

Classe Object

Todas as classes Java herdam implicitamente da classe Object

Esta classe tem alguns métodos que, por herança, podemos assumir que existem em todos os objectos que criamos

```
public class Object {  
    protected Object clone() { ... }  
    public boolean equals(Object obj) { ... }  
    public String toString() { ... }  
    ...  
}
```

Classe Object

Todas as classes Java herdam implicitamente da classe Object

Esta classe tem alguns métodos que, por herança, podemos assumir que existem em todos os objectos que criamos

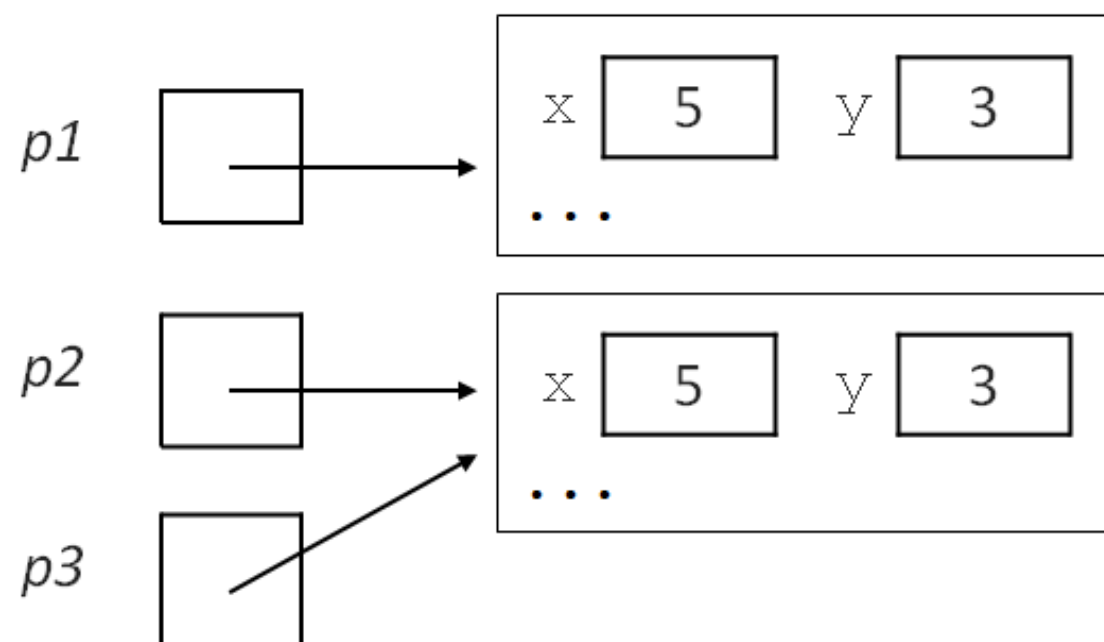
```
public class Object {  
    protected Object clone() { ... }  
    public boolean equals(Object obj) { ... }  
    public String toString() { ... }  
    ...  
}
```

Comparação de objetos

```
class Point {  
    private int x,y;  
  
    public Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}  
  
public class App {  
    public static void main(String[] args) {  
  
        Point p1 = new Point(5,3);  
        Point p2 = new Point(5,3);  
        Point p3 = p2;  
  
        // p1 == p2 is false  
        // p1 == p3 is false  
        // p2 == p3 is true  
  
    }  
}
```

==

compara referências para
objetos, não o seu
conteúdo



Comparação de objetos

```
class Object {
    public boolean equals(java.lang.Object obj) {
        return this == obj;
    }
}

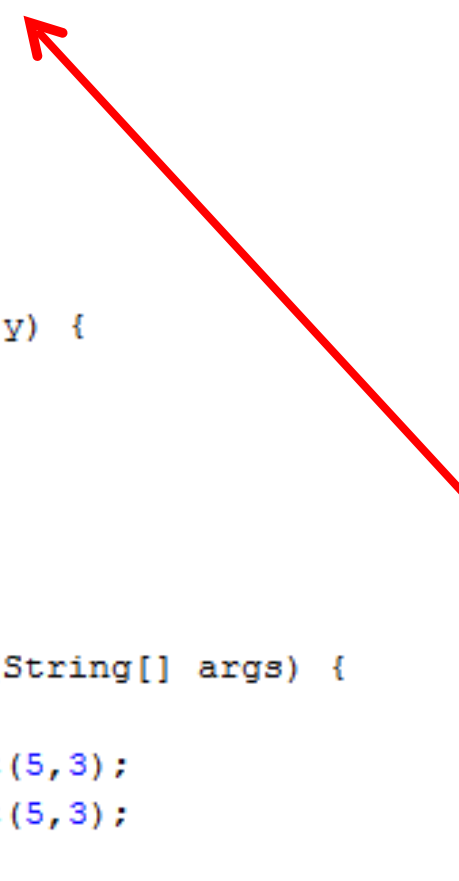
class Point extends Object {
    private int x,y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}

public class App {
    public static void main(String[] args) {

        Point p1 = new Point(5,3);
        Point p2 = new Point(5,3);
        Point p3 = p2;

        // p1.equals(p2) is false
        // p1.equals(p3) is false
        // p2.equals(p3) is true
    }
}
```



equals()

deve ser usado para
comparar o conteúdo de
objetos mas cuidado:

**Tem que ser redefinido
pelas classes respectivas**

A implementação do Object compara
referências e não o conteúdo

Comparação de objetos

```
class Point {
    private int x,y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

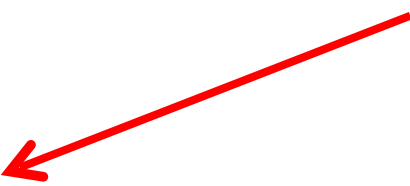
    @Override
    public boolean equals(Object otherObj) {
        Point otherPoint = (Point) otherObj;
        return (this.x == otherPoint.x) && (this.y == otherPoint.y);
    }
}

public class App {
    public static void main(String[] args) {

        Point p1 = new Point(5,3);
        Point p2 = new Point(5,3);
        Point p3 = p2;

        // p1.equals(p2) is true
        // p1.equals(p3) is true
        // p2.equals(p3) is true
    }
}
```

Deve-se redefinir o equals() para comparar o conteúdo (estado do objeto)



Para tipos primitivos (int, long, etc.) pode-se usar o ==



Comparação de objetos

Mas em que situações é que eu preciso de implementar o equals()?

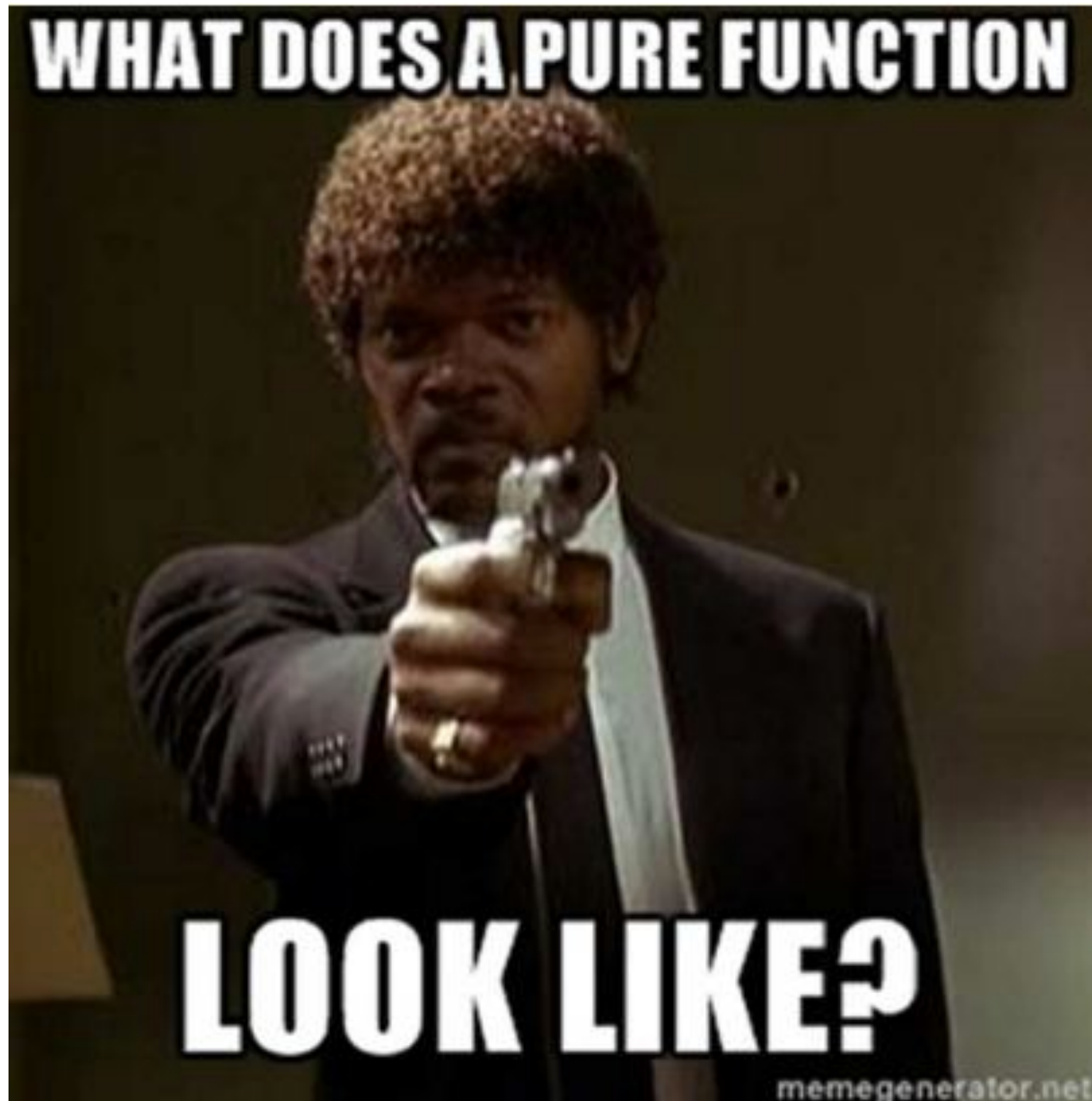
- Verificar se uma lista contém um certo objecto

```
if (lista.contains(new Musica("So what", "Miles Davis"))) {  
    ...  
}
```

- Garantir que não há duplicados num Set

```
Set<Musica> musicas = new HashSet<>();  
musicas.add(new Musica("So what", "Miles Davis"));  
musicas.add(new Musica("So what", "Miles Davis")); // ignora
```

Programação funcional



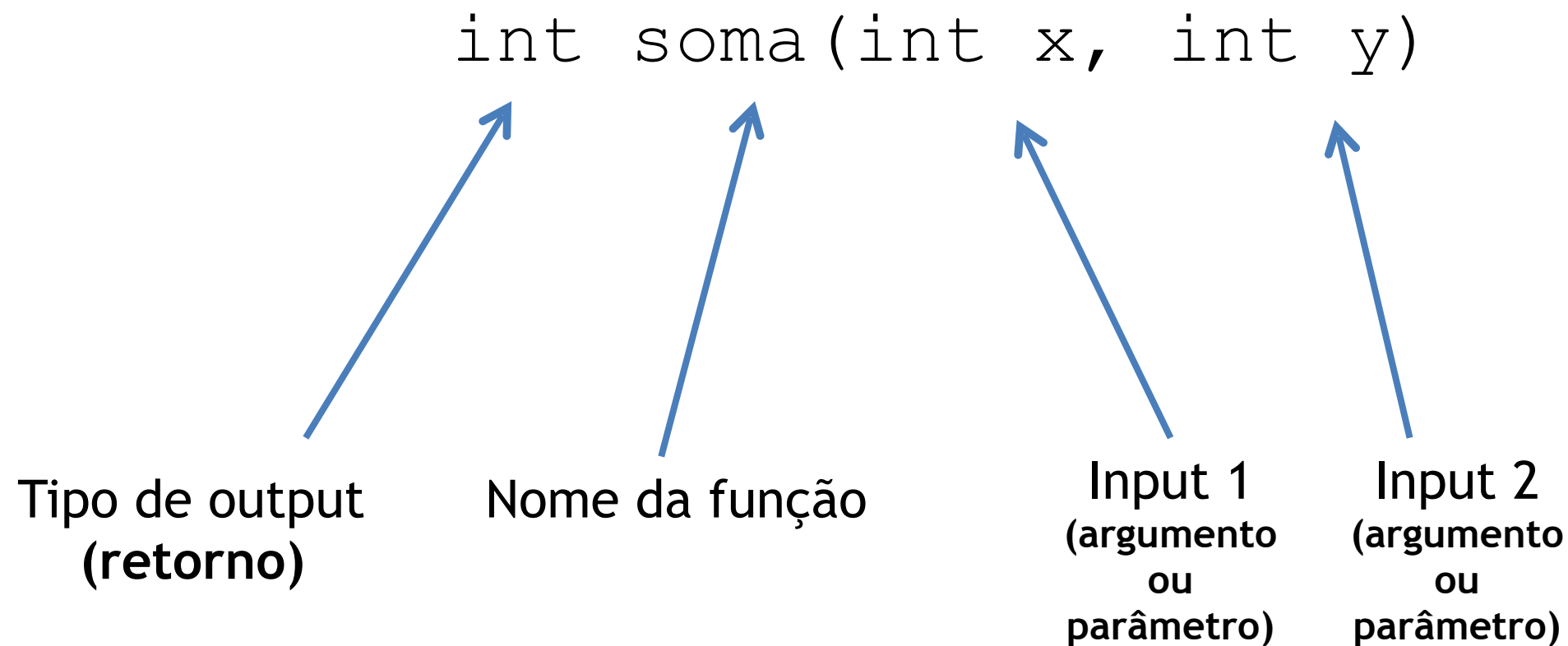
Paradigmas de programação

Paradigma	Descrição	Bom para...	Linguagens
Imperativo	Conjunto de instruções que afetam variáveis e são executadas de forma sequencial	Problemas algorítmicos simples (ex: ordenar um array)	Assembly, C
Funcional	Conjunto de funções que chamam funções, passando o estado (variáveis) entre elas	Programação paralela (ex: processamento de grandes quantidades de informação usando múltiplos computadores)	Haskell, Erlang, Lisp, Javascript, Java 8
Orientado a objectos	Conjunto de entidades que comunicam entre si. Cada entidade agrega estado e comportamento	Problemas complexos envolvendo muitas entidades (ex: gestão de funcionários de uma empresa)	C#, Java, Ruby, Python

Java 8 - Programação funcional

(Revisões de Fundamentos de Programação)

Funções



`<tipo_output> <nome_da_função> (<tipo_input> <identificador>, <tipo_input> <identificador>, ...)`

Java 8 - Programação funcional

Quando declaramos variáveis, temos que indicar o seu tipo:

- byte, short, int, long
- char
- boolean
- String
- array (ex: char[])
- objecto (ex: Carro)

Java 8 - Programação funcional

Quando declaramos variáveis, temos que indicar o seu tipo:

- byte, short, int, long
- char
- boolean
- String
- array (ex: char[])
- objecto (ex: Carro)
- função << em programação funcional, uma variável pode ser do tipo função

Exemplo (imperativo)

```
public class Matematica {  
    static Integer calculaMaximo(Integer[] numeros) {  
        Integer max = Integer.MIN_VALUE;  
        for (Integer numero: numeros) {  
            if (numero > max) {  
                max = numero;  
            }  
        }  
        return max;  
    }  
  
    public static void main(String[] args) {  
        Integer[] numeros = new Integer[] { 3, 7, 9, 2 };  
        Integer maximo = calculaMaximo(numeros);  
    }  
}
```

Programação imperativa, tal como aprendemos em FP

Java 8 - Programação funcional

```
static Integer calculaMaximo(Integer[] numeros) {  
    // ...  
}
```

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

```
    Function<Integer[],Integer> funcaoMax = Matematica::calculaMaximo;
```

```
}
```

A variável “funcaoMax”:

- é do tipo `Function<Integer[],Integer>`
- tem o valor `Matematica::calculaMaximo`

Java 8 - Programação funcional

```
Function<Integer[],Integer> funcaoMax;
```

O tipo Function recebe dois parâmetros, o tipo do Input (neste caso Integer[]) e o tipo do Output (neste caso Integer)

Atenção: O tipo function apenas funciona para funções:

- com um argumento
- que sejam static

Java 8 - Programação funcional

```
funcaoMax = Matematica::calculaMaximo;
```

Para referenciar uma função, usamos o nome da classe, seguido de “::”, seguido do nome da função

Nota: Isto só funciona para funções static

Exemplo (funcional?)

```
public class Matematica {  
  
    static Integer calculaMaximo(Integer[] numeros) {  
        Integer max = Integer.MIN_VALUE;  
        for (Integer numero: numeros) {  
            if (numero > max) {  
                max = numero;  
            }  
        }  
        return max;  
    }  
  
    public static void main(String[] args) {  
  
        Integer[] numeros = new Integer[] { 3, 7, 9, 2 };  
  
        Function<Integer[], Integer> funcaoMax = Matematica::calculaMaximo;  
  
        Integer maximo = funcaoMax.apply(numeros);  
    }  
}
```


Dúvida

```
public class Matematica {  
  
    static Integer calculaMaximo(Integer[] numeros) {  
        Integer max = Integer.MIN_VALUE;  
        for (Integer numero: numeros) {  
            if (numero > max) {  
                max = numero;  
            }  
        }  
        return max;  
    }  
  
    public static void main(String[] args) {  
  
        Integer[] numeros = new Integer[] { 3, 7, 9, 2 };  
  
        Function<Integer[], Integer> funcaoMax = Matematica::calculaMaximo;  
        Integer maximo = funcaoMax.apply(numeros);  
    }  
}
```

O que é que ganhamos com isto??

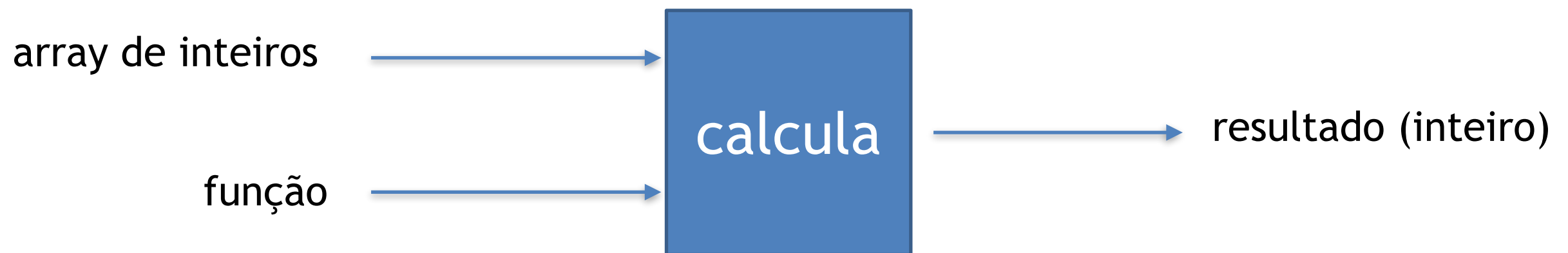
Problema funcional

Além do calculaMax posso também definir o calculaMin, calculaMedia, etc.

```
public class Matematica {  
    static Integer calculaMinimo(Integer[] numeros) {  
        ...  
    }  
  
    static Integer calculaMaximo(Integer[] numeros) {  
        ...  
    }  
  
    static Integer calculaMedia(Integer[] numeros) {  
        ...  
    }  
  
    // etc  
}
```

Problema funcional

E posso criar uma função *calcula* genérica, que aplica uma função a um array de números



```
Integer calcula(Integer[] numeros,  
                Function<Integer[],Integer> funcaoCalculadora) {  
    ...  
}
```

Exemplo (funcional!)

```
static Integer calcula(Integer[] numeros,  
                        Function<Integer[],Integer> funcaoCalculadora) {  
    Integer resultado = funcaoCalculadora.apply(numeros);  
    return resultado;  
}
```

```
public static void main(String[] args) {  
  
    Integer[] numeros = new Integer[] { 3, 7, 9, 2 };  
    Function<Integer[],Integer> funcaoMax = Matematica::calculaMaximo;  
    Function<Integer[],Integer> funcaoMin = Matematica::calculaMinimo;  
  
    Integer maximo = calcula(numeros, funcaoMax);  
    Integer minimo = calcula(numeros, funcaoMin);  
    Integer media = calcula(numeros, Matematica::calculaMedia);  
  
}
```

Exemplo (funcional!)

Só olhando para aqui, não sei o que isto faz...

```
static Integer calcula(Integer[] numeros,
                        Function<Integer[],Integer> funcaoCalculadora) {
    Integer resultado = funcaoCalculadora.apply(numeros);
    return resultado;
}
```

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

```
    Integer[] numeros = new Integer[] { 3, 7, 9, 2 };
    Function<Integer[],Integer> funcaoMax = Matematica::calculaMaximo;
    Function<Integer[],Integer> funcaoMin = Matematica::calculaMinimo;
```

```
    Integer maximo = calcula(numeros, funcaoMax);
    Integer minimo = calcula(numeros, funcaoMin);
    Integer media = calcula(numeros, Matematica::calculaMedia);
```

```
}
```

É aqui que indico
o que ela vai
realmente fazer

Usando uma analogia com objectos, é como se a função calcula fosse abstracta e eu tivesse que indicar aquilo que ela tem realmente de fazer, passando-lhe uma função

Exercício

```
public class Coisa {  
  
    static Integer f1(Integer n) {  
        if (n >= 0) {  
            return 2;  
        } else {  
            return -2;  
        }  
    }  
  
    static Integer f2(Integer n) { return n*3; }  
  
    static Integer f3(Integer n) { return n-3; }  
  
    static Integer f4(Integer i, Function<Integer,Integer> f) {  
        if (i > -5) {  
            return f.apply(i);  
        } else {  
            return i;  
        }  
    }  
  
    public static void main(String[] args) {  
        System.out.println(f4(-7, Coisa::f1));  
        System.out.println(f4(7, Coisa::f3));  
        System.out.println(f1(f4(6, Coisa::f2)));  
        System.out.println(f4(f4(8, Coisa::f1),Coisa::f2));  
        System.out.println(f4(f4(f4(6, Coisa::f1),Coisa::f3),Coisa::f1));  
    }  
}
```

Qual o output deste programa?
Enviar por teams para p4997

Resolução

```
public class Coisa {  
  
    static Integer f1(Integer n) {  
        if (n >= 0) {  
            return 2;  
        } else {  
            return -2;  
        }  
    }  
  
    static Integer f2(Integer n) { return n*3; }  
  
    static Integer f3(Integer n) { return n-3; }  
  
    static Integer f4(Integer i, Function<Integer,Integer> f) {  
        if (i > -5) {  
            return f.apply(i);  
        } else {  
            return i;  
        }  
    }  
  
    public static void main(String[] args) {  
        System.out.println(f4(-7, Coisa::f1));  
        System.out.println(f4(7, Coisa::f3));  
        System.out.println(f1(f4(6, Coisa::f2)));  
        System.out.println(f4(f4(8, Coisa::f1),Coisa::f2));  
        System.out.println(f4(f4(f4(6, Coisa::f1),Coisa::f3),Coisa::f1));  
    }  
}
```

Output:

-7
4
2
6
-2

Exemplo 2

Vamos criar uma função *transformaArray* genérica, que aplica uma função a um array de números, retornando outro array de números “transformado”



```
Integer[] transformaArray(Integer[] numeros,  
                          Function<Integer,Integer> funcaoTransformadora) {  
    ...  
}
```


Função *transformaArray*

```
static Integer[] transformaArray(Integer[] numeros,  
                                Function<Integer,Integer> funcaoTransformadora) {  
  
    Integer[] resultado = new Integer[numeros.length];  
    for (int i = 0; i < resultado.length; i++) {  
        resultado[i] = funcaoTransformadora.apply(numeros[i]);  
    }  
    return resultado;  
}
```



Exemplo (dobro)



```
static Integer dobro(Integer numero) {  
    return numero * 2;  
}
```

Exemplo (negativoPassaAZero)



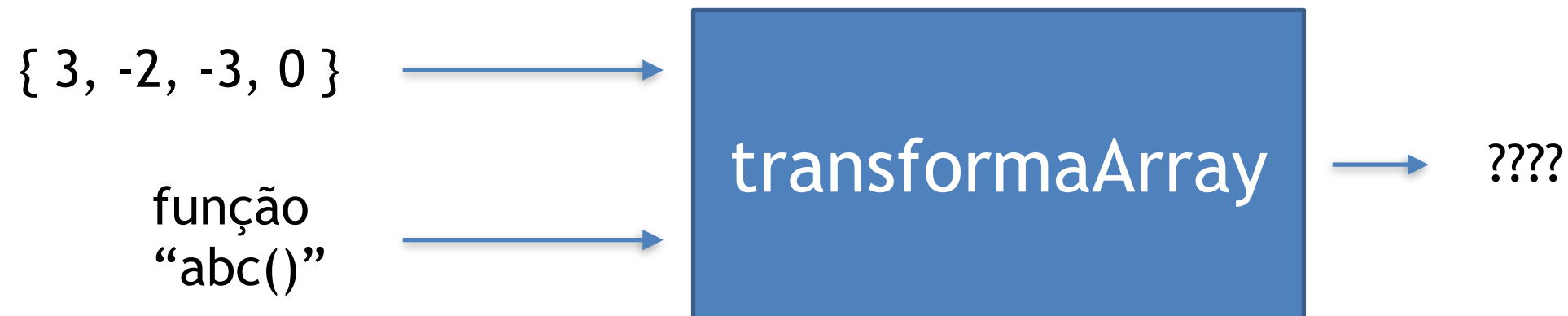
```
static Integer negativoPassaAZero(Integer numero) {  
    if (numero < 0) {  
        return 0;  
    } else {  
        return numero;  
    }  
}
```

Exemplo 2 completo

```
public class Transformacoes {  
  
    private static Integer dobro(Integer numero) {  
        return numero * 2;  
    }  
  
    private static Integer negativoPassaAZero(Integer numero) {  
        if (numero < 0) {  
            return 0;  
        } else {  
            return numero;  
        }  
    }  
  
    private static Integer[] transformaArray(Integer[] numeros,  
                                             Function<Integer,Integer> funcao) {  
  
        Integer[] resultado = new Integer[numeros.length];  
        for (int i = 0; i < resultado.length; i++) {  
            resultado[i] = funcao.apply(numeros[i]);  
        }  
        return resultado;  
    }  
  
    public static void main(String[] args) {  
        Integer[] numeros = new Integer[] { 3, -7, 9, -2 };  
        Integer[] resultado1 = transformaArray(numeros, Transformacoes::dobro);  
        Integer[] resultado2 = transformaArray(numeros, Transformacoes::negativoPassaAZero);  
    }  
}
```

Exercício

Enviar por teams para p4997



```
static Integer abc(Integer numero) {  
    if (numero < 0) {  
        return numero * 2;  
    } else {  
        return numero * 3;  
    }  
}
```

Resolução



```
static Integer abc(Integer numero) {  
    if (numero < 0) {  
        return numero * 2;  
    } else {  
        return numero * 3;  
    }  
}
```

Problema

```
public class Transformacoes {
```

```
    static Integer dobro(Integer numero) {  
        return numero * 2;  
    }
```

```
    static Integer negativoPassaAZero(Integer numero) {  
        ...  
    }
```

```
    static Integer[] transformaArray(Integer[] numeros,  
                                     Function<Integer,Integer> funcao) {  
        ...  
    }
```

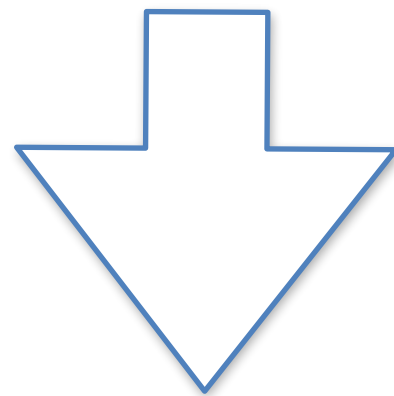
```
    public static void main(String[] args) {  
        Integer[] numeros = new Integer[] { 3, -7, 9, -2 };  
        Integer[] resultado1 = transformaArray(numeros, Transformacoes::dobro);  
    }  
}
```

É chato ter que declarar
uma função para fazer um
cálculo tão simples!

Lambdas

Quando as funções são simples (tipicamente, uma linha de código), podemos usar um mecanismo chamado **lambda**

```
static Integer dobro(Integer numero) {  
    return numero * 2;  
}  
  
...  
Function<Integer,Integer> funcao = Transformacoes::dobro;
```



```
Function<Integer,Integer> funcao = (numero) -> numero * 2;
```


Lambdas

Um lambda tem sempre duas partes separadas pela seta (->)

parâmetros -> corpo

- Os parâmetros são apenas o nome das parâmetros, separados por vírgula, dentro de parêntesis (pode-se omitir o tipo)
Ex: `(x, y) -> x + y;` *// este lambda tem dois parâmetros: x e y*
- Normalmente o corpo é uma expressão que é retornada. Nesse caso, a palavra reservada “return” pode ser omitida
`(x, y) -> x + y;` é equivalente a
`(x, y) -> return x + y;`
- Pode ser indicado tipo dos parâmetros, mas normalmente não é necessário
Ex: `(Integer x, Integer y) -> x + y;`

Exemplo 2 com lambdas

```
public class Transformacoes {  
  
    static Integer[] transformaArray(Integer[] numeros,  
                                    Function<Integer,Integer> funcao) {  
  
        Integer[] resultado = new Integer[numeros.length];  
        for (int i = 0; i < resultado.length; i++) {  
            resultado[i] = funcao.apply(numeros[i]);  
        }  
        return resultado;  
    }  
  
    public static void main(String[] args) {  
        Integer[] numeros = new Integer[] { 3, -7, 9, -2 };  
        Integer[] resultado1 = transformaArray(numeros, (numero) -> numero * 2;);  
    }  
}
```

Deixei de precisar de declarar a função “dobro”

Defina lambdas para as seguintes funções:

1. `int identidade(int x)`
 `(x) -> x`

2. `boolean maior(int x, int y)`

3. `int triplo(int x)`

4. `int compara(int x, int y)`
 // retorna positivo se $x > y$, 0 se forem iguais e negativo se $x < y$

5. `boolean eNegativo(int x)`

6. `String acrescenta(String texto, String sufixo)`

7. `boolean temMaisDoQue(String texto, int numLetras)`

Exercício com lambdas

Enviar por teams para p4997

Defina lambdas para as seguintes funções:

Resolução

1. `int identidade(int x)`
`(x) -> x`

2. `boolean maior(int x, int y)`
`(x, y) -> x > y`

3. `int triplo(int x)`
`(x) -> x * 3`

4. `int compara(int x, int y)`
`// retorna positivo se x > y, 0 se forem iguais e negativo se x < y`
`(x, y) -> x - y`

5. `boolean eNegativo(int x)`
`(x) -> x < 0`

6. `String acrescenta(String texto, String sufixo)`
`(texto, sufixo) -> texto + sufixo`

7. `boolean temMaisDoQue(String texto, int numLetras)`
`(texto, numLetras) -> texto.length > numLetras`