Introdução à linguagem Java: Instruções de controlo de fluxo e ciclos

UA.DETI.POO

Controlo de fluxo num programa

- A ordem de execução das instruções de um programa é normalmente linear
  - uma declaração após a outra, em sequência
- Algumas instruções permitem alterar esta ordem, decidindo:
  - se deve ou não executar uma declaração particular
  - executar uma declaração repetidamente, repetidamente
- Essas decisões são baseadas em expressões booleanas (ou condições)
  - que são avaliadas como verdadeiras ou falsas

Expressões booleanas

- Expressões booleanas retornam **true** ou **false**.
- As expressões booleanas usam operadores relacionais, de igualdade, e lógicos (AND, OR, NOT)

```
==      equal to        // Atenção!! x == y é diferente de x = y
!=      not equal to
<       less than
>       greater than
<=      less than or equal to
>=      greater than or equal to
!       NOT
&&      AND
||      OR
```

- Exemplos

```
x >= 10
(y < z) && (z > t)
```

Tabelas de verdade

- A álgebra booleana é baseada em tabelas de verdade.
- Considerando A e B, por ex: ((y < z) && (z > t))
  - Ambos têm que ser verdadeiros para a expressão **A && B** ser verdadeira.
  - Basta um ser verdadeiro para a expressão **A || B** ser verdadeira.

a	*!a*
true	false
false	true

a	*b*	*a && b*	*a || b*
false	false	false	false
false	true	false	true
true	false	false	true
true	true	true	true

Controle de Fluxo e ciclos similares ao Python

If : Java vs Python

```java
int x = 4;

if (x % 2 == 0) {
    System.out.println("Even");
}
```

```java
Scanner sc = new Scanner(System.in);
int x = sc.nextInt();

if (x > 0) {
    System.out.println("Positive");
} else {
    System.out.println("Negative");
}
```


```python
x = 4

if x % 2 == 0:
    print("Even")
```

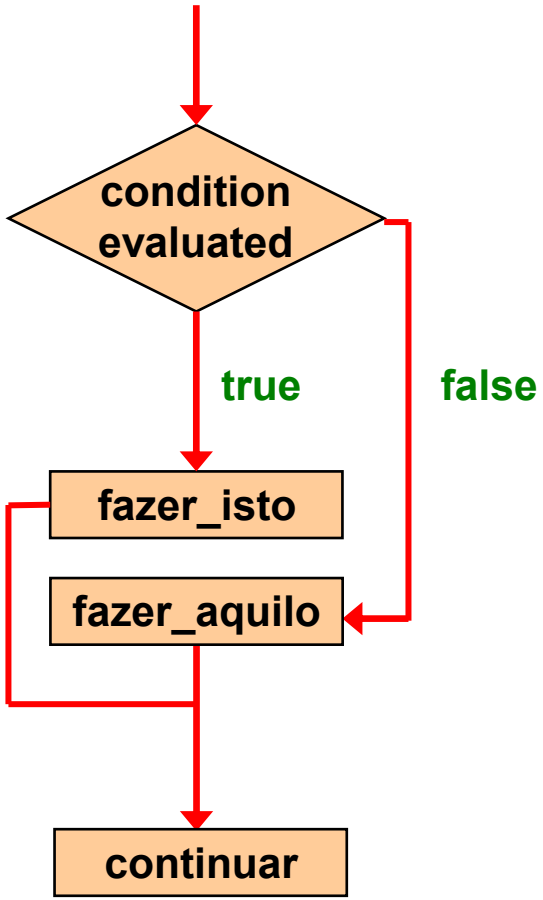

```python
x = input()

if x > 0:
    print("Positive")
else:
    print("Negative")
```

Instruções condicionais

- Em Java existem dois tipos de instruções de decisão/seleção:
  - **if**
  - **switch**

- A instrução if tem o seguinte formato:

```
if (expressãoBooleana)
   // fazer_isto;
else  // opcional
   // fazer_aquilo;
```



Instrução de decisão if

- Podemos encadear várias instruções if:

```
if (condição1)
        bloco1;
else if (condição2)
        bloco2;
else
        bloco3;
```

Se um bloco incluir mais que uma instrução, o bloco deve ser delimitado por { .. }.

- Exemplo

```
if (faltas <= 3)
        System.out.println("Pode ir ao exame teórico.");
else if (!regime.equals("T"))
        System.out.println("Reprovado por faltas.");
else {
        System.out.println("Aluno trabalhador sem a/c.");
        System.out.println("Deve fazer exame prático.");
}
```

Instrução de decisão if – o que não fazer

- As condições devolvem sempre um boleano.

Não fazer:

```
if (condição1)
        return true;
else
        return false;
```

```
return condição1;
```

Novidade: switch

- A instrução switch executa um de entre vários caminhos (case), consoante o resultado de uma expressão

```
switch (expressão) {
        case valor1:
                bloco1;
                break;
        case valor2:
                bloco2;
                break;
        //...
        default:
                blocoFinal;
}
```

O resultado da expressão é pesquisado na lista de alternativas existentes em cada case, pela ordem com que são especificados.

Se a pesquisa for bem sucedida, o bloco de código correspondente é executado.

A execução só termina quando aparecer um break ou se chegar ao fim do bloco de código de todo o switch. Se não for colocado o break no case correspondente, serão executadas todas as opções seguintes até que apareça um break ou seja atingido fim do switch.

Se a pesquisa não for bem sucedida e se o default existir, o bloco de código correspondente (blocoFinal) é executado.

Novidades: switch

```
int day = 2;

switch (day) {
    case 1: System.out.println("Mon"); break;
    case 2: System.out.println("Tue"); break;
    default: System.out.println("Invalid");
}
```


```
Scanner sc = new Scanner(System.in);
int category = sc.nextInt();

switch (category) {
 case 10:
   System.out.println ("a perfect score. Well done.");
   break;
 case 9:
   System.out.println ("well above average. Great.");
   break;
 case 8:
   System.out.println ("above average. Nice job.");
   break;
 case 7:
   System.out.println ("average.");
   break;
 case 6:
   System.out.println ("below average.");
   System.out.println ("See the instructor.");
   break;
 default:
   System.out.println ("not passing.");
}
```

Operador ternário

- O operador ternário (**?:**) é também conhecido como operador condicional.

```
result = testCondition ? valueIfTrue : valueIfFalse
```

  - Avalia uma expressão (1º operando) e, caso seja true, o resultado é igual ao 2º operando, caso contrário o resultado é igual ao 3º operando.

```
char code = 'F';
boolean capitalLetter = (code >= 'A') && (code <= 'Z');
System.out.println(capitalLetter ? "sim" : "não");

minVal = (a < b) ? a : b;
```

Ciclos

Ciclo while

- O ciclo **while** executa enquanto a condição do ciclo esteja verdadeira.
  - A condição é avaliada antes de cada iteração do ciclo.

```
while (condição)
        bloco_a_executar;
```

  - Exemplo:

```
Scanner sc = new Scanner(System.in);
int nota = -1;
while ( (nota > 20) || (nota < 0) ) {
        System.out.println("Insira a nota do aluno.");
        nota = sc.nextInt();
}
sc.close();
```

Ciclo for

- O ciclo **for** é mais geral pois suporta todas as situações de execução repetida.

```
for (inicialização; condição; atualização)
        bloco_a_executar;
```

1. Antes da 1ª iteração, faz a **inicialização** (só uma vez)
2. Depois realiza o teste da **condição**.
   Se for *true* executa o bloco, se for *false* termina
3. No fim de cada iteração, executa a parte de **atualização** e retoma no ponto 2 anterior.

Ciclo for – Relação com o while

- Na verdade, o ciclo **for** surgiu como uma forma mais compacta de se escrever um ciclo while, sendo os dois perfeitamente equivalentes:

```
inicialização;
while (condição)
        bloco_a_executar;
        atualização;
```

```
for (inicialização; condição; atualização)
        bloco_a_executar;
```

For e while : Java vs Python

```java
for (int i = 1; i <= 3; i++) {
    System.out.println(i);
}
```

```python
for i in range(1, 4):
    print(i)
```


```java
int i = 0;

while (i < 3) {
    System.out.println(i);
    i++;
}
```

```python
i = 0

while i < 3:
    print(i)
    i += 1
```

Ciclo for (sintaxe foreach)

- O ciclo for, quando usado com vetores, pode ter uma forma mais sucinta (foreach).

```
Scanner sc = new Scanner(System.in);
double[] a = new double[5];
for (int i = 0; i < a.length; i++)
        a[i] = sc.nextDouble();

for (double el : a)
        System.out.println(el);

sc.close();
```

Ciclo for (sintaxe foreach)

```java
Scanner sc = new Scanner(System.in);
int v = sc.nextInt();
Int n=12

List<Integer> twoTimes = new ArrayList<>();
for (int i = 1; i <= n; i++) {
   twoTimes.add(v * i);
}

int i = 1;
for (int value : twoTimes) {
   System.out.println(v + “ x " + i + " = " + value);
   i++;
}
```


```python
v = int(input())   # equivalent to Scanner + nextInt()
n = 12

two_times = []
for i in range(1, n + 1):
    two_times.append(v * i)

i = 1
for value in two_times:   # for-each over the list
    print(f"{v} x {i} = {value}")
    i += 1
```

Ciclo do while

- O ciclo **do...while** executa sempre uma vez e só depois verifica se é necessário repetir.
  - A condição é avaliada no fim de cada iteração do ciclo.

```
do
        bloco_a_executar;
while (condição);
```

  - Exemplo:

```
Scanner sc = new Scanner(System.in);
int nota;
do {
        System.out.println("Insira a nota do aluno.");
        nota = sc.nextInt();
} while ( (nota > 20) || (nota < 0) );
sc.close();
```

Novidades: do… while

```java
int i = 0;

do {
    System.out.println(i);
    i++;
} while (i < 3);
```

```python
i = 0

while True:
    print(i)
    i += 1
    if i >= 3:
        break
```

Equivalente em python

break e continue: análogo ao Python

- Podemos terminar a execução de um bloco de instruções com duas instruções especiais:
  - **break** e **continue**.

- A instrução **break** permite a saída imediata do bloco de código que está a ser executado.
  - É usada extensivamente no switch mas também pode ser usada nos ciclos – apesar de ser *visto como má prática, indicia uma má condição de paragem*.
- A instrução **continue** permite saltar a execução da iteração corrente, passando a execução para a iteração seguinte *(i.e., não termina o ciclo)*.

Instruções break e continue

- Exemplo:

```
public class Testes {
        public static void main(String[] args) {
                int[] numbers = { 10, 20, 30, 40, 50 };
                for (int x : numbers) {
                        if (x == 30) {
                                break;
                        }
                        System.out.println(x);
                }
        }
}
```

```
10
20
```

Suporte para operações matemáticas

java.lang.Math

- A classe *Math* contém métodos estáticos para executar operações numéricas básicas
  - funções exponenciais, logarítmicas, de raiz quadrada e trigonométricas.

Modifier and Type	Method and Description
static double	**abs**(double a) Returns the absolute value of a double value.
static float	**abs**(float a) Returns the absolute value of a float value.
static int	**abs**(int a) Returns the absolute value of an int value.
static long	**abs**(long a) Returns the absolute value of a long value.
static double	**acos**(double a) Returns the arc cosine of a value; the returned angle is in the range 0.0 through *pi*.

https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html

java.lang.Math

- Funções gerais
  - Math.abs()
  - Math.ceil()
  - Math.floor()
  - Math.min()
  - Math.max()
  - Math.round()
  - Math.random()
- Funções exponenciais, logarítmicas
  - Math.exp()
  - Math.log()
  - Math.log10()
  - Math.pow()
  - Math.sqrt()
- Funções trigonométricas
  - Math.PI
  - Math.sin()
  - Math.cos()
  - Math.tan()
  - Math.asin()
  - Math.acos()
  - Math.atan()
  - Math.atan2()
  - Math.sinh()
  - Math.cosh()
  - Math.tanh()
  - Math.toDegrees()
  - Math.toRadians()

Exemplos

```
public class Testes {

	public static void main(String[] args) {
		double x = 2.75;
		System.out.println("número aleatório = " + Math.random());
		System.out.println("x = " + x);
		System.out.println("sin = " + Math.sin(x));
		System.out.println("cos = " + Math.cos(x));
		System.out.println("sqrt = " + Math.sqrt(x));
		System.out.println("round = " + Math.round(x));
		System.out.println("ceil = " + Math.ceil(x));
	}

}
```

```
número aleatório = 0.7283141219266507
x = 2.75
sin = 0.38166099205233167
cos = -0.9243023786324636
sqrt = 1.6583123951777
round = 3
ceil = 3.0
```

(próxima aula: Strings, Formatação, RegEx)