Introdução ao Desenvolvimento Orientado a Objetos

Conceitos Básicos da Linguagem Java - Parte II

Comandos Condicionais

Tomando decisões (*if* e *if* ternário)

Comandos Condicionais

- Em Java podemos utilizar comandos condicionais como o comando *if*, que possui a seguinte sintaxe:

```
if (idade >= 18) {
    System.out.println("Você é um adulto.");
}
```

Comandos Condicionais

- Em Java podemos utilizar comandos condicionais como o comando *if*, que possui a seguinte sintaxe:

```
if (idade >= 18) {
    System.out.println("Você é um adulto.");
}
```

- Este comando permite executar um bloco de código em acordo com uma condição.

Comandos Condicionais

- Em Java podemos utilizar comandos condicionais como o comando *if*, que possui a seguinte sintaxe:

```
if (idade >= 18) {
    System.out.println("Você é um adulto.");
}
```

- A condição é uma expressão que precisa retornar um valor booleano.
  - Pode utilizar:
    - Operadores relacionais binários, tais como: >, >=, <, <=
    - Operadores relacionais unários, tais como: !
    - Operadores lógicos tais como: && (AND) e || (OR)

Comandos Condicionais

- Em Java podemos utilizar comandos condicionais como o comando *if*, que possui a seguinte sintaxe:

```
if (idade >= 18) {
    System.out.println("Você é um adulto.");
}
```

- Abertura e fechamento do bloco de código. Todo código que estiver neste bloco será executado quando a condição for verdadeira.

Comandos Condicionais

- O comando *if-else* possui a seguinte sintaxe:

```
if (idade >= 18) {
   System.out.println("Você é um adulto.");
} else {
   System.out.println("Você NÃO é um adulto.");
}
```

Comandos Condicionais

- O comando *if-else* possui a seguinte sintaxe:

```
if (idade >= 18) {
    System.out.println("Você é um adulto.");
} else {
    System.out.println("Você NÃO é um adulto.");
}
```

- O comando if-else permite a execução do bloco do if quando a avaliação da condição resulta em *true*.
- Caso a condição resulte em *false*, o bloco de código do else será executado.

Exercícios

7) Faça um programa que leia a idade de um nadador e imprima a categoria do mesmo.

Categoria Pré-Mirim: Crianças de 07 e 08 anos
Categoria Mirim: Crianças de 09 e 10 anos
Categoria Petiz: Crianças de 11 e 12 anos
Categoria Infantil: Crianças e adolescentes de 13 e 14 anos
Categoria Juvenil: Adolescentes de 15 e 16 anos
Categoria Júnior: Adolescentes e adultos com 17, 18 e 19 anos
Categoria Sênior: Adultos com 20 anos acima

Exercícios

8) Faça um programa que leia o salário de um funcionário e imprima o valor do Imposto de Renda que o mesmo deve pagar.

Base de Cálculo (R$)	Alíquota (%)	Dedução do IR (R$)
Até R$ 2.259,20	zero	zero
De R$ 2.259,21 até R$ 2.826,65	7,50%	R$ 169,44
De R$ 2.826,66 até R$ 3.751,05	15%	R$ 381,44
De R$ 3.751,06 até R$ 4.664,68	22,50%	R$ 662,77
Acima de R$ 4.664,68	27,50%	R$ 896,00

Comandos Condicionais

- O if ternário possui a seguinte sintaxe:

```
int idade = 20;
String msg = (idade >= 18 ? "Adulto" : "Não adulto");
System.out.println(msg);
```

- Experimente alterar a idade atribuída na primeira linha para 17.

Comandos Condicionais

- O if ternário possui a seguinte sintaxe:

```
int idade = 20;
String msg = (idade >= 18 ? "Adulto" : "Não adulto");
System.out.println(msg);
```

- O if-ternário retorna um de dois valores possíveis.

Comandos Condicionais

- O if ternário possui a seguinte sintaxe:

```
int idade = 20;
String msg = (idade >= 18 ? "Adulto" : "Não adulto");
System.out.println(msg);
```

- Condição: precisa resultar em um valor booleano.

Comandos Condicionais

- O if ternário possui a seguinte sintaxe:

```
int idade = 20;
String msg = (idade >= 18 ? "Adulto" : "Não adulto");
System.out.println(msg);
```

- A expressão que fica após a interrogação (?) é retornado se a condição resultar em true.

Comandos Condicionais

- O if ternário possui a seguinte sintaxe:

```
int idade = 20;
String msg = (idade >= 18 ? "Adulto" : "Não adulto");
System.out.println(msg);
```

- A expressão que fica após os dois pontos (:) é retornado se a condição resultar em false.

Comandos de Repetição

Repetindo trecho de código
(Loops *while*, *for* e *for-each*)

Comandos de Repetição

- O comando *while* repete um bloco de código enquanto a condição for verdadeira:

```
int i;

i = 0;

while (i < 5) {
    System.out.printf("Linha %d\n", i + 1);
    i++; //i = i + 1;
}
```

Comandos de Repetição

- O comando *while* repete um bloco de código enquanto a condição for verdadeira:

```
int i;

i = 0;

while (i < 5) {
    System.out.printf("Linha %d\n", i + 1);
    i++;
}
```

- Bloco de código a ser repetido.
  - O início do bloco de código é delimitado por “{“
  - O final do bloco de código é delimitado por “}”

Comandos de Repetição

- O comando *while* repete um bloco de código enquanto a condição for verdadeira:

```
int i;

i = 0;

while (i < 5) {
    System.out.printf("Linha %d\n", i + 1);
    i++;
}
```

- A condição é uma expressão que precisa retornar um valor booleano.
- A variável utilizada nesta condição é denominada “variável de controle”, pois ela controla se o laço continuará sendo executado.

Comandos de Repetição

- O comando *while* repete um bloco de código enquanto a condição for verdadeira:

```
int i;

i = 0;

while (i < 5) {
    System.out.printf("Linha %d\n", i + 1);
    i++;
}
```

- É necessário garantir que a condição poderá resultar em false em algum momento, caso contrário teremos um loop infinito.
- Nesta linha atualizamos o valor da variável de controle.

Comandos de Repetição

- O comando *while* repete um bloco de código enquanto a condição for verdadeira:

```
int i;

i = 0;

while (i < 5) {
    System.out.printf("Linha %d\n", i + 1);
    i++;
}
```

- A variável de controle precisa ser inicializada antes que o seu valor seja checado na condição.

Comandos de Repetição

- O comando *while* repete um bloco de código enquanto a condição for verdadeira:

```
int i;

        1
i = 0;

                2
while (i < 5) {
    System.out.printf("Lin
    i++;
}       3
```

- Em ordem, temos:
  1) Inicialização da variável de controle
  2) Condição do laço
  3) Atualização da variável de controle, que geralmente é feita dentro do bloco de código do laço.

Exercícios

9) Faça um programa que leia 10 idades e imprima a média das idades informadas pelo usuário. Neste exercício é obrigatório o uso do comando de repetição *while* para controlar o número de idades a serem lidas.

Exercícios

10) Faça um programa que leia idades informadas pelo usuário enquanto a idade informada for maior ou igual a zero. Ao término da leitura (assim que o usuário informar uma idade negativa), imprima a média das idades informadas pelo usuário. Neste exercício é obrigatório o uso do comando de repetição *while* para controlar o número de idades a serem lidas.

Comandos de Repetição

- O comando *for* permite a inicialização, a condição e a atualização seja feita em uma mesma linha:

```
for (int i = 0; i < 5; i++) {
   System.out.printf("Linha %d\n", i + 1);
}
```

Comandos de Repetição

- O comando *for* permite a inicialização, a condição e a atualização seja feita em uma mesma linha:

```
for (int i = 0; i < 5; i++) {
    System.out.printf("Linha %d\n", i + 1);
}
```

- Bloco de código a ser repetido.
  - O início do bloco de código é delimitado por “{“
  - O final do bloco de código é delimitado por “}”

Comandos de Repetição

- O comando *for* permite a inicialização, a condição e a atualização seja feita em uma mesma linha:

```
for (int i = 0; i < 5; i++) {
    System.out.printf("Linha %d\n", i + 1);
}
```

- Declaração e inicialização da variável de controle. Também é possível apenas inicializar uma variável declarada anteriormente.
  - É possível inicializar mais de uma variável, separando por vírgula. Exemplo:

```
for (i = 0, j = 0; i < 5; i++)
```

Comandos de Repetição

- O comando *for* permite a inicialização, a condição e a atualização seja feita em uma mesma linha:

```
for (int i = 0; i < 5; i++) {
    System.out.printf("Linha %d\n", i + 1);
}
```

- Condição. Enquanto esta condição for verdadeira, o bloco de código continuará sendo executado.

Comandos de Repetição

- O comando *for* permite a inicialização, a condição e a atualização seja feita em uma mesma linha:

```
for (int i = 0; i < 5; i++) {
    System.out.printf("Linha %d\n", i + 1);
}
```

- Atualização da variável de controle.
  - É possível atualizar mais de uma variável, separando por vírgula. Exemplo:

```
for (int i = 0; i < 5; i++, j++
```

Exercícios

11) Faça um programa que leia 10 números inteiros e, para cada número lido, imprima “par” se o mesmo for par ou imprima “ímpar” se o número lido for ímpar. Neste exercício é obrigatório o uso do comando de repetição *for* para controlar o número de inteiros a serem lidos.

Exercícios

12) Faça um programa que leia um inteiro *N*. Se *N* for menor ou igual a zero, o programa deve imprimir “Quantidade de números inválida!”. Se *N* for maior que zero, o programa deve ler *N* números reais. Ao término da leitura dos números o programa deve imprimir qual foi o maior número lido. Neste exercício é obrigatório o uso do comando de repetição *for* para controlar a quantidade de números reais a serem lidos.

Listas

Criando listas de tamanho variável com a classe ArrayList

ArrayList

- A classe ArrayList permite a criação de arrays com tamanho variável.

```
import java.util.ArrayList;

public class Main {
   public static void main(String[] args) {
        ArrayList<Integer> inteiros  = new ArrayList<Integer>();
        ArrayList<Double> reais  = new ArrayList<Double>();
        ArrayList<String> textos  = new ArrayList<String>();
   }
}
```

ArrayList

- A classe ArrayList permite a criação de arrays com tamanho variável.

```
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ArrayList<Integer> inteiros  = new ArrayList<Integer>();
        ArrayList<Double> reais  = new ArrayList<Double>();
        ArrayList<String> textos  = new ArrayList<String>();
    }
}
```

- Importação da classe ArrayList que fica no pacote java.util.

ArrayList

- A classe ArrayList permite a criação de arrays com tamanho variável.

```
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ArrayList<Integer> inteiros  = new ArrayList<Integer>();
        ArrayList<Double> reais  = new ArrayList<Double>();

    }
}
```

- A variável inteiros referencia um objeto do tipo ArrayList<Integer>.
  - Entre <> deve ser informado o tipo de dados dos elementos que serão armazenados no ArrayList.

ArrayList

- A classe ArrayList permite a criação de arrays com tamanho variável.

- **Atenção**
  - Os elementos em um ArrayList precisam ser **objetos**, logo você **não pode informar um tipo primitivo**. Caso seja necessário um ArrayList para um tipo int, por exemplo, informe a classe empacotadora do tipo **int** que é a **Integer**.
  - Outras classes empacotadoras que podem ser de seu interesse:
    - **Boolean** para o tipo **boolean**
    - **Character** para o tipo **char**
    - **Double** para o tipo **double**

ArrayList

- A classe ArrayList permite a criação de arrays com tamanho variável.

```
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ArrayList<Integer> inteiros  = new ArrayList<Integer>();
        ArrayList<Double> reais  = new ArrayList<Double>();
        ArrayList<String> textos  = new ArrayList<String>();
    }
}
```

- ArrayList para armazenar números reais.

ArrayList

- A classe ArrayList permite a criação de arrays com tamanho variável.

```
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        ArrayList<Integer> inteiros  = new ArrayList<Integer>();
        ArrayList<Double> reais  = new ArrayList<Double>();
        ArrayList<String> textos  = new ArrayList<String>();
    }
}
```

- ArrayList para armazenar strings.

ArrayList

- Adicionando elementos:

```
inteiros.add(5);
inteiros.add(2);

textos.add("Texto 1");
textos.add("Texto 2");
textos.add("Outro te
```

- Adiciona o inteiro 5 no ArrayList referenciado pela variável “inteiros”. Em seguida, adiciona o número 2.
  - O método add é uma sub-rotina, logo você pode passar qualquer expressão que resulte em um número inteiro por parâmetro para um ArrayList<Integer>.

ArrayList

- Adicionando elementos:

```
inteiros.add(5);
inteiros.add(2);

textos.add("Texto 1");
textos.add("Texto 2");
textos.add("Outro texto");
```

- Strings adicionadas no ArrayList referenciado pela variável “textos”.

ArrayList

- Removendo elementos:

```
ArrayList<Integer> inteiros  = new ArrayList<Integer>();

inteiros.add(5);
inteiros.add(2);
inteiros.add(1);
inteiros.add(4);

System.out.println(inteiros);
inteiros.remove(0);
System.out.println("-----------");
System.out.println(inteiros);
```

- Removendo o elemento que está no índice 0 do ArrayList. Os índices variam de 0 até tamanho - 1.

ArrayList

- Obtendo um elemento do ArrayList

```
ArrayList<Integer> inteiros  = new ArrayList<Integer>();

inteiros.add(5);
inteiros.add(2);
inteiros.add(1);
inteiros.add(4);

System.out.println(inteiros.get(0));
inteiros.set(0, 10);
System.out.println(inteiros);
```

- É possível obter um elemento que está no ArrayList utilizando para isso o método get.
  - Este método recebe por parâmetro o índice do elemento a ser lido

ArrayList

- Alterando um elemento do ArrayList

```
ArrayList<Integer> inteiros  = new ArrayList<Integer>();

inteiros.add(5);
inteiros.add(2);
inteiros.add(1);
inteiros.add(4);

System.out.println(inte      .get(0));
inteiros.set(0, 10);
System.out.println(inteiros);
```

- É possível alterar um elemento que está no ArrayList utilizando para isso o método set.
  - Este método recebe por parâmetro o índice do elemento a ser alterado e o novo valor do elemento.

ArrayList

- Percorrendo os elementos do ArrayList

```
ArrayList<String> textos  = new ArrayList<String>();

textos.add("Texto 1");
textos.add("Texto 2");
textos.add("Outro texto");

for (int i = 0; i < textos.size(); i++) {
   System.out.println((textos.get(i)));
}
```

- É possível percorrer os elementos de um ArrayList com laços *for* ou *while*.

ArrayList

- Percorrendo os elementos do ArrayList

```
ArrayList<String> textos  = new ArrayList<String>();

textos.add("Texto 1");
textos.add("Texto 2");
textos.add("Outro texto");

for (String texto: textos) {
   System.out.println(texto);
}
```

- É possível percorrer os elementos de um ArrayList com laços *for-each*.

Exercícios

13) Faça um programa que leia idades informadas pelo usuário enquanto a idade informada for maior ou igual a zero. Ao término da leitura (assim que o usuário informar uma idade negativa), imprima a média das idades informadas pelo usuário assim como a quantidade de idades acima da média. Neste exercício é obrigatório o uso de um *ArrayList*.

Exercícios

14) Faça um programa que leia 10 números reais em um *ArrayList*. O programa deve criar um segundo *ArrayList* e preenchê-lo com os elementos do primeiro *ArrayList*, só que em ordem inversa.

Fim de material

Dúvidas?