Universidade do Estado de Santa Catarina - UDESC
Centro de Educação Superior do Alto Vale do Itajaí
Departamento de Engenharia de Software
Bacharelado em Engenharia de Software

Fundamentos de Java
25PRO1 – Programação 1

Ibirama
Segunda Fase

Fundamentos de Java
25PRO1 – Programação 1

Por
Fernando dos Santos

7 de março de 2022

Sumário

Avisos

O propósito destas notas de aula é organizar a apresentação do conteúdo referente a fundamentos de programação Java da disciplina 25PRO1 da UDESC Ibirama. O aluno não deve utilizar este documento como única fonte de aprendizagem, e sim buscar material, conteúdo e exercícios nos materiais abaixo e nos livros citados no plano de ensino da disciplina.

Legenda de códigos Java
Nos trechos de código apresentados, os símbolos `< >` são usados para indicar um elemento no código que deve ser substituído conforme orientações no texto ao se escrever o código Java definitivo.

Legenda de exercícios
Os exercícios práticos estão divididos nos seguintes níveis de dificuldade:

★ Fácil
★★ Médio
★★★ Complexo
♣ Desafio

Leituras e Materiais Recomendados

Vídeos

A *playlist Curso de Java para Iniciantes*, do canal *Curso em Video* disponibiliza vídeos os assuntos abordados neste documento. Em especial, recomendo assistir os seguintes vídeos (clique para abrir):

- Curso de Java #02 - Como Funciona o Java
- Curso de Java #04 - Primeiro Programa em Java
- Curso de Java #06 - Tipos Primitivos e Manipulação de Dados
- Curso de Java #07 - Operadores Aritméticos e Classe Math
- Curso de Java #08 - Operadores Lógicos e Relacionais
- Curso de Java #09 - Estruturas Condicionais (Parte 1)
- Curso de Java #10 - Estruturas Condicionais (Parte 2)
- Curso de Java #11 - Estruturas de Repetição (Parte 1)

- Curso de Java #12 - Estruturas de Repetição (Parte 2)
- Curso de Java #13 - Estruturas de Repetição (Parte 3)
- Curso de Java #14 - Vetores
- Curso de Java #15 - Métodos

Livros

- Capítulos 6, 7 e 11 do livro (Santos, 2013).
- Seções 2.4, 2.11, 2.12 e 2.13 do livro (Barnes and Kölling, 2004) ou (Barnes and Kölling, 2009).
- Capítulos 1 e 3 do livro (Sierra and Bates, 2007).
- Capítulos 4, 5, 6 e 7 do livro (Deitel, 2010).

(todos os livros acima estão disponíveis na biblioteca da UDESC Alto Vale)

1 Estrutura, Tipos de Dados e Operadores

1.1 Estrutura de um Programa Java

Todo programa Java **executável** deve seguir a estrutura apresentada no código 1-1.

Código Java 1-1 Estrutura de programa Java - arquivo: `MeuPrograma.java`

```
1 public class MeuPrograma {
2   public static void main(String[] args) {
3     // aqui vai o programa executável
4     System.out.println("Hello World");
5   }
6 }
```

Esta estrutura é imposta pelo Java. A estrutura rígida é imposta pelo Java por ser uma linguagem **fortemente orientada a objetos**. O assunto *orientação a objetos* será visto no próximo tópico da disciplina. Por hora, para construir um programa executável Java será necessário:

- Declarar uma classe pública com o nome do seu programa: `public class` (linha 1). **Atenção**: o nome do arquivo deve ser **exatamente igual** ao da classse (inclusive maiúsculas e minúsculas).
- Declarar um método público estático, com retorno `void`, chamado `main`: `public static void main` (linha 2). É dentro deste método que o seu programa é implementado.
- Observe que o Java utiliza { } (chaves) para delimitar blocos.

1.2 Variáveis e Tipos de Dados

Assim como em várias outras linguagems, em Java você utiliza **variáveis** para armazenar dados relevantes ao seu programa. Em Java toda variável é declarada com um determinado **tipo de dado**.

Os tipos de dados primitivos oferecidos pelo Java são:

1.2.1 Inteiros

`byte`: valores inteiros entre -128 e +127.

`short`: valores inteiros entre -32.768 e +32.767.

`int`: valores inteiros entre -2.147.483.648 e 2.147.483.647.

`long`: valores inteiros entre -9.223.372.036.854.775.808 e 9.223.372.036.854.775.807.

1.2.2 Decimais

`float`: valores reais

`double`: valores reais com maior precisão/capacidade.

1.2.3 Lógico

`boolean`: valores booleanos (true/false).

1.2.4 Caracter

`char`: um único caracter. Exemplos: `'a'`, `'2'` (observe o uso dos apóstrofos; são eles que delimitam os caracteres no código fonte).

1.2.5 Cadeia de caracteres

`String`: armazena uma cadeia de caracteres. Exemplos: `"Programação 1"`, `"Silvio Santos"` (observe o uso de aspas).

1.2.6 Exemplos de Variáveis

O código 1-2 apresenta exemplos de **declaração** de variáveis e **atribuição de valores**. Observe que todas as variáveis são declaradas dentro do método `main`.

Código Java 1-2 Exemplos de Variáveis

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

    // Declaração de variáveis
    String nome;
    char sexo;
    int idade;

    // Atribuição de valores à variáveis
    nome = "Silvio Santos";
    sexo = 'M';
    idade = 90;

    // Declaração + atribuição de valor (tudo junto)
    String nomeConjuge = "";
    double patrimonio = 99999999.5993; // ponto para decimal
```

```
17
18      // Atribuição de variável em variável
19      patrimonio = idade; // double recebe int
20      short menorIdade = 18;
21      idade = menorIdade;  // int recebe short
22    }
23  }
```

Em Java, **variáveis numéricas** podem armazenar valores contidos em outras **variáveis numéricas**, dependendo de sua capacidade (e consequentemente, do espaço reservado na memória para a variável). A regra é: variáveis de maior capacidade podem armazenar valores contidos em variáveis de menor capacidade; o contrário não é possível. Por exemplo, uma variável `int`, tem espaço suficiente para armazenar o valor de uma variável de menor capacidade, no caso `short` ou `byte`. Já uma variável `double` pode armazenar o valor de uma variável `float` ou qualquer variável inteira. Exemplos de variáveis armazenando outras variáveis estão entre as linhas 19 e 21 do código 1-2.

1.3 Operadores aritméticos

Operadores aritméticos atuam sobre dois operandos e resultam em valores numéricos. Os operadores aritméticos disponíveis no Java são:

`+` adição.

`-` subtração.

`*` multiplicação.

`/` divisão.

`%` módulo (resto da divisão).

1.4 Operadores relacionais

Os operadores relacionais comparam dois operandos ou duas expressões e resultam em valores lógicos (**VERDADEIRO** ou **FALSO**). Os operadores relacionais disponíveis no Java são:

`==` igual a.

`!=` diferença.

`>` maior que.

`<` menor que.

`>=` maior ou igual a.

`<=` menor ou igual a.

`.equals` para verificar igualdade entre objetos `String`.

Exemplos:

`2 + 5 > 4` resulta **VERDADEIRO**.

`3 != 3` resulta **FALSO**.

`nome.equals("Silvio")` retorna **VERDADEIRO** caso o conteúdo da variável `nome` seja `"Silvio"` (inclusive comparando maiúsculas e minúsculas), e **FALSO** caso não seja.

É um erro de programação usar o operador `==` para comparar valores/variáveis `String`, por exemplo `nome == "Silvio"` . O Java não irá acusar erro na compilação, e o resultado da comparação não levará em consideração o **conteúdo** das strings.

1.5 Operadores lógicos

Os operadores lógicos atuam sobre expressões e também resultam em valores lógicos **VERDADEIRO** ou **FALSO**. Os operadores relacionais disponíveis no Java são:

`&&` E (conjunção).

`||` OU (disjunção).

`!` NÃO (negação).

Exemplos:

`3 > 2 && 5 < 3` resulta **FALSO**.

`3 > 2 || 5 < 3` resulta **VERDADEIRO**.

`! (5 < 3)` resulta **VERDADEIRO**.

Ao elaborar expressões complexas que misturam operadores aritméticos, relacionais e lógicos, esteja atento a ordem de prioridade que estes operadores são avaliados: primeiro os **aritméticos**, depois **relacionais**, e por fim os **lógicos**. Assim como na matemática, você pode usar parêntesis na expressão para definir uma ordem específica de avaliação.

1.6 Saída (impressão) de Dados

Para impressão (saída) de valores em modo texto, usaremos o método `System.out.println`. A sintaxe do `System.out.println` é apresentada no código 1-3. Para que o valor de alguma variável seja impresso junto ao texto, basta concatenar a variável usando "`+`", conforme mostrado nas linhas 2 a 4. Se desejar, é possível imprimir apenas o valor da variável (sem concatenar com outro texto), como exemplificado na linha 5. Por fim, para imprimir sem gerar nova linha (ou seja, sem "quebra de linha" ao final do texto) basta usar o método `System.out.print` (sem "`ln`" no final), conforme mostrado na linha 8.

Por padrão, valores decimais são impressos com diversas casas decimais. Para definir a quantidade de casas decimais a serem impressas, deve-se utilizar o método `String.format`, conforme exemplificado na linha 13.

O método `String.format` requer dois argumentos. O primeiro argumento é uma `String` que especifica o formato desejado. No caso da linha 13, o caractere "`%`" indica que a string define uma formatação, e o caractere "`f`" define uma formatação de valor decimal (*floating point*). Por fim, o valor "`.2`" antes do "`f`" indica a quantidade de dígitos a serem mostrados após o ponto (separador de decimal). Altere este valor para mostrar diferentes quantidades de dígitos. No segundo argumento informa-se a variável cujo valor será formatado.

Para conhecer outros formatos, consulte (Alexander, 2019).

Código Java 1-3 Saída de dados em texto (assumir as variáveis do código 1-2)

```
1
2  System.out.println("Nome: " + nome);
3  System.out.println("Sexo: " + sexo);
4  System.out.println("Idade: " + idade);
5  System.out.println(nome);
6
7  // Impressão sem gerar nova linha
8  System.out.print("Nome: +"nome);
9
10
11 // Impressão de decimais, sem e com limites de casas
12 System.out.println("Patrimônio: " + patrimonio);
13 System.out.println("Patrimônio (2 decimais): " +
   ↪  String.format("%.2f", patrimonio));
14
15 // Impressão de texto complexo com várias variáveis
16 System.out.println("O apresentador " + nome + " tem " + idade +
   ↪  " anos e patrimônio de " + String.format("%.2f",
   ↪  patrimonio) + " reais");
```

1.7 Entrada (leitura) de Dados

Para entrada (leitura) de dados, a maneira mais simples é utilizando um **objeto** `Scanner`. O código 1-4 mostra como usar um objeto `Scanner` para ler diferentes tipos de dados. Observe que:

- Na linha 1 é feita a importação da classe que contém a implementação do `Scanner`. Sempre que for utilizar `Scanner` seu código precisa conter, logo no início, a declaração: `import java.util.Scanner`.
- Na linha 6 é criado o objeto `Scanner` que será utilizado para fazer as leituras. Ele é armazenado na variável `scan`, que é do tipo `Scanner`. Apenas após criar o objeto é possível usar o `Scanner`. Caso você tente usar a variável `scan` sem fazer o `new Scanner()` ocorrerá o erro `NullPointerException` ao executar o programa.
- Na linha 7 é executado o método `useDelimiter` no objeto `scan`. Este método configura qual caractere será utilizado pelo objeto `Scanner` como **separador de valores**. Neste exemplo estamos definindo que a separação dos valores será feita pelo caractere `"\n"`, que representa uma **quebra de linha** (tecla *enter*).

Nos laboratórios virtuais de programação do Moodle deve-se **sempre utilizar** o método `useDelimiter` conforme exemplo da linha 7. Caso contrário, os valores não serão corretamente lidos pelo `Scanner`

Código Java 1-4 Entrada de Dados (Leitura em Texto)

```
1  import java.util.Scanner;
2
3  public class MeuPrograma {
4    public static void main(String[] args) {
5      // objeto Scanner para entrada de dados
6      Scanner scan = new Scanner(System.in);
7      scan.useDelimiter("\n"); // para ler Strings corretamente
8
9      System.out.println("Informe String");
10     String valorString = scan.next();
11     System.out.println("Valor String: " + valorString);
12
13     System.out.println("Informe Inteiro");
14     int valorInt = scan.nextInt();
15     System.out.println("Valor Inteiro: "+ valorInt);
16
17     System.out.println("Informe Float");
18     float valorFloat = scan.nextFloat();
19     System.out.println("Valor Float: " + valorFloat);
```

```
20
21      System.out.println("Informe Double");
22      double valorDouble = scan.nextDouble();
23      System.out.println("Valor Double: " + valorDouble);
24
25      System.out.println("Informe booleano");
26      boolean valorBoolean = scan.nextBoolean();
27      System.out.println("Valor Boolean: " + valorBoolean);
28
29      scan.close(); // fechamento do objeto Scanner
30    }
31  }
```

É possível utilizar **qualquer caractere** no método `useDelimiter`. Por exemplo, ao utilizar `scan.useDelimiter(";")`, faremos com que o `Scanner` separe os dados pelos `";"` para então fazer a leitura dos valores. Veja um exemplo de implementação no código 1-5.

Código Java 1-5 Exemplo de scan.useDelimiter

```
1  import java.util.Scanner;
2
3  public class MeuPrograma {
4    public static void main(String[] args) {
5      Scanner scan = new Scanner(System.in);
6      scan.useDelimiter(";"); // para ler valores separados por ;
7      System.out.println("Informe 3 inteiros separados por ';'");
8      int valor1 = scan.nextInt();
9      int valor2 = scan.nextInt();
10     int valor3 = scan.nextInt();
11     System.out.println(valor1);
12     System.out.println(valor2);
13     System.out.println(valor3);
14     scan.close(); // fechamento do objeto Scanner
15   }
16 }
```

Ao executar o código acima, o usuário deve digitar os três valores inteiros de uma só vez, separados por `";"` desta forma:

```
5;8;10;
```

Observe que há um `";"` após o último valor. Isto é necessário para que o objeto `Scanner` consiga identificar onde está o último valor.

1.8 Exercícios de Fixação

Exercício 1-1 Estrutura, Tipos de Dados e Operadores

Considere as seguintes variáveis:

```
int x = 3
int y = 16
float z = 2.5
String nome1 = "Ana"
String nome2 = "Paulo"
```

Calcule o **resultado** das seguintes expressões e especifique o **tipo de dado** resultante.

a) `nome1.equals(nome2)`
b) `nome1.equals("ANA")`
c) `nome2.equals("Paulo")`
d) `x * y + 3`
e) `y / x`
f) `y / z`
g) `y % x`
h) `z < x && z < y`
i) `!(y % x == 0)`

1.9 Exercícios Práticos

Para cada exercício, há um **Laboratório de Programação** no Moodle. Após implementar, submeta seu código para avaliação e atribuição de nota.

Exercício 1-2 ★ Fundamentos: Hello World

Implemente e execute o exemplo *Hello World* apresentado no código 1-1.

Exercício 1-3 ★ Fundamentos: Média 3 notas

Implemente e execute um programa que lê o nome e as três notas de um aluno, calcula, e mostra a média simples deste aluno.

Exemplos de entrada: nome as 3 notas

```
Ana
7,5
8,0
9,0
```

```
Pedro
9,5
10,0
6,0
```

Exemplos de saídas para entradas ao lado

```
Ana
8,2
```

```
Pedro
8,5
```

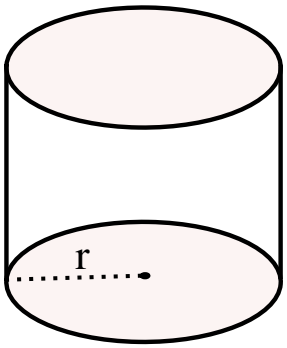
Para que o Moodle VPL possa avaliar seu programa, os únicos comandos `System.out.println` permitidos são os que imprimem as saídas solicitadas. Isto significa que você deve remover os `println's` que imprimem mensagens aos usuários, por exemplo: `System.out.println("Informe Nota 1")`. Caso você mantenha estes `println's`, o corretor do Moodle VPL acusará falha nas saídas do seu programa.

O formato da(s) saída(s) deve ser **exatamente igual** ao solicitado.

Exercício 1-4 ★★ Fundamentos: Volume caixa d'água

Faça um programa Java que determine o volume V de uma caixa d'água cilíndrica, sendo que o raio e a altura devem ser fornecidos(lidos pelo teclado).

$$V = PI * Raio^2 * altura$$



Entradas: raio \n altura

```
1,0
2,0
```

```
0,65
1,20
```

```
0,8
2,5
```

```
0,4
1,0
```

Exemplos de saídas para entradas ao lado

```
6,28
```

```
1,59
```

```
5,03
```

```
0,50
```

Exercício 1-5 ★★ Fundamentos: Salário Funcionário

Escrever um programa Java que lê, em ordem:

1. O nome de um funcionário.
2. O número de horas trabalhadas no mês.
3. O valor que recebe por hora.
4. O número de filhos.

Com estes dados, calcular o salário deste funcionário, sabendo que para cada filho, o funcionário recebe 3% a mais, calculado sobre o salário bruto.

Entradas: dados do funcionário

```
Bill Gates
180
35,29
3
```

```
Juca
200
14,99
5
```

Saídas:

```
Bill Gates: 6923,90
```

```
Juca: 3447,70
```

Exercício 1-6 ★★★ Fundamentos: Troco

Implemente um programa Java que calcula o menor número de cédula que deve ser dado de troco para um pagamento efetuado. O programa deve ler o nome do estabelecimento,

o valor a ser pago e o valor efetivamente pago. Supor que existam as cédulas de 50, 20, 10, 5, 2 e 1 real. Assumir que o valor a ser pago é inteiro. Ou seja, não é necessário se preocupar com centavos (troco em moedas).

Entradas: estabelecimento \n valor a ser pago \n valor efetivamente pago

```
Nardelão
75
100
```

```
Posto Ipiranga
37
50
```

Saídas: quantidade de cédulas "x" valor da cédula (ordenadas da maior para a menor)

```
Nardelão
0x50
1x20
0x10
1x5
0x2
0x1
```

```
Posto Ipiranga
0x50
0x20
1x10
0x5
1x2
1x1
```

2 Estruturas Condicionais

São utilizadas para definir quais comandos serão executados em função de uma **condição**. A condição deve ser uma expressão **lógica**, que retorna verdadeiro ou falso.

2.1 Condicional Simples

É o comando `if`. A sintaxe é mostrada no código 2-1.

Código Java 2-1 Estrutura Condicional Simples

```
1 if ( <condição> ) {
2   // comandos que são executadas se <condição> for VERDADEIRA
3 }
```

Atenção às chaves {}. São elas que delimitam o bloco de comandos associado à estrutura condicional.

2.2 Condicional Composta

É o comando `if` com `else`. A sintaxe é mostrada no código 2-2.

Código Java 2-2 Estrutura Condicional Composta

```
1 if ( <condição> ) {
2   // comandos que são executadas se <condição> for VERDADEIRA
3 }else{
4   // comandos que são executados se <condição> for FALSA
5 }
```

2.3 Condicional Aninhada

Trata-se de várias condicionais simples e/ou compostas juntas. Neste caso, as chaves {} são fundamentais para delimitar qual(is) comando(s) pertencem a qual condicional. Um exemplo é mostrado no código 2-3.

Código Java 2-3 Estrutura Condicional Aninhada

```
if ( <condição1> ) {
  ...
  if ( <condição2> ) {
    ...
    if ( <condição3> ) {
      ...
    }
    ...
  } else {
    ...
  }
  ...
} else {
  ...
  if ( <condição4> ) {
    ...
  }
  ...
}
```

2.4 Condicional Múltipla

Permite testar um valor/variável e especificar quais comandos são executados dependendo do valor. Esta estrutura de seleção permite simplificar os programas. Sem ela, teriamos que escrever vários `if`/`else` aninhados. A sintaxe é mostrada no código 2-4.

Código Java 2-4 Estrutura Condicional Múltipla

```
switch ( <variável ou expressão>) {
  case <valor_A> : {
    // bloco de instruções A
    break;
  }
  case <valor_B> : {
    // bloco de instruções B
    break;
  }
  // outros cases, conforme necessidade
  default: {
    // bloco de instruções default
  }
}
```

A estrutura `switch` é formada por vários comandos `case`, e um comando `default` (opcional). Ao executar a estrutura `switch`, o programa compara a `<variável>` (ou `<expressão>`) com o valor de cada `case`. Quando encontra valores coincidentes, o programa executa o bloco de instruções que está dentro daquele `case`.

A instrução `break` é utilizada para finalizar a estrutura `switch`. Quando o programa encontra um `break`, ele passa a executar a próxima instrução após o `switch`. Sem o `break`, todos os `case` da estrutura `switch` que estivessem abaixo daquele escolhido seriam executados.

Se nenhum `case` coincidir com o valor da `<variável>` (ou `<expressão>`), o `default` (se existir) é executado.

Dentro de cada `case`, e também dentro do `default`, podem ser declaradas vários outros comandos (`if`, `while`, `switch`, etc).

O código 2-5 mostra um exemplo de condicional múltipla que determina qual é o dia da semana correspondente a um número de 1 a 7.

Código Java 2-5 Exemplo de Condicional Múltipla

```
1  int diaNumerico; // variável que contém o dia, de 1 a 7
2  String diaExtenso; // variável para armazenar o extenso
3  switch(diaNumerico){
4    case 1: {
5      diaExtenso = "Segunda-Feira";
6      break;
7    }
8    case 2:{
9      diaExtenso = "Terça-Feira";
10     break;
11   }
12   case 3:{
13     diaExtenso = "Quarta-Feira";
14     break;
15   }
16   case 4:{
17     diaExtenso = "Quinta-Feira";
18     break;
19   }
20   case 5:{
21     diaExtenso = "Sexta-Feira";
22     break;
23   }
24   case 6:{
25     diaExtenso = "Sábado";
26     break;
27   }
28   case 7:{
```

```
29        diaExtenso = "Domingo";
30        break;
31      }
32      default:{
33        diaExtenso = "Inválido";
34        break;
35      }
36    }
37    System.out.println("Dia da semana é "+diaExtenso);
```

A `<variável>` ou `<expressão>` que está dentro do `switch` só pode ter como resultado um valor do tipo `byte`, `int`, `char` ou `String`.

2.5 Exercícios de Fixação

Exercício 2-1 Condicionais aninhados

Considere o trecho de código a seguir, e as condições: `cond1`, `cond2`, e `cond3`.

```
1  if (cond1) {
2    System.out.println("Oi;");
3    System.out.println("Jose;");
4  }
5  if (cond2) {
6    System.out.println("Tudo bem?;");
7    if (cond3) {
8      System.out.println("Maria;");
9      System.out.println("Tchau;");
10   }
11   System.out.println("Ate logo;");
12   System.out.println("Pedro;");
13 }
14 System.out.println("Joao;");
```

Responda o que será impresso **na sequência**, quando:

a) `cond1 = true; cond2 = false; cond3 = false;`
b) `cond1 = true; cond2 = true; cond3 = true;`
c) `cond1 = false; cond2 = true; cond3 = false;`
d) `cond1 = true; cond2 = false; cond3 = true;`

Exercício 2-2 Condicionais aninhados

Considere o trecho de código a seguir, e as condições: `condA`, `condB`, `condC` e `condD`.

```
1  System.out.println("Inicio;");
2  if (condA) {
3    System.out.println("CEAVI;");
4    if (condD) {
5      System.out.println("Oi;");
6    }
7  } else {
8    if (condB) {
9      System.out.println("Prog1;");
10     if (condC) {
11       System.out.println("Ibirama;");
12     } else {
13       System.out.println("SC;");
14     }
15     System.out.println("UDESC;");
16   }
17   System.out.println("Fim;");
18 }
```

Responda o que será impresso **na sequência**, quando:

a) `condA = true; condB = true; condC = false; condD = false;`
b) `condA = false; condB = false; condC = true; condD = true;`
c) `condA = false; condB = true; condC = true; condD = true;`

Exercício 2-3 Condicionais aninhados

Ainda com relação ao trecho de código do exercício anterior (2-2), quais devem ser os valores de `condA`, `condB`, `condC` e `condD` para que a impressão do programa **contenha** os valores a seguir?

a) “Oi;”
b) “SC;”
c) “Prog1;UDESC;”

Exercício 2-4 Condicionais múltiplos

Considere o trecho de código a seguir.

```
1  int mes = scan.nextInt();
2
3  switch (mes) {
4    case 2:{
5      System.out.println("28 dias");
6      break;
7    }
8    case 4:
9    case 6:
10   case 9:
11   case 11: {
12     System.out.println("30 dias");
13     break;
14   }
15   default:{
16     System.out.println("31 dias");
17   }
18 }
```

O que será impresso quando:

a) `mes = 2`
b) `mes = 1`
c) `mes = 4`
d) `mes = 6`
e) `mes = 7`
f) `mes = 12`

2.6 Exercícios Práticos

Para cada exercício, há um **Laboratório de Programação** no Moodle. Após implementar, submeta seu código para avaliação e atribuição de nota.

Exercício 2-5 ★ Condicionais: Empréstimo

Faça um programa Java que lê o nome, o valor do salário de uma pessoa e o valor de um financiamento pretendido. Caso o financiamento seja menor ou igual a 5 vezes o

salário da pessoa, o algoritmo deverá imprimir "Financiamento CONCEDIDO"; se não, ele deverá imprimir "Financiamento NEGADO". Independente de conceder ou não o financiamento, o algoritmo imprimirá depois a frase "Obrigado por nos consultar".

Atenção para maiúsculas/minúsculas nas `Strings` impressas.

Exemplos de entradas:

```
Bento
3658,30
10000,00
```

Exemplos de saídas:

```
Bento
Financiamento CONCEDIDO
Obrigado por nos consultar
```

```
Josué
1500,25
12690,30
```

```
Josué
Financiamento NEGADO
Obrigado por nos consultar
```

Exercício 2-6 ★★ Condicionais: Calculadora Básica

Faça um programa Java que calcula algumas operações matemáticas elementares. O programa lê, em ordem:

1. O operador (`String`), conforme detalhes abaixo.
2. O primeiro termo da operação.
3. O segundo termo da operação, apenas se for uma operação sobre dois termos.

Após ler os dados acima, o programa deve calcular e imprimir o resultado da operação. Os operadores possíveis são:

- `"sum"` calcula a soma dos dois termos.
- `"sub"` calcula a subtração dos primeiro pelo segundo termo.
- `"mult"` calcula a multiplicação dos dois termos.
- `"div"` calcula a divisão do primeiro pelo segundo termo. Se o segundo termo for zero, então não deve realizar o cálculo, apenas imprimir "Divisão por zero".
- `"abs"` esta é uma operação que requer apenas o primeiro termo, e calcula seu valor absoluto. **Obs:** no caso desta operação, o programa **não deve ler o segundo termo**.

É obrigatório o uso de uma estrutura **condicional múltipla** (`switch/case`).

Exemplos de entradas:

```
sum
5
5
```

```
sub
6,8
2,2
```

```
abs
-9,1
```

```
div
13,5
0
```

Saídas para entradas ao lado

```
10,0
```

```
4,6
```

```
9,1
```

```
Divisão por zero
```

Exercício 2-7 ★★★ Condicionais: Equação Segundo Grau

Uma equação do segundo grau tem o seguinte formato:

$$ax^2 + bx + c$$

Escreva um programa Java para resolver equações do segundo grau. O programa deve ler os coeficientes a, b, e c, e resolver a equação conforme instruções abaixo.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

O seu programa deve calcular e imprimir a(s) raiz(es) da equação.

- Se o interior da raiz quadrada é igual a zero, então existe **uma raiz**.
- Se o interior da raiz quadrada é positivo, então existem **duas raízes**.
- Se o interior da raiz quadrada é negativo, então não existe raiz real. O programa deve imprimir "Sem raiz real".

Entradas: a \n b \n c

```
2
5
-3
```

Saídas: a(s) raiz(es). Se houver duas então devem estar na mesma linha e separadas por ponto e vírgula

```
0,50;-3,00
```

```
1
2
1
```

```
-1,00
```

```
Sem raiz real
```

```
2,5
5
5
```

Java oferece diversas operações matemáticas prontas (ex: raiz quadrada, potência, etc). Veja exemplos em `https://www.w3schools.com/java/java_math.asp`

3 Estruturas de Repetição

São utilizadas para repetir a execução de um bloco de comandos até que uma CONDIÇÃO seja satisfeita.

3.1 Repetição com Teste no Início

É o comando `while`. Nesta estrutura, a `<condição>` é testada no início, ou seja, antes de repetir. Os comandos apenas são executados quando a `<condição>` é VERDADEIRA. A sintaxe é mostrada no código 3-1.

Código Java 3-1 Repetição com Teste no Início

```
1   while ( <condição> ) {
2     // comandos que são executados ENQUANTO <condição> for
      ↪  VERDADEIRA
3   }
```

Atenção às chaves {}. São elas que delimitam o bloco de comandos associado à estrutura de repetição.

3.2 Repetição com Teste no Fim

É o comando `do-while`. Nesta estrutura, a `<condição>` é testada no fim, ou seja, os comandos são executados **ao menos uma vez**. Após a primeira execução dos comandos, eles são repetidos enquanto a `<condição>` é VERDADEIRA. A sintaxe é mostrada no código 3-2.

Código Java 3-2 Repetição com Teste no Fim

```
1   do {
2     // comandos que são executados ao menos uma vez e depois
3     // são repetidos ENQUANTO a <condição> for VERDADEIRA
4   } while ( <condição> )
```

3.3 Repetição com Variável de Controle

É o comando `for`. Nesta estrutura, a quantidade de vezes que os comandos são repetidos é gerenciada pela variável de controle. A sintaxe é mostrada no código 3-3.

Código Java 3-3 Repetição com Variável de Controle

```
1 for(<variável de controle>; <condição>; <atualização>){
2   // comandos que serão executados repetidamente
3 }
```

Na estrutura `for`, a `<variável de controle>` é inicializada e no início de cada repetição ela é verificada na `<condição>`. A `<condição>` atua como uma **condição de continuação**: se resultar **verdadeiro**, os comandos são executados; se resultar **falso**, os laço `for` é finalizado. Após a execução do último comando da lista, a `<atualização>` é executada automaticamente. Isto se repete até que a `<condição>` resulte **falso**.

A estrutura de repetição com variável de controle (`for`) é preferencialmente utilizada em situações em que **sabe-se** previamente o número de repetições a serem feitas. Este número de repetições pode ser uma constante ou estar em uma variável.

O código 3-4 apresenta laço `for` para imprimir os valores ímpares entre 1 e 1000. Neste exemplo, temos que:

- `<variável de controle>`: é a declaração e inicialização da variável `int i=1`.
- `<condicao>`: é a expressão relacional `i < 1000`; ou seja, os comandos serão repetidos **enquanto** o valor de `i` for menor que 1000.
- `<atualização>`: é o incremento da variável de controle `i = i + 2`. Caso quisessemos incrementar apenas uma unidade, poderiamos utilizar `i++`. É possível especificar qualquer expressão de atualização (soma, subtração, multiplicação, etc).

Código Java 3-4 Estrutura de Repetição para Imprimir Valores Ímpares

```
1   for(int i=1; i<1000; i=i+2){
2     System.out.println(i);
3   }
```

3.4 Exercícios de Fixação

Exercício 3-1 Repetição com Variável de Controle

Quantas vezes a palavra será impressa ao se executar o código a seguir?

```
1    for(int i = 0; i < 50; i++){
2      System.out.println("Oi");
3    }
```

Exercício 3-2 Repetição com Variável de Controle

Quantas vezes a palavra será impressa ao se executar o código a seguir?

```
1    for(int j = 1; j <= 35; j++){
2      System.out.println("Alo");
3    }
```

Exercício 3-3 Repetição com Variável de Controle

Quantas vezes a palavra será impressa ao se executar o código a seguir?

```
1    for(int k = 5; k >= 0; k--){
2      System.out.println("Tchau");
3    }
```

Exercício 3-4 Repetição com Variável de Controle

Quantas vezes a palavra será impressa ao se executar o código a seguir?

```
1    for(int m = 0; m < 10; m--){
2      System.out.println("Teste");
3    }
```

Exercício 3-5 Repetição com Variável de Controle

Quantas vezes a palavra será impressa ao se executar o código a seguir?

```
1    for(int n = 0; n < 20; n = n + 2){
2      System.out.println("Ibirama");
3    }
```

Exercício 3-6 Repetição com Variável de Controle

Quantas vezes cada palavra será impressa ao se executar o código a seguir?

```
1    for(int i = 0; i < 10; i++){
2      for(int j = 0; j < 10; j++){
3        System.out.println("interno");
4      }
5      System.out.println("externo");
6    }
```

3.5 Exercícios Práticos

Para cada exercício, há um **Laboratório de Programação** no Moodle. Após implementar, submeta seu código para avaliação e atribuição de nota.

Exercício 3-7 ★ Repetição: Fatorial

Faça um programa Java que lê um valor inteiro positivo N, calcula e imprime o valor de $N!$ (fatorial de N).

Entradas: um valor N por linha	Saídas: o valor de $N!$
5	120
0	1
9	362880
20	2432902008176640000

Exercício 3-8 ★★ Repetição: MDC

Faça um programa Java para calcular o MDC (máximo divisor comum) de dois números. O seu programa deve ler dois números inteiros a e b, e em seguida calcular e imprimir $MDC(a, b)$.

Use o algoritmo de Euclides. Consulte (Di Giacomo, 2020?).

Entradas: *a* `\n` *b*

```
32
12
```

Saídas: o valor do MDC

```
4
```

```
1320
25
```

```
5
```

Exercício 3-9 ★★★ Repetição: Dados Demográficos

Faça um programa Java para determinar dados demográficos de uma turma de alunos. O seu programa deve ler os dados coletados durante a entrevista que é feita com diversos alunos da turma. Estes dados são:

- Nome completo.
- Idade.
- Sexo (caractere `M` ou `F`).
- Cidade de nascimento.

A quantidade de alunos (tamanho da amostra) não é conhecida *a priori*. Ou seja, seu programa deve ler os dados de uma quantidade indeterminada de alunos. Para encerrar as entrevistas, será informado `"fim"` como nome do aluno.
Ao encerrar as entrevistas, seu programa deve calcular e imprimir as seguintes informações:

- O nome do(a) aluno(a) com a menor idade dentre todos.
- A quantidade de alunos com idade entre 18 e 20 anos.
- A média de idade da turma.
- O percentual de alunos do sexo feminino.
- A quantidade de alunos que nasceram em Ibirama.

A seguir, exemplos de entradas e saídas.

Entradas: dados dos alunos

```
Huguinho
19
M
Ibirama
Zezinho
17
M
Rio do Sul
Luizinho
21
M
Blumenau
Patricia
18
F
Ibirama
fim
```

Saídas: dados demográficos

```
Menor idade: Zezinho
Quantidade 18-20: 2
Média idade: 18,75
Percentual feminino: 25,00
Nascidos em Ibirama: 2
```

4 Arrays (Vetores)

Um *array* (ou vetor) é uma estrutura de dados **homogênea** capaz de armazenar vários dados **do mesmo tipo**. Estes dados são acessados através de **índices**. Um índice é um valor numérico **começando em 0**.

Para utilizar um *array* é necessário seguir três passos:

1. Declaração do *array*.
2. Criação (instanciação) do *array*.
3. Manipulação (uso) do *array*.

O código 4-1 exemplifica como estes três passos são realizados em Java. O exemplo define um *array* de 5 posições, para armazenar as notas da prova 1 de uma turma de 5 alunos. As subseções a seguir explicam em detalhes cada um dos três passos.

Código Java 4-1 Declaração, criação, e manipulação de array

```
1   // Declaração de um array para guardar notas da prova 1
2   double[] notasProva1;
3
4   // Criação (instanciação) do array com 5 posições
5   notasProva1 = new float[5];
6
7   // Uso do array: armazenando dados via índice
8   notasProva1[0] = 7.5;
9   notasProva1[1] = 10.0;
10  notasProva1[2] = 8.5;
11  notasProva1[3] = 9.9;
12  notasProva1[4] = 8.0;
13
14  // Uso do array: lendo dados via índice
15  double soma1 = notasProva1[0] + notasProva1[1] +
    ↪  notasProva1[2] + notasProva1[3] + notasProva1[4];
16
17  // Uso do array: repetição FOR para acessar dados
18  double soma2 = 0;
19  for( int i = 0; i < 5; i++){
20    soma2 = soma2 + notasProva1[i]; // variável como índice
21  }
```

4.1 Declaração de Arrays

Em Java, *arrays* são declarados usando a **notação de colchetes**. A sintaxe é a mesma da declaração de uma variável. A diferença é que deve-se colocar um **par de colchetes** `[]` ao lado do tipo da variável, como visto na linha 2 do código 4-1.

4.2 Criação (instanciação) de Arrays

Em Java, todo *array* só pode ser utilizado após ser **instanciado**. É na instanciação que o Java reserva um espaço na memória para armazenar os elementos do *array*. Para a criação do *array* é obrigatório definir a **quantidade de elementos** que serão armazenados, ou seja, o tamanho do *array*. Isto é necessário para que o Java saiba quanta memória precisa reservar.

O comando `new` é utilizado para a criação do *array*, como visto na linha 5 do código 4-1. O comando `new` requer que seja informado novamente o **tipo de dados** do *array*, seguido pela quantidade de elementos entre colchetes.

Como seria a representação visual do vetor `notasProva1`?

Uma vez que o *array* é criado, não é possível alterar seu tamanho.

4.3 Manipulação (uso) de Arrays

Os elementos do *array* **sempre** são acessados através de seus **índices**.

Os índices iniciam em **0** e vão até o **(tamanho − 1)** do *array*.

As linhas 8 a 12 do código 4-1 mostram como fazer atribuições de valores às posições do *array*. O índice é utilizado para definir qual posição do *array* irá receber cada valor de nota. Observe que a primeira posição tem índice 0.

O índice também deve ser utilizado para leitura dos dados armazenados no *array*. Isto é mostrado na linha 15 do código 4-1.

Estruturas de repetição `for` são frequentemente utilizadas quando é necessário atribuir/ler todas as posições do *array*. Um exemplo é mostrado na linha 19 do código 4-1. Observe que a **variável de controle** é utilizada como índice do *array*, logo, ela inicia em 0 e a **condição de continuação** é *enquanto for estritamente menor* que o tamanho do *array*.

4.4 Dimensões de um Array

Arrays podem ter 1, 2, 3, ... até 255 dimensões. Para criar uma nova dimensão, basta escrever um novo par de colchetes tanto na declaração quanto na manipulação do *array*.

O código 4-2 apresenta um exemplo de *array* bidimensional (com duas dimensões). Um *array* bidimensional é frequentemente chamado de **matriz**. No caso do exemplo, as linhas da matriz `notas` representam os alunos, e as colunas são as os valores das notas de cada avaliação.

Código Java 4-2 Exemplo de *array* bidimensional (matriz)

```
1  // Declaração de array bidimensional
2  float[][]  notas;
3
4  // Inicialização: 10 linhas e 5 colunas
5  notas = new float[10][5];
6
7  // Uso do array
8  // É necessário informar um índice para cada dimensão
9  notas[3][2] = 7.5;
```

A figura 4.1 mostra como seria a visualização do *array* bidimensional.

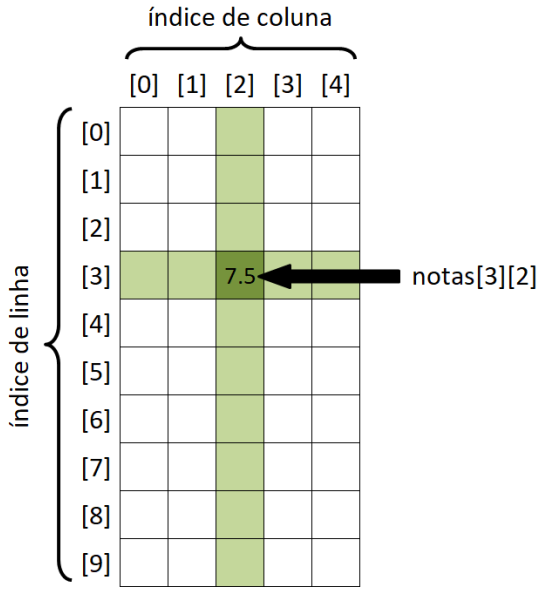


Figura 4.1: Visualização de um Array Bidimensional

4.5 Exercícios de Fixação

Exercício 4-1 Arrays

Deseja-se usar um *array* de 3 dimensões, denominado `temperaturas`, para armazenar a temperatura do ambiente em cada **hora**, **minuto**, e **segundo** de um dia. Pede-se:

a) Declare o *array* em Java.
b) Crie o *array* em Java.
c) Escreva um exemplo de uso (atribuição e leitura de uma posição).
d) Desenhe a representação visual do *array*.

4.6 Exercícios Práticos

Para cada exercício, há um **Laboratório de Programação** no Moodle. Após implementar, submeta seu código para avaliação e atribuição de nota.

Exercício 4-2 ★ Arrays: Unidimensional

Faça um programa Java que lê um vetor de 15 valores inteiros positivos. Em seguida, seu programa deve calcular e imprimir as seguintes informações, em ordem:

- A média dos valores.
- A quantidade valores que são >= a média.
- O menor valor e sua posição. Em caso de empate, considerar o de menor índice.

Entradas: os 15 valores em uma linha, separados por `";"`

```
3;9;4;5;8;0;1;8;5;1;3;2;6;4;7;
```

```
25;33;64;8;100;99;42;71;29;84;37;15;66;8;11;
```

Exemplos de saídas

```
Média: 4,40
Qtd. >= média: 7
Menor valor: 0
Posição menor: 5
```

```
Média: 46,13
Qtd. >= média: 6
Menor valor: 8
Posição menor: 3
```

Para que o `Scanner` leia os valores separados por `";"`, basta trocar o seu delimitador: `scan.useDelimiter(";")`.

Exercício 4-3 ★★ Arrays: Matriz

Faça um programa Java que lê uma matriz M(5,5) de inteiros, calcula e imprime as somas:

a) da **quarta linha** de M.
b) da **segunda coluna** de M.
c) da diagonal principal.
d) da diagonal secundária.
e) de todos os elementos da matriz.

Entradas: valores de M separados por `";"`

```
1;2;3;4;5;
6;7;8;9;0;
1;2;3;4;5;
6;7;8;7;0;
1;2;3;4;5;
```

Saídas: na ordem que foram solicitadas

```
28
20
23
25
103
```

Para que o `Scanner` leia adequadamente as diversas linhas, após ler o último valor de uma linha é necessário executar o comando `scan.nextLine()` para que o `Scanner` pule para a próxima linha.

Exercício 4-4 ★★★ Arrays: Multiplicação de Matrizes

Faça um programa Java que lê duas matrizes numéricas A e B, cujas dimensões (linhas e colunas) são informadas para cada matriz. Em seguida, o programa deve multiplicar as duas matrizes e armazenar o resultado em uma matriz C. Ao final, seu programa deve imprimir a matriz C, ou uma mensagem "Impossível multiplicar" caso não seja possível multiplicar $A \times B$. Duas matrizes A e B podem ser multiplicadas se a quantidade de colunas de A for igual a quantidade de linhas de B.
Exemplo de multiplicação de matrizes:

$$\begin{bmatrix} 1 & 0 & 2 \\ -1 & 3 & 1 \end{bmatrix} \times \begin{bmatrix} 3 & 1 \\ 2 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} (1\times 3 + 0\times 2 + 2\times 1) & (1\times 1 + 0\times 1 + 2\times 2) \\ (-1\times 3 + 3\times 2 + 1\times 1) & (-1\times 1 + 3\times 1 + 1\times 0) \end{bmatrix} = \begin{bmatrix} 5 & 1 \\ 4 & 2 \end{bmatrix}$$

A dimensão da matriz resultante terá a quantidade de linhas de A e a quantidade de colunas de B

Formato da entrada:

- <qtd. linhas de A>
- <qtd. colunas de A>
- <elementos de A>
- <qtd. linhas de B>
- <qtd. colunas de B>
- <elementos de B>

```
2
3
1;0;2;
-1;3;1;
3
2
3;1;
2;1;
1;0;
```

Saída é a matriz C

```
5;1;
4;2;
```

Para ler a quantidade de linhas e colunas da matriz A, o seu objeto `Scanner scan` deve estar configurado com `scan.useDelimiter("\n ")`.
Já para ler os elementos da matriz A, o seu objeto `Scanner scan` deve estar configurado com `scan.useDelimiter(";")`
Você pode trocar o delimitador ao longo do seu programa, quantas vezes for necessário.
Lembre-se que quando for utilizado o delimitador `";"` é necessário executar o comando `scan.nextLine()` após ler o último valor de cada linha da matriz.

5 Métodos

Um programa é uma **rotina** que resolve um problema.

Um método é uma **sub-rotina** que resolve um sub-problema. Em algumas linguagens (ex. Python) métodos são chamados de **funções**.

Métodos são úteis para **simplificar** e **organizar** programas, pois permitem *quebrar* um programa grande em partes (sub-rotinas) menores, que são executadas conforme a necessidade. Além disso, métodos permitem evitar a **duplicação de código**: ao invés de se escrever um trecho de código diversas vezes, escreve-se um método, que é chamado diversas vezes.

As principais características dos métodos são:

1. Eles reduzem o tamanho do algoritmo;
2. Facilitam a compreensão e visualização do algoritmo;
3. Podem ser chamados em qualquer ponto após sua declaração;
4. Podem retornar algum valor, ou não retornar nada.

Observação. Ao utilizarmos orientação a objetos, métodos terão um propósito adicional: representar comportamentos dos objetos. Por hora, usaremos métodos como meio de codificar sub-rotinas.

5.1 Implementação de Métodos

A estrutura geral para implementação de qualquer método é apresentada no código 5-1, nas linhas 3 e 5. Observe que métodos devem ficar **dentro da classe** e fora de qualquer outro método.

Código Java 5-1 Implementação de método: assinatura e corpo

```
1  public class MeuPrograma{
2    // assinatura do método
3    static <tipo de retorno> <nome>(<argumentos>){
4      // aqui vai a implementação do corpo do método
5    }
6    public static void main (String [] args) {
7      // chamada do método
8      <nome>(<argumentos>);
9    }
10 }
```

A linha 3 é denominada de **assinatura do método**. Esta assinatura é que diferencia um método de outros. A implementação do método devem ficar dentro de um par de chaves { }, sendo denominada de **corpo do método**.

A seguir são detalhados os elementos da assinatura de um método.

5.1.1 `<tipo de retorno>`

Em Java, a assinatura de um método **sempre** deve definir qual é o seu tipo de retorno, ou seja, qual o tipo de dado que será retornado pelo método. O tipo de retorno pode ser:

a) Qualquer **tipo de dado primitivo** visto na seção 1.2.
b) Um objeto. Veremos isto no assunto de orientação a objetos.
c) `void`, quando o método não retornará qualquer valor.

Nos casos (a) e (b), o comando `return` deve, **obrigatoriamente**, ser utilizado no corpo do método para finalizar o método e retornar um valor.

5.1.2 `<nome>`

É o nome do método. Segue o mesmo padrão de nomes das variáveis.

5.1.3 `<argumentos>`

São os valores requeridos pelo método para que ele consiga realizar a sua operação.

Argumentos são declarados como variáveis e possuem o mesmo comportamento que elas.

5.1.4 Exemplos de Métodos

O código 5-2 apresenta exemplos de métodos.

O método `somar` (linha 3) calcula e retorna a soma de dois valores inteiros recebidos como argumentos `v1` e `v2`.

O método `potencia` (linha 8) recebe dois argumentos, `base` e `expoente`, e calcula e retorna a potência $base^{expoente}$.

O método `parOuImpar` (linha 16) verifica se o valor recebido como argumento é par ou impar. Este método possui retorno `void`, ou seja, ele não é necessário existir um comando `return` em seu corpo.

Por fim, o método `metodoSemParametro` (linha 16) mostra como é a declaração de um método que não recebe argumentos. Note que os parêntesis `()` são obrigatórios. Por ser `void`, o método não retorna valor algum.

Código Java 5-2 Exemplos de métodos

```
1   public class MeuPrograma{
2
3     static int somar (int v1, int v2){
4       int soma = v1 + v2;
5       return soma;
6     }
7
8     static double potencia(double base, int expoente){
9       double resultado = 1;
10      for (int i = 1; i <= expoente; i++){
11        resultado = resultado * base;
12      }
13      return resultado;
14    }
15
16    static void parOuImpar(int valor){
17      if (valor %2 == 0){
18        System.out.println("Eh par");
19      }else{
20        System.out.println("Eh par");
21      }
22    }
23
24    static void metodoSemArgumentos(){
25      System.out.println("Oi, sou um método sem argumentos");
26    }
27
28    public static void main (String [] args) {
29      // exemplos de chamadas dos métodos
30      int resultadoSoma = somar(3, 2);
31      double resultadoPotencia = potencia(3, 3);
32      parOuImpar(8);
33      metodoSemArgumentos();
34    }
35  }
```

5.2 Escopo de Variáveis

O **escopo** de uma variável determina sua **visibilidade**: o(s) bloco(s) de código onde a variável pode ser acessada e modificada.

Argumentos de métodos e variáveis locais (aquelas declaradas dentro do método) tem

escopo de método. Ou seja, elas só podem ser acessadas e modificadas dentro do método onde foram declaradas.

O que delimita um escopo?
Resposta: em Java, um escopo é delimitado por um par de chaves.

5.3 Exercícios de Fixação

Exercício 5-1 Métodos

Escreva um método que calcula e retorna o somatório dos N primeiros números inteiros.

Exercício 5-2 Métodos

Modifique o método anterior para utilizar recursão.
Recursão: técnica em que um método chama a sí próprio para resolver uma parte menor do problema, desta forma quebrando o problema em pedaços menores. Sempre deve existir uma condição de parada, para finalizar a recursão.

5.4 Exercícios Práticos

Para cada exercício, há um **Laboratório de Programação** no Moodle.
Após implementar, submeta seu código para avaliação e atribuição de nota.

Exercício 5-3 ★ Métodos: Fatorial

Modifique o exercício 3-7 e crie um método para calcular o fatorial. O seu método deve receber um valor N e retornar o valor de $N!$.
A assinatura do método **obrigatoriamente** deve ser:

```
static long fatorial(int valor)
```

Exemplos de entradas	Saída: o valor de $N!$
5	125
0	1
9	362880
20	2432902008176640000

Exercício 5-4 ★★ Métodos: Salário Funcionário com Descontos

Estenda o exercício 1-5 para aplicar descontos de Imposto de Renda (IR) e de Previdência Social (INSS) no salário do funcionário.
Utilize as faixas a seguir para calcular o INSS:

- Salário bruto $\leq$ 1000 então desconto INSS é 8,5% do salário bruto.
- Salário bruto $>$ 1000 então desconto INSS é 9% do salário bruto.

Utilize as faixas a seguir para calcular o IR:

- Salário bruto $\leq$ 500 então desconto IR é zero.
- 500 $>$ Salário bruto $\leq$ 1000 então desconto IR é 5% do salário bruto.
- Salário bruto $>$ 1000 então desconto IR é 7% do salário bruto.

Métodos devem ser criados para calcular o desconto de IR e de INSS. Os nomes destes métodos devem ser exatamente conforme abaixo, inclusive maiúsculas/minúsculas. Os argumentos são conforme necessidade do cálculo.

- `calcularIR(<argumentos>)`
- `calcularINSS(<argumentos>)`

Entradas: igual ao exerc. 1-5

```
Bill Gates
180
35.29
3
```

```
Juca
200
14.99
5
```

Saídas: dados separados por `";"`, na ordem a seguir.

- `<nome>`
- `<salário bruto>`
- `<desconto INSS>`
- `<desconto IR>`
- `<salário líquido>`

```
Bill Gates;6923,90;623,15;484,67,5816,07
```

```
Juca;3447,70;310,29;241,34,2896,07
```

Exercício 5-5 ★★ Métodos: Fatorial Recursivo

Modifique o exercício 5-3 e transforme o método que calcula o fatorial em um **método recursivo**. As entradas e saídas seguem o mesmo formato.

Nenhuma estrutura de repetição deve ser utilizada em seu programa.

Exercício 5-6 ★★★ Métodos: Áreas de Figuras

Faça um programa Java que calcula e imprime a área das seguintes figuras geométricas: quadrado, retângulo, círculo e triângulo. As fórmulas para calcular a área são:

Quadrado: $lado^2$
Retângulo: $base * altura$
Círculo: $\pi * raio^2$
Triângulo: a área é calculada pelo teorema de Heron. Requer primeiro o cálculo de um valor s, para depois calcular a área.

$$s = \frac{(lado1 + lado2 + lado3)}{2}$$

$$area = \sqrt{s * (s - lado1) * (s - lado2) * (s - lado3)}$$

Para cada tipo de figura, deve-se implementar um método que calcula e retorna a área a partir das medidas necessárias. Os nomes destes métodos devem ser **exatamente** conforme abaixo, inclusive maiúsculas e minúsculas. Os argumentos são conforme o tipo de figura.

- `areaQuadrado(<argumentos>)`
- `areaRetangulo(<argumentos>)`
- `areaCirculo(<argumentos>)`
- `areaTriangulo(<argumentos>)`

O seu programa deve ler as medidas de uma figura, calcular e imprimir sua área.

Formato da entrada depende do tipo de figura:

- Quadrado: `Q;<lado>;`
- Retângulo: `R;<base>;<altura>;`
- Círculo: `C;<raio>;`
- Triângulo: `T;<lado1>;<lado2>;<lado3>;`

Saída: a área da figura

`Q;3,5;`	`12,25`
`R;6,2;7,9;`	`48,98`
`C;4,1;`	`52,81`
`T;7;8;9;`	`26,83`

Exercício 5-7 ♣ Métodos: Números Romanos

Faça um programa que leia uma `String` que representa um número em algarismos romanos (a `String` terá no máximo dez caracteres). O seu programa deve calcular e imprimir o número em algarismos arábicos. Os caracteres romanos e seus equivalentes são: M=1000; D=500; C=100; L=50; X=10; V=5; I=1.
Atenção: O seu programa deve ter um método que converte um caracter romano para seu valor arábico. A assinatura deste método deve ser:

```
static int converteChar(char caractere)
```

Entrada: String com número romano	Saídas: Número em arábico
`III`	`3`
`IV`	`4`
`LXXXVI`	`86`
`CCCXIX`	`319`
`MCCLIV`	`1254`

Referências Bibliográficas

Alvin Alexander. A 'printf' format reference page (cheat sheet). Disponível em: `https://alvinalexander.com/programming/printf-format-cheat-sheet`, 2019. Acesso em 03/02/2020.

David J. Barnes and Michael Kölling. *Programação Orientada a Objetos com Java: Uma Introdução Prática Utilizando o Blue J*. Pearson Prentice Hall, São Paulo, 2004.

David J. Barnes and Michael Kölling. *Programação Orientada a Objetos com Java: Uma Introdução Prática Utilizando o Blue J*. Pearson Prentice Hall, São Paulo, 2009.

H. M. Deitel. *Java: como programar*. Prentice-Hall, Porto Alegre, 2010.

Sonia Regina Di Giacomo. Sala de estudo: Algoritmo de Euclides para determinação de MDC. Disponível em: `http://clubes.obmep.org.br/blog/sala-de-estudos-algoritmo-de-euclides-para-determinacao-de-mdc/`, 2020? Acesso em 05/02/2019.

Rafael Santos. *Introdução à programação orientada a objetos usando JAVA*. Elsevier, Rio de Janeiro, 2 edition, 2013.

Kathy Sierra and Bert Bates. *Use a cabeça!: Java*. Alta Books, Rio de Janeiro, 2007.