vez
- **Pipeline**: operações em streams retornam novas streams. Então é possível criar uma cadeia de operações (fluxo de processamento).

Operações intermediárias e terminais

- O pipeline é composto por zero ou mais operações intermediárias e uma terminal.
- Operação intermediária:
  - Produz uma nova streams (encadeamento)
  - Só executa quando uma operação terminal é invocada (lazy evaluation)
- Operação terminal:
  - Produz um objeto não-stream (coleção ou outro)
  - Determina o fim do processamento da stream

Operações intermediárias

- filter
- map
- flatmap
- peek
- distinct
- sorted
- skip
- limit (*)

** short-circuit*

Operações terminais

- forEach
- forEachOrdered
- toArray
- reduce
- collect
- min
- max
- count
- anyMatch (*)
- allMatch (*)
- noneMatch (*)
- findFirst (*)
- findAny (*)

** short-circuit*

Criar uma stream

- Basta chamar o método `stream()` ou `parallelStream()` a partir de qualquer objeto Collection.
  https://docs.oracle.com/javase/10/docs/api/java/util/Collection.html

- Outras formas de se criar uma stream incluem:
  - Stream.of
  - Stream.ofNullable
  - Stream.iterate

Demo - criação de streams

```
List<Integer> list = Arrays.asList(3, 4, 5, 10, 7);
Stream<Integer> st1 = list.stream();
System.out.println(Arrays.toString(st1.toArray()));


Stream<String> st2 = Stream.of("Maria", "Alex", "Bob");
System.out.println(Arrays.toString(st2.toArray()));


Stream<Integer> st3 = Stream.iterate(0, x -> x + 2);
System.out.println(Arrays.toString(st3.limit(10).toArray()));


Stream<Long> st4 = Stream.iterate(new long[]{ 0L, 1L }, p->new long[]{ p[1], p[0]+p[1] }).map(p -> p[0]);
System.out.println(Arrays.toString(st4.limit(10).toArray()));
```

Pipeline (demo)

http://educandoweb.com.br

Prof. Dr. Nelio Alves

Demo - pipeline

```
List<Integer> list = Arrays.asList(3, 4, 5, 10, 7);

Stream<Integer> st1 = list.stream().map(x -> x * 10);
System.out.println(Arrays.toString(st1.toArray()));

int sum = list.stream().reduce(0, (x, y) -> x + y);
System.out.println("Sum = " + sum);

List<Integer> newList = list.stream()
    .filter(x -> x % 2 == 0)
    .map(x -> x * 10)
    .collect(Collectors.toList());
System.out.println(Arrays.toString(newList.toArray()));
```

Exercício resolvido - filter, sorted, map, reduce

http://educandoweb.com.br

Prof. Dr. Nelio Alves

Fazer um programa para ler um conjunto de produtos a partir de um arquivo em formato .csv (suponha que exista pelo menos um produto).

Em seguida mostrar o preço médio dos produtos. Depois, mostrar os nomes, em ordem decrescente, dos produtos que possuem preço inferior ao preço médio.

Veja exemplo na próxima página.

https://github.com/acenelio/lambda6-java

Input file:

```
Tv,900.00
Mouse,50.00
Tablet,350.50
HD Case,80.90
Computer,850.00
Monitor,290.00
```

Execution:

```
Enter full file path: c:\temp\in.txt
Average price: 420.23
Tablet
Mouse
Monitor
HD Case
```

https://github.com/acenelio/lambda6-java

Exercício de fixação

http://educandoweb.com.br

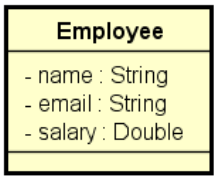
Prof. Dr. Nelio Alves

Fazer um programa para ler os dados (nome, email e salário) de funcionários a partir de um arquivo em formato .csv.

Em seguida mostrar, em ordem alfabética, o email dos funcionários cujo salário seja superior a um dado valor fornecido pelo usuário.

Mostrar também a soma dos salários dos funcionários cujo nome começa com a letra 'M'.

Veja exemplo na próxima página.



https://github.com/acenelio/lambda7-java

Input file:

```
Maria,maria@gmail.com,3200.00
Alex,alex@gmail.com,1900.00
Marco,marco@gmail.com,1700.00
Bob,bob@gmail.com,3500.00
Anna,anna@gmail.com,2800.00
```

Execution:

```
Enter full file path: c:\temp\in.txt
Enter salary: 2000.00
Email of people whose salary is more than 2000.00:
anna@gmail.com
bob@gmail.com
maria@gmail.com
Sum of salary of people whose name starts with 'M': 4900.00
```

https://github.com/acenelio/lambda7-java