AfroDev

Java Básico - Aula 1 - Parte 2

Operadores - Definição

- Servem para fazer alguma coisa com as variáveis declaradas
- Se estiverem na mesma linha, tem uma ordem de execução
- Parentesis pode ser usado para forçar uma determinada ordem de execução
- Vários tipos. Tipos mais comuns, como por exemplo: + e -
- Tipos mais raros, como por exemplo: >>>

Operadores - Precedência

Operator Precedence

Operators	Precedence
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Operadores - Dicas gerais

- Erro comum: confundir “=” (atribuição) com “==” (igualdade)
  - a = b (a recebe b)
  - a == b (a é igual a b?)
- A soma “+” quando envolve uma String, vira concatenação na precedência
  - int a = 1, b = 2;
  - String s = “bla”
  - a + b + s = “3bla”
- & e && são coisas diferentes. O & avalia toda uma expressão, mesmo que logicamente não seja necessário. Já o && possui curto circuito
  - a & b & c & d -> avalia toda a expressão
  - false && a && b && c && d -> avalia false && a; se for false, não avalia o resto

Operadores - Dicas gerais

- x**++** (pós) e **--**y (pré)
  - x**++** -> tem o valor incrementado em 1 APÓS a execução da linha
  - **--**y -> tem o valor decrementado em 1 ANTES da execução da linha
- x**+=**2; (adicionar 2 ao valor já existente em x. Se x tinha 10, passou a ser 12)
- Ternário é “simples”
  - a ==b **?** true **:** false
- O operador % é utilizado para se saber o resto da divisão entre dois números. Exemplo: 4 % 2 == 0 (4 dividido por 2 tem resto 0)

Controle de fluxo

Geralmente, o código é executado de cima para baixo, na ordem em que aparece no fonte.

Controle de fluxo muda essa lógica, provendo a execução CONDICIONAL de determinados blocos de código.

Tipos de controle fluxo:

- decision-making (**if-then, if-then-else**, switch)
- looping statements (for, while, do-while)
- branching statements (break, continue, return)

```
void verificaColestorelAlto() {

  //precisa retornar um boolean

  if (indice > 200){

      // the "then" clause: decrease current speed

      System.out.println(“Colesterol alto detectado”);

   }

}
```

```
void verificaColestorelAlto() {

   //precisa retornar um boolean

   if (indice > 200){

      // the "then" clause: decrease current speed

      System.out.println(“Colesterol alto detectado”);

   } else {

      System.out.println(“Fique tranquilo”);

   }

}
```

Controle de fluxo

Geralmente, o código é executado de cima para baixo, na ordem em que aparece no fonte.

Controle de fluxo muda essa lógica, provendo a execução CONDICIONAL de determinados blocos de código.

Tipos de controle fluxo:

- decision-making (if-then, if-then-else, **switch**)
- looping statements (for, while, do-while)
- branching statements (break, continue, return)

```
switch (mes) {
case 1:
case 2:
case 3:
case 4:
case 5:
case 6:
    System.out.println("Primeiro semestre");
    break;
case 7:
case 8:
case 9:
case 10:
case 11:
case 12:
    System.out.println("Segundo semestre");
    break;
default:
    System.out.println("Mes invalido!");
    break;
}
```

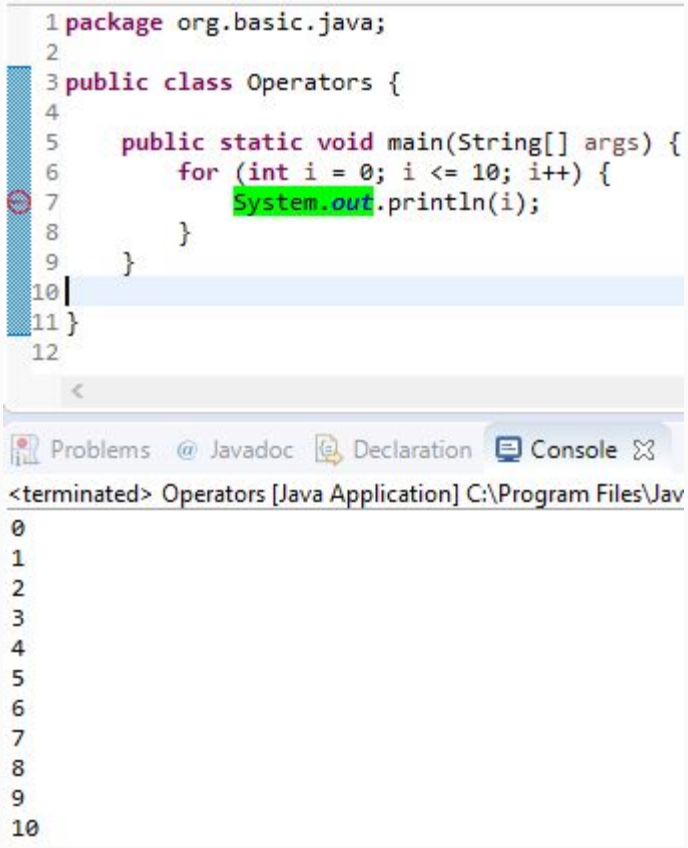
Controle de fluxo

Geralmente, o código é executado de cima para baixo, na ordem em que aparece no fonte.

Controle de fluxo muda essa lógica, provendo a execução CONDICIONAL de determinados blocos de código.

Tipos de controle fluxo:

- decision-making (if-then, if-then-else, switch)
- looping statements (**for**, while, do-while)
- branching statements (break, continue, return)



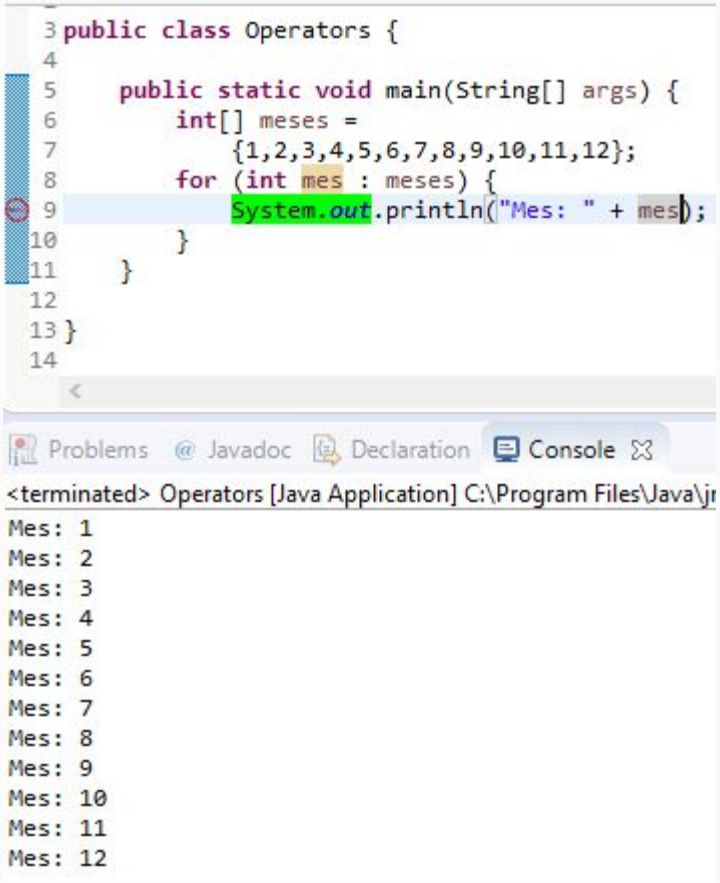
Controle de fluxo

Geralmente, o código é executado de cima para baixo, na ordem em que aparece no fonte.

Controle de fluxo muda essa lógica, provendo a execução CONDICIONAL de determinados blocos de código.

Tipos de controle fluxo:

- decision-making (if-then, if-then-else, switch)
- looping statements (**for**, while, do-while)
- branching statements (break, continue, return)



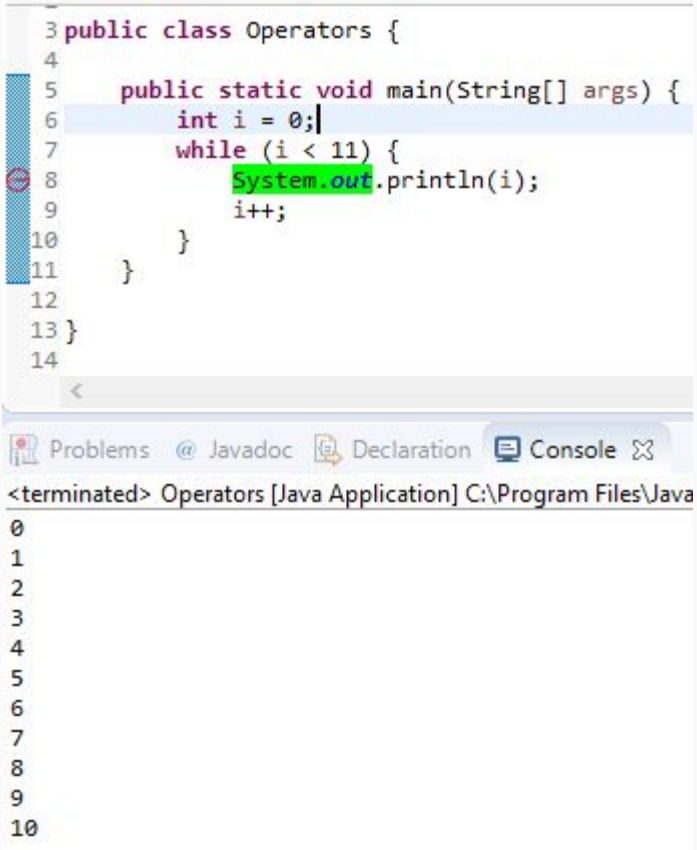
Controle de fluxo

Geralmente, o código é executado de cima para baixo, na ordem em que aparece no fonte.

Controle de fluxo muda essa lógica, provendo a execução CONDICIONAL de determinados blocos de código.

Tipos de controle fluxo:

- decision-making (if-then, if-then-else, switch)
- looping statements (for, **while**, do-while)
- branching statements (break, continue, return)



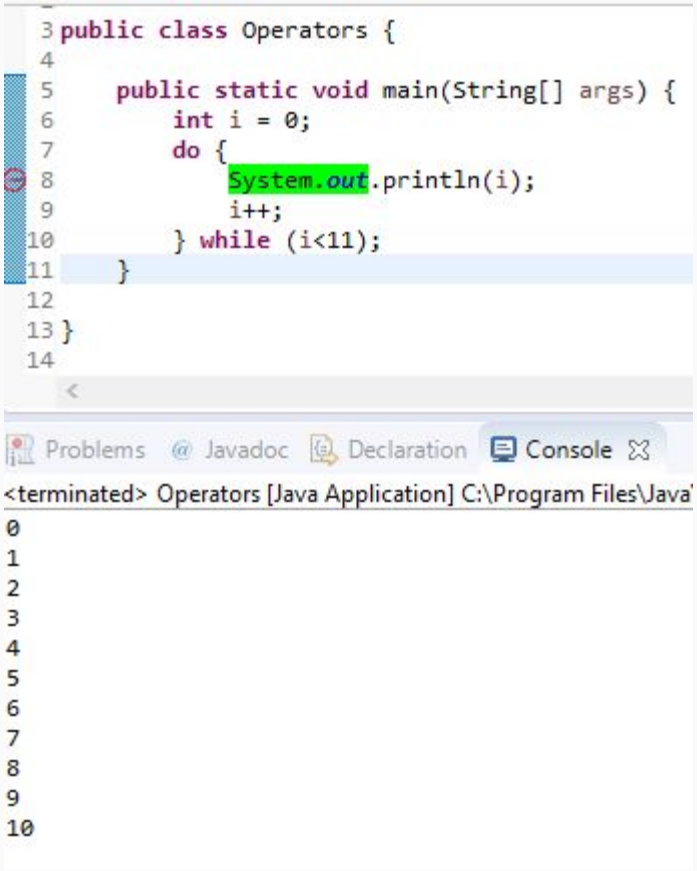
Controle de fluxo

Geralmente, o código é executado de cima para baixo, na ordem em que aparece no fonte.

Controle de fluxo muda essa lógica, provendo a execução CONDICIONAL de determinados blocos de código.

Tipos de controle fluxo:

- decision-making (if-then, if-then-else, switch)
- looping statements (for, while, **do-while**)
- branching statements (break, continue, return)



Controle de fluxo

Geralmente, o código é executado de cima para baixo, na ordem em que aparece no fonte.

Controle de fluxo muda essa lógica, provendo a execução CONDICIONAL de determinados blocos de código.

Tipos de controle fluxo:

- decision-making (if-then, if-then-else, switch)
- looping statements (for, while, do-while)
- branching statements (**break**, continue, return)

```
5    public static void main(String[] args) {
6
7        int loopCount = 0;
8
9        for (int i = 0; i <= 10; i++) {
10           if (i>5) {
11               loopCount++;
12               System.out.println("loopCount parcial: " + loopCount);
13               break;
14           }
15           System.out.println("Indice: " + i);
16       }
17
18       System.out.println("loopCount final: " + loopCount);
19
20   }
21
```

Problems | Javadoc | Declaration | Console

```
<terminated> Operators [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (25 de
Indice: 0
Indice: 1
Indice: 2
Indice: 3
Indice: 4
Indice: 5
loopCount parcial: 1
loopCount final: 1
```

Controle de fluxo

Geralmente, o código é executado de cima para baixo, na ordem em que aparece no fonte.

Controle de fluxo muda essa lógica, provendo a execução CONDICIONAL de determinados blocos de código.

Tipos de controle fluxo:

- decision-making (if-then, if-then-else, switch)
- looping statements (for, while, do-while)
- branching statements (break, **continue**, return)

```
5   public static void main(String[] args) {
6
7       int loopCount = 0;
8
9       for (int i = 0; i <= 10; i++) {
10          if (i>5) {
11              loopCount++;
12              System.out.println("loopCount parcial: " + loopCount);
13              continue;
14          }
15          System.out.println("Indice: " + i);
16      }
17
18      System.out.println("loopCount final: " + loopCount);
19
20  }
21
```

Problems | Javadoc | Declaration | Console

<terminated> Operators [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (25 de

```
Indice: 0
Indice: 1
Indice: 2
Indice: 3
Indice: 4
Indice: 5
loopCount parcial: 1
loopCount parcial: 2
loopCount parcial: 3
loopCount parcial: 4
loopCount parcial: 5
loopCount final: 5
```

Controle de fluxo

Geralmente, o código é executado de cima para baixo, na ordem em que aparece no fonte.

Controle de fluxo muda essa lógica, provendo a execução CONDICIONAL de determinados blocos de código.

Tipos de controle fluxo:

- decision-making (if-then, if-then-else, switch)
- looping statements (for, while, do-while)
- branching statements (break, continue, **return**)

```
5   public static void main(String[] args) {
6
7       int loopCount = 0;
8
9       for (int i = 0; i <= 10; i++) {
10          if (i>5) {
11              loopCount++;
12              System.out.println("loopCount parcial: " + loopCount);
13              return;
14          }
15          System.out.println("Indice: " + i);
16      }
17
18      System.out.println("loopCount final: " + loopCount);
19
20  }
21
```

Problems @ Javadoc Declaration Console

```
<terminated> Operators [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (25 de
Indice: 0
Indice: 1
Indice: 2
Indice: 3
Indice: 4
Indice: 5
loopCount parcial: 1
```

Controles de Fluxo

- Dicas (fique atento):
  - Critério de parada
  - Acessar elemento inicial, do meio e final
  - Não se esqueça do “for-each”

Arrays

- É um vetor de **objetos de mesmo tipo** e **tamanho fixo**.
- int numeros[10]. Vetor de 10 posições de inteiros
- numeros[5] = 4. Acesso a 5ª posição do vetor e atribui o número 4
- int numeros = {1,2,3} //inicializa um array de int de 3 posições
- String brinquedos = {“carro”,”bola”} //inicializa um array de Strings de 2 posições

Objetos

- Coisa material (física) ou mental (virtual) que eu consigo pensar ou sentir
- Coisas…
- Tudo a nossa redor
- Facilita a codificação
- Facilita o processo de modelagem de sistemas (carrinho de compras, produtos, viagens, etc)
- Paradigma amplamente utilizado
- Têm atributos e comportamentos

Objetos - Exemplo Bicicleta

- Altura
- Marchas
- RPM
- Velocidade
- Altura
- Cor
- Modelo
- Fabricante

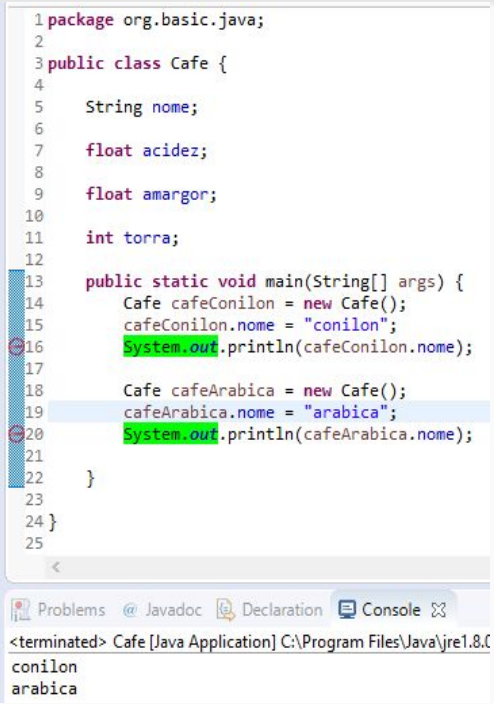
- Frear
- Mudar marcha
- Virar esquerar
- Virar direita
- Pedalar

Classes

- São as “plantas” (esquema, modelo, abstração) de objetos reais
- No mundo real existem vários objeto e cada um com sua particularidade, mas existe um “tipo” que os agrega.
- Exemplo: conilon e arábica são tipos de café. Existem muitos outros, cada um com suas características únicas, mas o tipo continua sendo café. Portanto, ele pode ser uma classe no seu programa JAVA.
- Os atributos dos objetos são representados pelas variáveis de uma classe
- Os comportamentos dos objetos são representados pelos métodos de uma classe

Instâncias da Classe de Objetos

- São objetos “reais” e únicos
- Nascem através da palavra reservada “new”
- Objetos criados ficam na heap da VM
- São limpos quando não possuem mais referência



```
1 package org.basic.java;
2
3 public class Cafe {
4
5     String nome;
6
7     float acidez;
8
9     float amargor;
10
11    int torra;
12
13    public static void main(String[] args) {
14        Cafe cafeConilon = new Cafe();
15        cafeConilon.nome = "conilon";
16        System.out.println(cafeConilon.nome);
17
18        Cafe cafeArabica = new Cafe();
19        cafeArabica.nome = "arabica";
20        System.out.println(cafeArabica.nome);
21
22    }
23
24 }
25
```

Problems @ Javadoc Declaration Console

<terminated> Cafe [Java Application] C:\Program Files\Java\jre1.8.0

```
conilon
arabica
```

Compilando nosso primeiro programa

- Instalar o JDK: https://www.oracle.com/br/java/technologies/javase/javase-jdk8-downloads.html
- Definir JAVA_HOME
- java -version para testar
- Salvar o código anterior num arquivo chamado Cafe.java
- Compilar o programa através do comando: javac Cafe.java
- Executar o programa através do comando: java Cafe

Revisão

- Aprendemos os operadores JAVA e sua ordem de execução (quando mais de um). Eles são utilizados para “fazer algo” com as variáveis declaradas
- O fluxo tradicional de execução de cima para baixo pode ser alterados através dos controle de fluxo como *ifs*, *fors*, e *whiles*
- Fizemos um resgate no conceito de objetos, classes e instâncias
- Criamos um objeto através do comando *new*
- Compilamos e geramos o bytecode do nosso programa *Cafe* utilizando *javac* e o executamos utilizando o comando *java*

Desafio

- Escreva um programa JAVA, sem utilizar IDE, que faça o seguinte
- Crie um array de números com quantos números quiser, mas mais do que 3 elementos
- Faça um loop no array e a cada 3 número imprima o número encontrado