

Expressões lambda

Interfaces funcionais

UA.DETI.POO

Cálculo lambda

- ❖ As linguagens de programação funcional são baseadas no cálculo lambda (cálculo- λ).
 - Lisp, Haskell, Scheme
- ❖ O cálculo lambda pode ser visto como uma linguagem de programação abstrata em que funções podem ser combinadas para formar outras funções.
- ❖ Ideia geral: formalismo matemático
 - $x \rightarrow f(x)$ i.e. x é transformado em $f(x)$
- ❖ O cálculo lambda trata as funções como *elementos de primeira classe*
 - podem ser utilizadas como argumentos e retornadas como valores de outras funções.

Sintaxe

- ❖ Uma expressão lambda descreve uma função anónima. Representa-se na forma:
 - (argument) -> (body)
`(int a, int b) -> { return a + b; }`
- ❖ Pode ter zero, um, ou mais argumentos
 - () -> { body }
`() -> System.out.println("Hello World");`
 - (arg1, arg2...) -> { body }
- ❖ O tipo dos argumentos pode ser explicitamente declarado ou inferido
 - (type1 arg1, type2 arg2...) -> { body }
`(int a, int b) -> { return a + b; }`
`a -> return a*a // um argumento – podemos omitir os parêntesis`
- ❖ O corpo (body) pode ter uma ou mais instruções

Exemplos

lambda expression	equivalent method
<code>() -> { System.gc(); }</code>	<code>void nn() { System.gc(); }</code>
<code>(int x) -> { return x+1; }</code>	<code>int nn(int x) return x+1; }</code>
<code>(int x, int y) -> { return x+y; }</code>	<code>int nn(int x, int y) { return x+y; }</code>
<code>(String... args) ->{return args.length;}</code>	<code>int nn(String... args) { return args.length; }</code>
<code>(String[] args) -> { if (args != null) return args.length; else return 0; }</code>	<code>int nn(String[] args) { if (args != null) return args.length; else return 0; }</code>

Como usar?

- ❖ Uma expressão lambda não pode ser declarada isoladamente

`(n) -> (n % 2)==0 // Erro de compilação`

- ❖ Precisamos de outro mecanismo adicional
 - Interfaces funcionais
 - onde as expressões lambda passam a ser implementações de métodos abstratos.
 - O compilador Java converte uma expressão lambda num método da classe (isto é um processo interno).

Functional interface

❖ Contém apenas um método abstrato

❖ Exemplo

– Dada a interface:

```
@FunctionalInterface
interface MyNum {
    double getNum(double n);
}
```



– Podemos usar:

```
public class Lambda1 {
    public static void main(String[] args) {
        MyNum n1 = (x) -> x+1;
        // qualquer expressão que transforme double em double
        System.out.println(n1.getNum(10));
        n1 = (x) -> x*x;
        System.out.println(n1.getNum(10));
    }
}
```

```
11.0
100.0
```

Exemplos

@FunctionalInterface

```
interface Ecra {  
    void escreve(String s);  
}
```

interface funcional

```
public class Lambda2 {
```

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

```
        Ecra xd = (String s) -> {  
            if (s.length() > 2)  
                System.out.println(s);  
            else  
                System.out.println("..");  
        };
```

```
        xd.escreve("Lambda print");
```

```
        xd.escreve("?");
```

```
    }
```

```
}
```

Lambda print

..

Exemplos

```
// Another functional interface.
```

```
interface NumericTest {  
    boolean test(int n);  
}
```

```
class Lambda3 {  
    public static void main(String args[]) {  
        // A lambda expression that tests if a number is even.  
        NumericTest isEven = (n) -> (n % 2) == 0;  
        if (isEven.test(10)) System.out.println("10 is even");  
        if (!isEven.test(9)) System.out.println("9 is not even");  
        // A lambda expression that tests if a number is non-negative.  
        NumericTest isNonNeg = (n) -> n >= 0;  
        if (isNonNeg.test(1)) System.out.println("1 is non-negative");  
        if (!isNonNeg.test(-1)) System.out.println("-1 is negative");  
    }  
}
```

```
10 is even  
9 is not even  
1 is non-negative  
-1 is negative
```


Exemplos

// Demonstrate a lambda expression that takes two parameters.

```
interface NumericTest2 {  
    boolean test(int n, int d);  
}
```

```
public class Lambda4 {  
    public static void main(String args[]) {  
        // This lambda expression determines if one number is  
        // a factor of another.  
        NumericTest2 isFactor = (n, d) -> (n % d) == 0;  
        if (isFactor.test(10, 2))  
            System.out.println("2 is a factor of 10");  
        if (!isFactor.test(10, 3))  
            System.out.println("3 is not a factor of 10");  
    }  
}
```

2 is a factor of 10

3 is not a factor of 10

Expressões Lambda como argumento

- ❖ Podemos definir interfaces genéricas (com parâmetros).
- ❖ Por exemplo:

```
interface MyFunc<T> {  
    T func(T n);  
}
```

...

// Função que aceita uma expressão lambda e o seu argumento (T n)

```
static String stringOp(MyFunc<String> sf, String s) {  
    return sf.func(s);  
}
```

Interface funcional

Argumento da interface

...

// Outro exemplo

```
static Person PersonOp(MyFunc<Person> sf, Person s) {  
    return sf.func(s);  
}
```

Expressões Lambda como argumento

❖ Utilização

```
String inStr = "Lambdas add power to Java";
String outStr = stringOp((str) -> str.toUpperCase(), inStr);
System.out.println("The string in uppercase: " + outStr);
// This passes a block lambda that removes spaces.
outStr = stringOp((str) -> {
    StringBuilder result = new StringBuilder();
    for(int i = 0; i < str.length(); i++)
        if(str.charAt(i) != ' ')
            result.append( str.charAt(i) );
    return result.toString();
}, inStr);
System.out.println("The string with spaces removed: " + outStr);
```






```
The string in uppercase: LAMBDAAS ADD POWER TO JAVA
The string with spaces removed: LambdasaddpowertoJava
```

Interfaces funcionais pré-definidas

- ❖ Geralmente não precisamos de criar novas interfaces funcionais

- Utilizamos as que já existem definidas em Java.

- ❖ Exemplos

- `java.util.function.Predicate<T>`
`boolean test(T t)`
 - `java.util.function.Consumer<T>`
`void accept(T t)`
 - `java.util.function.Function<T, R>`
`R apply(T t)`
 - `java.util.function.Supplier<T>`
`T get()`
 - `java.util.Comparator<T>`
`int compare(T o1, T o2)`

Referência a métodos

- ❖ São um tipo especial de expressões lambda.
 - Permitem substituir expressões lambda que invocam métodos existentes.

- ❖ Exemplos

- Podemos substituir:

- `str -> System.out.println(str)`

- `(s1, s2) -> {return s1.compareToIgnoreCase(s2); }`

- por:

- `System.out::println`

- `String::compareToIgnoreCase`

```
String[] names = { "Steve", "Rick", "Aditya", "Negan", "Lucy", "Sansa"};
Arrays.sort(names, String::compareToIgnoreCase);
for(String str: names){
    System.out.println(str);
}
```

Referências a métodos: 4 variedades

Kind	Syntax/Examples	Equivalent to
Reference to a static method	Class::staticMethod Math::abs Double::compare Math::random	(args) -> Class.staticMethod(args) (x) -> Math.abs(x) (x, y) -> Double.compare(x, y) () -> Math.random()
Reference to an instance method of a particular object	obj::method System.out::println "abcdef"::substring	(args) -> obj.method(args) (s) -> System.out.println(s) (a, b) -> "abcdef".substring(a, b)
Reference to an instance method of arbitrary object of a particular type	Type::method String::compareTo String::strip	(arg1, args) -> arg1.method(args) (s, t) -> s.compareTo(t) (s) -> s.strip()
Reference to a constructor	Class::new File::new int[]::new	(args) -> new Class(args) (name) -> new File(name) (size) -> new int[size]

[Method references \(Java tutorial\)](#)

Utilização de expressões lambda

❖ Iterar sobre Java Collections

// solução 1

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

// solução 2

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

// solução 3, method reference (:: double colon operator)

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

TreeSet – ordenação natural

```
public class Test {  
    public static void main(String args[]) {  
        TreeSet<String> ts = new TreeSet<>();  
  
        ts.add("viagem");  
        ts.add("calendário");  
        ts.add("prova");  
        ts.add("zircónio");  
        ts.add("ilha do sal");  
        ts.add("avião");  
        for (String element : ts)  
            System.out.println(element + " ");  
    }  
}
```

avião
calendário
ilha do sal
prova
viagem
zircónio

TreeSet – ordem definida

TreeSet aceita um `java.util.Comparator<T>`


```
public class Test {  
    public static void main(String args[]) {  
        TreeSet<String> ts =  
            new TreeSet<>(Comparator.comparing(String::length));  
  
        ts.add("viagem");  
        ts.add("calendário");  
        ts.add("prova");  
        ts.add("zircórnio");  
        ts.add("ilha do sal");  
        ts.add("avião");  
        for (String element : ts)  
            System.out.println(element + " ");  
    }  
}
```

prova
viagem
zircórnio
calendário
ilha do sal

TreeSet – ordem definida

TreeSet aceita um `java.util.Comparator<T>`

```
public class Test {  
    public static void main(String args[]) {  
        Set<String> ts =  
            new TreeSet<>(Comparator.comparing(String::length));  
  
        ts.add("viagem");  
        ts.add("calendário");  
        ts.add("prova");  
        ts.add("zircórnio");  
        ts.add("ilha do sal");  
        ts.add("avião");  
        for (String element : ts)  
            System.out.println(element + " ");  
    }  
}
```

 código
equivalente

```
TreeSet<String> ts = new TreeSet<>((s1, s2) -> {  
    if (s1.length() > s2.length())  
        return 1;  
    else if (s1.length() < s2.length())  
        return -1;  
    else  
        return 0;  
});
```

```
prova  
viagem  
zircórnio  
calendário  
ilha do sal
```

Algoritmos

- ❖ As bibliotecas de Java fornecem um conjunto de algoritmos que podem ser usados em coleções e vetores
- ❖ Duas classes abstratas fornecem métodos estáticos de utilização global
 - `java.util.Collections` - Note que é diferente de `java.util.Collection` (interface)!!
 - `java.util.Arrays` - Classe que contém vários métodos para manipular vetores (ordenação, pesquisa, ..). Também permite converter vectores para listas.
- ❖ Exemplos de métodos:
 - `sort`, `binarySearch`, `copy`, `shuffle`, `reverse`, `max`, `min`, etc.

java.util.Collections

Ordenação natural

```
public static void main(String[] args) {  
    List<Integer> list = new ArrayList<>();  
  
    for (int i=0;i<10;i++) {  
        list.add((int) (Math.random() * 100));  
    }  
  
    System.out.println("Initial List: "+list);  
    Collections.sort(list);  
    System.out.println("Sorted List: "+list);  
    Collections.reverse(list);  
    System.out.println("Reverse List: "+list);  
}
```

Initial List: [53, 46, 6, 93, 13, 57, 76, 56, 40, 93]
Sorted List: [6, 13, 40, 46, 53, 56, 57, 76, 93, 93]
Reverse List: [93, 93, 76, 57, 56, 53, 46, 40, 13, 6]

java.util.Collections

Ordenação com Comparator

```
public static void main(String[] args) {  
    System.out.println("--Sorting with natural order");  
    List<String> l1 = createList();  
    Collections.sort(l1);  
    l1.forEach(System.out::println);  
  
    System.out.println("--Sorting with a lambda expression");  
    List<String> l2 = createList();  
    l2.sort((s1, s2) -> s1.compareTo(s2));  
    l2.forEach(System.out::println);  
  
    System.out.println("--Sorting with a method reference");  
    List<String> l3 = createList();  
    l3.sort(String::compareTo);  
    l3.forEach(System.out::println);  
}  
  
private static List<String> createList() {  
    List<String> list = new ArrayList<>();  
    list.add("Ubuntu");  
    list.add("Android");  
    list.add("MacOS");  
    return list;  
}
```

```
--Sorting with natural order  
Android  
MacOS  
Ubuntu  
--Sorting with a lambda expression  
Android  
MacOS  
Ubuntu  
--Sorting with a method reference  
Android  
MacOS  
Ubuntu
```

java.util.Arrays - Exemplo

```
public static void main(String[] args) {  
    String[] vec1 =  
        new String[] { "once", "upon", "a", "time", "in", "Aveiro" };  
    display(vec1);  
    String[] res1 = Arrays.copyOfRange(vec1, 0, 3);  
    display(res1);  
    Arrays.sort(vec1);  
    display(vec1);  
    Arrays.sort(vec1, Comparator.comparing(String::length));  
    display(vec1);  
    String[] vec2 = new String[10];  
    Arrays.fill(vec2, "UA");  
    System.out.println(Arrays.toString(vec2)); // em vez de display()  
    List<String> list1 = Arrays.asList(vec1);  
    list1.forEach(System.out::println);  
}  
  
public static void display(String[] vec) {  
    for (String s : vec) System.out.print(s + " ");  
    System.out.println();  
}
```

```
once upon a time in Aveiro  
once upon a  
Aveiro a in once time upon  
a in once time upon Aveiro  
[UA, UA, UA, UA, UA, UA, UA, UA, UA, UA]  
a  
in  
once  
time  
upon  
Aveiro
```

Sumário

- ❖ Funções lambda
- ❖ Interfaces funcionais
- ❖ Ordenação de vetores, listas, árvores, ..
- ❖ `java.util.Collections`
- ❖ `java.util.Arrays`