

COLETÂNEA DE EXERCÍCIOS E NOTAS DE AULA EM LINGUAGEM DE PROGRAMAÇÃO JAVA

Com Introdução à Engine de Jogos JSGE
e Exercícios Criativos

Segunda Edição

DAVID BUZATTO

IFSP - CÂMPUS SÃO JOÃO DA BOA VISTA

COLETÂNEA DE EXERCÍCIOS E NOTAS DE AULA EM LINGUAGEM DE PROGRAMAÇÃO JAVA

COM INTRODUÇÃO À ENGINE DE JOGOS JSGE E EXERCÍCIOS

CRIATIVOS

Segunda Edição

PROF. DR. DAVID BUZATTO

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO
CÂMPUS SÃO JOÃO DA BOA VISTA**

**Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)**

Buzatto, David
Coletânea de exercícios e notas de aula em
linguagem de programação Java [livro eletrônico] :
com introdução à Engine de jogos JSGE e exercícios
criativos / David Buzatto. -- 2. ed. --
Vargem Grande do Sul, SP : Ed. do Autor, 2025.
PDF

Bibliografia.
ISBN 978-65-01-28801-7

1. Algoritmos de computadores 2. Ciência da
computação 3. Java (Linguagem de programação
para computadores) 4. Linguagem de programação
(Computadores) 5. Programação (Computadores)
I. Título.

25-246677

CDD-005.133

Índices para catálogo sistemático:

1. Java : Lingagem de programação : Computadores :
Processamento de dados 005.133

Aline Grazielle Benitez - Bibliotecária - CRB-1/3129

*À minha filha Aurora,
luz da minha vida*

*À minha esposa,
Fernanda*

*À minha mãe,
Selma*

SUMÁRIO

0.1	Preparação do Ambiente de Desenvolvimento	3
I	Construções Básicas da Linguagem de Programação Java	5
1	Entrada e Saída Padrão Formatados	7
1.1	Exemplos em Linguagem Java	8
1.1.1	Operadores Aritméticos e de Atribuição	19
1.2	Exercícios	21
1.3	Exercícios Criativos	47
2	Estruturas Condicionais	65
2.1	Estrutura Condicional <i>if</i> (se)	65
2.1.1	Exemplo e Diagrama de Fluxo da Estrutura Condicional <i>if</i>	66
2.1.2	Exemplo e Diagrama de Fluxo da Estrutura Condicional <i>if...else</i> (se...senão)	67
2.1.3	Exemplo e Diagrama de Fluxo da Estrutura Condicional <i>if...else f...else</i> (se...senão se...senão)	68
2.1.4	Operadores de Igualdade, Relacionais e Lógicos	69
2.1.5	Operador Ternário	70
2.1.6	O tipo <i>boolean</i>	73
2.1.7	Exercícios	75
2.2	Estrutura Condicional <i>switch</i> (escolha)	95
2.2.1	Exemplo e Diagrama de Fluxo da Estrutura Condicional <i>switch</i>	95
2.2.2	Exercícios	97
2.3	Exercícios Criativos	102
3	Estruturas de Repetição	109
3.1	Estrutura de Repetição <i>for</i>	109
3.1.1	Operadores Unários de Incremento e Decremento	110
3.1.2	Exemplo e Diagrama de Fluxo da Estrutura de Repetição <i>for</i> (para)	111
3.1.3	Exercícios	113
3.2	Estruturas de Repetição <i>while</i> (enquanto) e <i>do...while</i> (faça ... enquanto)	138
3.2.1	Exemplo e Diagrama de Fluxo da Estrutura de Repetição <i>while</i> e <i>do...while</i>	139
3.2.2	Exercícios	140
3.3	Exercícios Criativos	149
4	Arrays Unidimensionais	161
4.1	Exemplos em Linguagem Java	162
4.2	Representação Gráfica de Arrays	167

4.3	Exercícios	168
4.4	Exercícios Criativos	185
4.5	Desafios	187
5	Arrays Multidimensionais	189
5.1	Exemplos em Linguagem Java	190
5.2	Representação Gráfica de Arrays Multidimensionais	193
5.3	Exercícios	194
5.4	Exercícios Criativos	205
5.5	Desafios	207
6	Classe Math e Seus Métodos Matemáticos	209
6.1	Exemplos em Linguagem Java	210
6.2	Exercícios	216
6.3	Exercícios Criativos	222
6.4	Projetos	229
7	Métodos Estáticos	233
7.1	Exemplos em Linguagem Java	234
7.2	Exercícios	240
8	Caracteres e Strings	257
8.1	Exemplos em Linguagem Java	258
8.2	Exercícios	277
8.3	Exercícios Criativos	299
9	Arquivos	305
9.1	Exemplos em Linguagem Java	306
9.2	Exercícios	311
10	Recursividade	315
10.1	Exercícios	321
II	Introdução à Programação Orientada a Objetos em Java	325
11	Classes, Atributos e Métodos	327
11.1	Exemplos em Linguagem Java	328
11.2	Representação em UML	342
11.3	Exercícios	343
11.4	Exercícios Criativos	360
11.5	Exemplos em Linguagem Java	360
11.6	Projetos	390
12	Encapsulamento	393
12.1	Exemplos em Linguagem Java	394
12.2	Representação em UML	407
13	Composição, Agregação e Associação	409

13.1 Exemplos em Linguagem Java	410
13.2 Representação em UML	423
13.3 Exercícios	424
14 Herança e Polimorfismo	435
14.1 Exemplos em Linguagem Java	436
14.2 Representação em UML	447
15 Interfaces	449
15.1 Exemplos em Linguagem Java	450
15.2 Representação em UML	457
16 Enumerações	459
16.1 Exemplos em Linguagem Java	460
16.2 Representação em UML	464
17 Records	465
17.1 Exemplos em Linguagem Java	466
III Tópicos Complementares	469
18 Java Collections Framework	471
18.1 Pilhas, Filas e Deques	471
18.2 Listas	479
18.3 Conjuntos	481
18.4 Mapas	483
19 Expressões Lambda e Streams	489
19.1 Exemplos em Linguagem Java	490
IV Finalizando...	495
20 Conclusão	497
Bibliografia	499

APRESENTAÇÃO

“Any fool can write code that a computer can understand. Good programmers write code that humans can understand”.

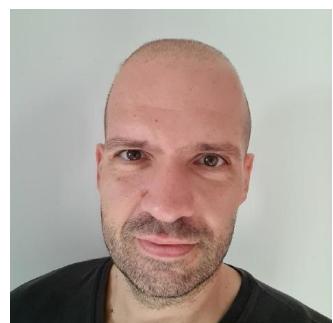
Martin Fowler



REZADO aluno e leitor, seja bem-vindo! Este livro contém diversos Capítulos, organizados de forma a guiá-lo no processo de fixação dos conteúdos aprendidos em aula, nas disciplinas de Programação Orientada a Objetos (POO) que usam a linguagem de programação Java como opção de implementação, por meio de exercícios e desafios aplicados nessa linguagem. A ordem dos Capítulos obedece a um caminho lógico que será empregado pelo professor no seu processo de aprendizagem, ou seja, a ordem dos Capítulos segue a ordem cronológica dos tópicos que serão apresentados, ensinados e treinados em laboratório. Note que apesar deste livro ter sido organizado para apoiar no desenvolvimento de disciplinas, nada impede que seja usado como material de apoio para outros cursos, bem como para pessoas que desejam treinar por meio de exercícios suas habilidades com programação básica e POO. Ainda, existem alguns Capítulos na edição atual em que não existem exercícios, pois a ideia do livro é focar em assuntos primordiais, mas nada impede que em futuras edições isso seja alterado. Nesta segunda edição, são apresentadas novidades, como um guia para preparação do seu ambiente de desenvolvimento em Windows e exercícios criativos, onde será utilizada uma Engine de desenvolvimento de jogos e simulações 2D em Java, chamada JSimple Game Engine (JSGE). Falaremos mais sobre ela posteriormente.

Antes de começarmos, eu gostaria de me apresentar:

Meu nome é David Buzatto e sou Bacharel em Sistemas de Informação pela Fundação de Ensino Octávio Bastos (2007), Mestre em Ciência da Computação pela Universidade Federal de São Carlos (2010) e Doutor em Biotecnologia pela Universidade de Ribeirão Preto (2017). Tenho interesse em construção de compiladores, teoria da computação, análise e projeto de algoritmos, estruturas de dados, algoritmos em bioinformática e desenvolvimento de jogos eletrônicos. Atualmente sou professor efetivo do Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP), câmpus São João da Boa Vista. A melhor forma de contatar é através do e-mail davidbuzatto@ifsp.edu.br.



Para que você possa aproveitar a leitura deste livro de forma plena, vale a pena entender alguns

padrões que foram utilizados neste texto. As cinco caixas apresentadas abaixo serão empregadas para mostrar, a você leitor, respectivamente, boas práticas de programação, exemplos, conteúdos complementares para melhorar e aprofundar seu aprendizado, dicas pertinentes ao que está sendo apresentado e, por fim, itens que precisam ser tratados com cuidado ou que podem acarretar em erros comuns de programação.

🔗 | Boa Prática

As caixas de **Boa Prática** serão usadas para apresentar sugestões de como se deve escrever código-fonte de maneira limpa, organizada e legível, bem como outras técnicas e padrões para auxiliar nesses objetivos.

| Exemplo

Pequenos trechos de código serão mostrados nas caixas de **Exemplo**, além de informações relativas aos códigos mostrados.

ⓘ | Saiba Mais

Nas caixas de **Saiba Mais** serão apresentados recursos complementares para você conhecer mais sobre determinado assunto.

💡 | Dica

As caixas de **Dica** vão ser usadas com o objetivo de apresentar uma dica ou complementar alguma informação.

⚠️ | Atenção!

Quando for necessário realizar alguma observação mais importante, serão empregadas as caixas de **Atenção!**.

Além disso, diversos exercícios conterão caixas de entrada e/ou saída, em que serão apresentados exemplos de dados de entrada para o problema proposto e de saída que devem ser obtidos após o processamento da entrada fornecida como exemplo.

Note também que este livro foi escrito de forma quase coloquial, com o objetivo de conversar com você, tentando te ensinar, e não com o objetivo de ser um material de pesquisa.

É de suma importância que você resolva cada um dos exercícios básicos de cada Capítulo, visto que a utilização de uma linguagem de programação, e mais importante ainda, a obtenção de maturidade no desenvolvimento de algoritmos e na aplicação do paradigma orientado a objetos, é ferramenta primordial para o seu sucesso profissional e intelectual na área da Computação.

Um ponto importante sobre os exercícios é que todas as saídas apresentadas ao usuário, quando feitas em modo texto, serão feitas sem usar acentos. Ou seja, se um programa precisar exibir uma palavra acentuada, algo como, por exemplo, “Olá”, você deverá digitar tal palavra sem usar o acento, ficando “Ola”. O principal motivo para essa restrição é que normalmente a página de códigos utilizada para executar o programa em modo texto diverge da codificação utilizada na compilação, criando inconsistências, principalmente ao se usar as linguagens C e C++ em ambiente Windows.

Atualmente isso acontece também na linguagem Java. Por isso, nas entradas e saídas de todos os enunciados, você perceberá que faltarão acentos nas palavras acentuadas, o que é feito de propósito visto essa “restrição”. Há como contornar tal característica, mas isso envolve alguns detalhes que fogem do escopo desta obra e que lhe farçaria a inserir código no seu programa que não faz parte da solução, nem dos conceitos que devem ser aprendidos. Usando a linguagem Python ou versões mais antigas da linguagem Java não haverá tal problema, entretanto, será mantido o padrão de não usar acentos para todos os exercícios. Essas quatro linguagens de programação foram citadas, pois serão linguagens que, dependendo da disciplina em que essa lista for usada, poderão estar sendo ensinadas/aplicadas. Os programas de exemplo conterão acento normalmente.

Um outro ponto que merece atenção é que dada a quantidade de exercícios e os detalhes para os redigir, testar etc., podem haver erros de digitação ou algum tipo de informação passada equivocadamente. Caso perceba qualquer problema, peço que faça a gentiliza de me contatar via e-mail, apontando o erro e a página, para assim eu poder fazer a correção e disponibilizar a versão atualizada do livro o quanto antes. Sugestões e críticas são muito bem-vindas!

Por fim, espero sinceramente que este livro lhe seja útil! Vamos começar?

0.1 Preparação do Ambiente de Desenvolvimento

Iniciaremos nosso estudo preparando o ambiente que você utilizará para aprender a programar usando a linguagem de programação Java e realizar os exercícios propostos no livro. Ficaremos em três tarefas. Para que o processo seja o mais fácil possível, a explicação será apresentada no vídeo acessível através do link <<https://youtu.be/7kTMZvR1v5Y>>. Assista-o, seguindo o passo a passo. Ao terminar, você terá o ambiente pronto para poder trabalhar durante o semestre ao acompanhar o livro. Caso não esteja fazendo a disciplina na qual esse livro é baseado, não há problema também, pois você terá um ambiente básico pronto para uso da mesma forma!

Parte I

Construções Básicas da Linguagem de Programação Java

ENTRADA E SAÍDA PADRÃO FORMATADOS

“A mente que se abre a uma nova ideia jamais voltará ao seu tamanho original”.

Albert Einstein



ESTE Capítulo serão treinados os conceitos E/S (Entrada e Saída)¹, sendo estes fundamentais para o funcionamento de qualquer programa de computador. Além disso, serão utilizadas variáveis e outros comandos, além de expressões básicas aprendidas em aula. Os trechos de código abaixo contém lembretes das estruturas principais que devem ser usadas para resolver os exercícios propostos. Note que em todos os Capítulos a maioria dos conceitos necessários serão apresentados dentro de trechos de código mostrados no início de cada um. Sendo assim, sempre estude os códigos apresentados no início de cada Capítulo e leia com atenção os comentários inseridos neles.

¹Em inglês o termo é I/O que vem de *Input* (Entrada) e *Output* (Saída).

1.1 Exemplos em Linguagem Java

Saída padrão usando o método println

```
1  /*
2   * Arquivo: capitulo01/EntradaSaidaPadraoFormatadosOláMundo.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6 public class EntradaSaidaPadraoFormatadosOláMundo {
7
8     // o método main é o ponto de entrada dos nossos programas
9     public static void main( String[] args ) {
10
11         // comentários de uma linha iniciam com // (duas barras)
12
13         // o método println direciona o texto inserido (entre aspas duplas)
14         // para a saída padrão e pula uma linha
15         System.out.printf( "Olá mundo!" );
16
17     }
18
19 }
```

Caracteres de escape comuns

```
1  /*
2   * Arquivo: capitulo01/EntradaSaidaPadraoFormatadosSaida.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6  public class EntradaSaidaPadraoFormatadosSaida {
7
8      public static void main( String[] args ) {
9
10         // comentários de múltiplas linhas iniciam com /* e terminam com */
11
12         /*
13          * O método print direciona o texto inserido (entre aspas duplas)
14          * para a saída padrão, sem pular linha ao final.
15          *
16          * O caractere \ (contra barra) é usado para "escapar" caracteres
17          * especiais. Os principais são \n e \t servindo, respectivamente,
18          * para pular uma linha do console e continuar a saída de texto
19          * no ínicio da próxima linha e para inserir um caractere de
20          * tabulação.
21          */
22         System.out.print( "Um texto!\n" );
23         System.out.print( "Outro texto, na linha de baixo!\n" );
24         System.out.print( "\tEssa linha inicia tabulada!" );
25
26     }
27
28 }
```

Declaração de variáveis

Especificadores de Formato

```
1  /*
2   * Arquivo: capitulo01/EntradaSaidaPadraoFormatadosFormatacao.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6  public class EntradaSaidaPadraoFormatadosFormatacao {
7
8      public static void main( String[] args ) {
9
10         /*
11          * O método printf é capaz de interpolar dados dentro
12          * do texto (string) que irá imprimir na saída padrão.
13          * seu funcionamento é similar ao da linguagem C.
14          *
15          * Para isso, é necessário marcar posições dentro da string
16          * usando o caractere % (porcento) e utilizar um especificador
17          * de formato específico para cada tipo de variável.
18          *
19          * Os especificadores de formato que serão usados por
20          * enquanto são:
21          *     %d: para variáveis inteiiras (int)
22          *     %c: para variáveis de caracteres (char)
23          *     %f: para variáveis decimais (double)
24          *
25          * Alguns especificadores possuem opções de formatação.
26          * por exemplo, para fixar a quantidade de casas decimais
27          * que serão exibidas para uma variável double, usa-se:
28          *     %.nf: onde "n" é a quantidade de casas decimais.
29          *     Exemplo: %.2f => o valor da variável double
30          *               será formatado usando duas casas decimais
31         */
32
33         double pi = 3.1415;
34         double raio = 20.78;
35         double circunferencia = 2 * pi * raio;
36         double area = pi * raio * raio;
37
38         System.out.printf( "O circulo de raio %f tem:\n", raio );
39         System.out.printf( "\tCircunferencia = %.2f\n", circunferencia );
40         System.out.printf( "\tArea = %.2f\n", area );
41
42     }
43
44 }
```

Operadores aritméticos

```
1  /*
2   * Arquivo: capitulo01/EntradaSaidaPadraoFormatadosAritmetica.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6  public class EntradaSaidaPadraoFormatadosAritmetica {
7
8      public static void main( String[] args ) {
9
10         /*
11          * Existem na linguagem Java cinco operadores aritméticos:
12          *      +: adição
13          *      -: subtração
14          *      *: multiplicação
15          *      /: divisão
16          *      %: módulo (resto da divisão inteira)
17          *
18          * Esses operadores atuam de forma específica dependendo do
19          * tipo numérico sendo operado. Isso se nota principalmente
20          * em relação ao operador / (divisão) quando atuado em valores
21          * inteiros e de ponto flutuante!
22          *
23          * O operador de resto/módulo que é dado pelo símbolo
24          * % (porcento), é usado apenas para números inteiros.
25          */
26
27      int numeroInteiro1 = 9;
28      int numeroInteiro2 = 2;
29      double numeroDecimal1 = 9;
30      double numeroDecimal2 = 2;
31
32      // resulta em 4
33      int divisaoInteira = numeroInteiro1 / numeroInteiro2;
34
35      // resulta em 4.5
36      double divisaoDecimal = numeroDecimal1 / numeroDecimal2;
37
38      System.out.printf( "Inteiros: %d / %d = %d\n",
39                         numeroInteiro1, numeroInteiro2, divisaoInteira );
40      System.out.printf( "Decimais: %f / %f = %f\n",
41                         numeroDecimal1, numeroDecimal2, divisaoDecimal );
42
43  }
```

Entrada padrão usando a classe Scanner

```
1  /*
2   * Arquivo: capitulo01/EntradaSaidaPadraoFormatadosEntrada.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6 // importa a classe Scanner do pacote java.util para ser utilizada
7 import java.util.Scanner;
8
9 public class EntradaSaidaPadraoFormatadosEntrada {
10
11     public static void main( String[] args ) {
12
13         /*
14          * Para realizar a entrada de dados em um programa
15          * pelo dispositivo de entrada padrão configurado no sistema
16          * operacional (usualmente o teclado) usa-se a classe Scanner.
17          *
18          * Um objeto dessa essa classe será "ligado" ao fluxo de
19          * entrada padrão do processo e, através desse objeto,
20          * processaremos os dados que serão fornecidos, convertendo-os
21          * para os tipos apropriados.
22          */
23
24     char primeiraLetra;
25     int idade;
26     double peso;
27     double altura;
28     double imc;
29
30     /*
31      * Declara uma variável do tipo Scanner, chamada scan
32      * e a inicializa com um novo objeto do tipo Scanner
33      * ligado ao fluxo de entrada padrão do processo (System.in)
34      */
35     Scanner scan = new Scanner( System.in );
36
37     System.out.printf( "Entre com sua idade: " );
38
39     /*
40      * Para evitar problemas, por enquanto sempre leremos
41      * uma linha INTEIRA do Scanner e então converteremos
42      * esses dados lidos. A leitura da linha é feita através
43      * do método nextLine().
44      *
45      * Perceba que na linha abaixo, usamos o método parseInt
46      * da classe empacotadora Integer para converter o texto
47      * lido através do método nextLine(), do objeto do tipo
```

```
48     * Scanner, para um valor inteiro.  
49     */  
50     idade = Integer.parseInt( scan.nextLine() );  
51  
52     System.out.printf( "Entre com seu peso: " );  
53  
54     /*  
55      * Na linha abaixo, usamos o método parseDouble  
56      * da classe empacotadora Double para converter o texto  
57      * lido através do método nextLine(), do objeto do tipo  
58      * Scanner, para um valor decimal. A entrada dos dados  
59      * deve ser feita usando o ponto (.) como separador decimal.  
60      */  
61     peso = Double.parseDouble( scan.nextLine() );  
62  
63     System.out.printf( "Entre com sua altura: " );  
64     altura = Double.parseDouble( scan.nextLine() );  
65  
66     System.out.printf(  
67         "Entre com a primeira letra de seu primeiro nome: "  
68     );  
69  
70     /*  
71      * Para a leitura de um caractere, basta pegarmos o  
72      * primeiro caractere da linha lida.  
73      */  
74     primeiraLetra = scan.nextLine().charAt( 0 );  
75  
76     imc = peso / ( altura * altura );  
77  
78     System.out.printf( "%c (idade %d), seu IMC é: %.2f",  
79                     primeiraLetra, idade, imc );  
80  
81     /*  
82      * Após o uso do Scanner, precisamos liberar os recursos  
83      * alocados à ele, nesse caso, a entrada padrão. Como  
84      * nossos programas serão simples e não ficarão ocupando  
85      * tal recurso por muito tempo, podemos omitir a linha  
86      * abaixo nos exercícios, pois ao terminar a execução  
87      * do método main, a máquina virtual Java será finalizada  
88      * e quaisquer recursos ocupados por ela serão liberados.  
89      */  
90     scan.close();  
91  
92 }  
93  
94 }
```

Tratamento básico de exceções

```
1  /*
2   * Arquivo: capitulo01/EntradaSaidaPadraoFormatadosExcecoes.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6 import java.util.Scanner;
7
8 public class EntradaSaidaPadraoFormatadosExcecoes {
9
10    public static void main( String[] args ) {
11
12        /*
13         * Na linguagem Java há um mecanismo de tratamento de exceções que
14         * tem como objetivo "detectar" situações problemáticas durante
15         * a execução do código de modo que, ao ser usado apropriadamente,
16         * podemos, na maioria das vezes, escrever código que seja capaz
17         * de se "recuperar" de tais situações.
18         */
19
20        int numeroInteiro;
21        double numeroDecimal;
22        Scanner scan = new Scanner( System.in );
23
24        /*
25         * Neste exemplo vamos forçar erros de formatação. Sempre que
26         * estivermos usando os métodos "parse" das classes empacotadoras
27         * pode ocorrer uma situação onde o usuário fornece os dados
28         * esperados da forma incorreta. Por exemplo, fornecer uma letra
29         * ou palavra ao invés de um número inteiro ou decimal, ou então
30         * errar ao fornecer o valor de um número decimal trocando o
31         * separador decimal esperado, que é o ponto por uma vírgula, que
32         * é o separador que usamos no Brasil. Ou seja, o valor 1.75 deve
33         * ser aceito, mas o valor 1,75 contém um erro em seu formato.
34         */
35
36        /*
37         * O objetivo ao executar essa classe é realmente forçar os erros
38         * para vermos o que acontecerá. Execute essa classe e forneça
39         * um valor que NÃO representa um valor inteiro.
40         */
41        System.out.println( "Entre com um valor inteiro incorreto: " );
42        numeroInteiro = Integer.parseInt( scan.nextLine() );
43
44        // caso a exceção seja lançada, a próxima linha não será alcançada
45        System.out.println( "Valor fornecido: " + numeroInteiro );
46
47        /*

```

```
48 * Ao fazer o que foi instruído, você deve ter visto algo assim
49 * na saída:
50 *
51 * Exception in thread "main" java.lang.NumberFormatException...
52 *
53 * Veja que a JVM indicou que houve um caso excepcional na execução
54 * do código, pois o que estava sendo esperado (um inteiro) não foi
55 * fornecido. Para indicar isso, foi lançada uma exceção do tipo
56 * NumberFormatException, ou seja, uma exceção que indica um formato
57 * numérico incorreto.
58 *
59 * Existem dois tipos de exceções na linguagem Java. As exceções
60 * checadas e as não checadas. A NumberFormatException é uma exceção
61 * não checada, pois o programador não precisa obrigatoriamente
62 * trata-la. Se nós estivermos lidando com a abertura de um arquivo
63 * para podemos processá-lo (veremos isso mais adiante) normalmente
64 * precisaremos tratar uma exceção checada chamada
65 * FileNotFoundException, pois pode se que o arquivo que vai ser
66 * lido talvez não exista mais no sistema de arquivos subjacente.
67 *
68 * Mesmo que uma exceção não seja checada, podemos tratá-la e é isso
69 * que faremos agora. Para isso, usamos a construção "try" (tente).
70 * Existem diversas variações de como podemos utilizá-la, mas vamos
71 * nos focar na forma mais usual.
72 *
73 * Execute novamente essa classe e agora não force o erro anterior,
74 * ou seja, forneça de fato um número inteiro, para que a execução
75 * chegue nas próximas linhas. O fornecimento do valor decimal
76 * que agora deve ser fornecido de forma incorreta.
77 */
78 try { // tente
79
80     System.out.println( "Entre com um valor decimal incorreto: " );
81
82     // converter o texto fornecido na entrada para um double
83     numeroDecimal = Double.parseDouble( scan.nextLine() );
84
85     // caso a exceção seja lançada, a próxima linha não será
86     // alcançada
87     System.out.println( "Valor fornecido: " + numeroDecimal );
88
89     /*
90      * Caso algo dentro do bloco try lance a exceção
91      * NumberFormatException, o catch (capturar) correspondente
92      * "capturará" tal exceção e teremos a oportunidade de
93      * recolocar o programa em um estado válido.
94      */
95 } catch ( NumberFormatException exc ) {
```

```
97         System.out.println( "Formato incorreto fornecido!" );
98         numeroDecimal = 0;
99
100    }
101
102   /*
103    * Note que agora que aprendemos a usar tal construção, somos
104    * capazes de tratar tais situações excepcionais. Nos nossos
105    * exercícios não precisamos lidar com isso, mas no mundo real,
106    * em muitas situações teremos que realizar tal tratamento.
107    */
108
109    scan.close();
110
111 }
112
113 }
```

💡 | Boa Prática

Sempre utilize comentários no código fonte com o objetivo de auxiliar você e/ou outro programador que fará a leitura do código posteriormente.

 | Boas Práticas

- Declare uma variável por linha!
 - Ao nomear variáveis, dê preferência para nomes que identificam corretamente o propósito delas e inicie seus nomes com letras minúsculas;
 - Um padrão comum para nomenclatura de variáveis em linguagens de programação mais modernas é chamado de CamelCase. Nesse padrão, caso uma variável seja uma palavra composta ou uma frase, une-se todas as palavras e as iniciamos com letras maiúsculas. Por exemplo, se quisermos declarar uma variável chamada “altura da pessoa”, faríamos algo assim: alturaDaPessoa ;

| Exemplo

No exemplo apresentado abaixo é mostrada a declaração e inicialização de três variáveis, sendo que uma representação gráfica didática do que acontece na memória principal após a execução desse código é apresentada na Figura 1.1.

Declaração e inicialização de variáveis

```
1 int idade = 38;  
2 char letra = 'd';  
3 double altura = 1.84;
```

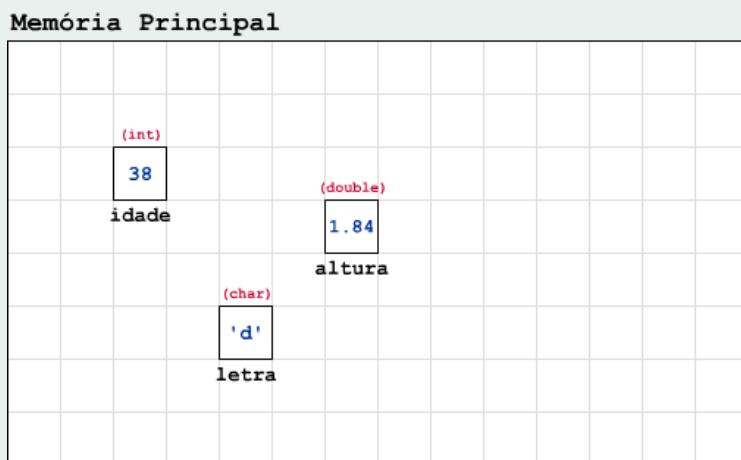


Figura 1.1: Variáveis na memória

| Saiba Mais

Uma descrição do padrão CamelCase pode ser encontrada no link <<https://pt.wikipedia.org/wiki/CamelCase>>. Esse padrão será usado praticamente o tempo todo enquanto você programar. Note que algumas linguagens possuem seus próprios padrões de codificação

dependendo de suas respectivas comunidades e, além disso, cada empresa que você trabalhar poderá ter um padrão específico.

⚠ | Atenção!

- Não é permitido iniciar o nome (identificador) de uma variável utilizando números;
- Use somente letras (sem acentos), números, “_” (*underscore*) e “\$” (cifrão) para nomear uma variável. Na linguagem Java é permitido usar uma infinidade de caracteres para nomear variáveis, inclusive caracteres acentuados, mas vamos nos ater a esses indicados;
- Identificadores de variáveis não podem conter espaços.

💡 | Dica

Para imprimir um símbolo de porcentagem (%) na saída ao se usar o método `System.out.printf`, preceda-o de outro símbolo de porcentagem. Por exemplo:

```
1 System.out.printf( "9%" );      // imprime 9%
```

1.1.1 Operadores Aritméticos e de Atribuição

Na Tabela 1.1 abaixo são apresentados os operadores aritméticos e na Tabela 1.2 os operadores de atribuição da linguagem Java. A precedência dos operadores aritméticos é a mesma da álgebra, podendo também serem agrupados usando parênteses.

Operador	Significado
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da Divisão Inteira (módulo)

Tabela 1.1: Operadores aritméticos

| Exemplo

Usando o operador aritmético de adição

```
1 int a = 1;
2 int b = 2;
3
4 // a variável r conterá o resultado da soma de a e b
5 int r = a + b;
```

Operador	Significado
=	Atribuição simples
+=	Atribuição composta (adição)
-=	Atribuição composta (subtração)
*=	Atribuição composta (multiplicação)
/=	Atribuição composta (divisão)
%=	Atribuição composta (resto)

Tabela 1.2: Operadores de atribuição e atribuição composta

| Exemplo

Usando o operador de atribuição composto para adição

```

1 int a = 1;
2 int b = 1;
3 b += a; // o mesmo que b = b + a;
4 // idem para os outros operadores de atribuição compostos

```

ⓘ | Saiba Mais

Caso queira ler mais sobre as especificações das diversas versões da linguagem Java, você pode consultar o link <<https://docs.oracle.com/javase/specs/>>. A documentação da *Application Programming Interface*^a (API) da versão 17 da linguagem Java é acessível através do link <<https://docs.oracle.com/en/java/javase/17/docs/api/index.html>>.

^aA API, em tradução livre Interface de Programação de Aplicações, consiste no conjunto de todos os recursos implementados em uma determinada linguagem de programação, ou *framework* ou biblioteca.

Agora que já conhecemos o básico da linguagem de programação Java, nós podemos começar a trabalhar nos exercícios. Como já frisado, o desenvolvimento das tarefas é ESSENCIAL para o sucesso no aprendizado de qualquer linguagem de programação. Vamos lá!

1.2 Exercícios

Exercício 1.1:

Escreva um programa que imprima a mensagem “Ola Mundo!” quando executado.

Arquivo com a solução: Exercicio1\$1.java

Saída

Ola Mundo !

⚠ | Atenção!

Note que o primeiro exercício possui somente a caixa de **Saída**, indicando que nesse exercício seu programa deve apenas gerar saída para o usuário. Vários dos próximos exercícios seguirão a mesma ideia.

Exercício 1.2 (DEITEL; DEITEL, 2017):

Escreva um programa que imprima o seguinte desenho quando executado.

Arquivo com a solução: Exercicio1\$2.java

Saída

```
****  
*****  
* ****  
*****
```

⚠ | Atenção!

Em alguns exercícios será necessário apresentar espaços na saída de modo a “espaçar caracteres”. Para que a quantidade de espaços fique evidente em cada linha, nessas saídas eles serão apresentados como asteriscos quase pretos. Esse padrão será usado no decorrer do livro.

Exercício 1.3 (DEITEL; DEITEL, 2017):

Escreva um programa que imprima o seguinte desenho quando executado.

Arquivo com a solução: Exercicio1\$3.java

Saída

```
#####  
#*****#  
#*****#  
#*****#  
#*****#  
#####
```

Exercício 1.4 (DEITEL; DEITEL, 2017):

Escreva um programa que imprima o seguinte desenho quando executado.

Arquivo com a solução: Exercicio1\$4.java

Saída

```
*****  
*****  
*****  
****  
***  
**
```

Exercício 1.5 (DEITEL; DEITEL, 2017):

Escreva um programa que imprima o seguinte desenho quando executado.

Arquivo com a solução: Exercicio1\$5.java

Saída



A partir do próximo exercício começaremos a trabalhar com a entrada de dados do usuário, sendo assim, além da caixa de **Saída**, será apresentada também a caixa de **Entrada**, com o objetivo lhe mostrar os dados que foram fornecidos ao programa para que ele gerasse a saída apropriada.

Exercício 1.6:

Escreva um programa que peça para o usuário fornecer o valor de dois números inteiros. O programa deve usar o valor dos números para calcular o valor das quatro operações aritméticas básicas (adição, subtração, multiplicação e divisão). O resultado de cada operação deve ser armazenado em uma variável diferente. No final, o programa deve exibir ao usuário o resultado de cada operação.

Arquivo com a solução: Exercicio1\$6.java

Entrada

```
Primeiro numero: 7  
Segundo numero: 3
```

Saída

```
7 + 3 = 10  
7 - 3 = 4  
7 * 3 = 21  
7 / 3 = 2
```

Exercício 1.7:

Escreva um programa que peça para o usuário fornecer o valor do lado de um quadrado em uma unidade arbitrária. O valor deve ser um número inteiro. O programa deve calcular os valores da área e do perímetro desse quadrado. O resultado de cada cálculo deve ser armazenado em uma variável. No final, o programa deve exibir ao usuário os valores obtidos. Lembrando que:

- $P = 4l$
- $A = l^2$
- Onde:
 - P é o perímetro do quadrado;
 - A é a área do quadrado;
 - l é o valor do lado do quadrado.

Arquivo com a solução: Exercicio1\$7 . java

Entrada

Valor do lado: 5

Saída

Perimetro = 20
Area = 25

Exercício 1.8:

Escreva um programa que peça para o usuário fornecer os valores da largura e da altura de um retângulo em uma unidade arbitrária. Os valores devem ser números inteiros. O programa deve calcular os valores da área e perímetro desse retângulo. O resultado de cada cálculo deve ser armazenado em uma variável. No final, o programa deve exibir ao usuário os valores obtidos. Lembrando que:

- $P = (2l) + (2h)$
- $A = lh$
- Onde:
 - P é o perímetro do retângulo;
 - A é a área do retângulo;
 - l é o valor da largura do retângulo;
 - h é o valor da altura do retângulo.

Arquivo com a solução: Exercicio1\$8.java

Entrada

```
Valor da largura: 5
Valor da altura: 10
```

Saída

```
Perimetro = 30
Area = 50
```

Exercício 1.9:

Escreva um programa que peça para o usuário fornecer os valores da base e da altura de um triângulo em uma unidade arbitrária. Os valores devem ser números inteiros. O programa deve calcular o valor da área desse triângulo. O resultado deve ser armazenado em uma variável. No final, o programa deve exibir ao usuário o valor obtido. Lembrando que:

- $A = \frac{bh}{2}$
- Onde:
 - A é a área do triângulo;
 - b é o valor da base do triângulo;
 - h é o valor da altura do triângulo.

Arquivo com a solução: Exercicio1\$9.java

Entrada

```
Valor da base: 10  
Valor da altura: 5
```

Saída

```
Area = 25
```

Exercício 1.10:

Escreva um programa que peça para o usuário fornecer os valores da base maior, da base menor e da altura de um trapézio em uma unidade arbitrária. Os valores devem ser números inteiros. O programa deve calcular o valor da área desse trapézio. O resultado deve ser armazenado em uma variável. No final, o programa deve exibir ao usuário o valor obtido. Lembrando que:

- $A = \frac{(B + b)h}{2}$
- Onde:
 - A é a área do trapézio;
 - B é o valor da base maior do trapézio;
 - b é o valor da base menor do trapézio;
 - h é o valor da altura do trapézio.

Arquivo com a solução: Exercicio1\$10.java

Entrada

```
Valor da base maior: 10
Valor da base menor: 6
Valor da altura: 5
```

Saída

```
Area = 40
```

Exercício 1.11:

Escreva um programa que peça para o usuário fornecer os valores da diagonal maior e da diagonal menor de um losango em uma unidade arbitrária. Os valores devem ser números inteiros. O programa deve calcular o valor da área desse losango. O resultado deve ser armazenado em uma variável. No final, o programa deve exibir ao usuário o valor obtido. Lembrando que:

- $A = \frac{Dd}{2}$
- Onde:
 - A é a área do losango;
 - D é o valor da diagonal maior do losango;
 - d é o valor da diagonal menor do losango.

Arquivo com a solução: Exercicio1\$11.java

Entrada

```
Valor da diagonal maior: 12  
Valor da diagonal menor: 6
```

Saída

```
Area = 36
```

⚠ | Atenção!

A partir de agora, a maioria dos nossos programas que lidarão com números farão uso do tipo **double** (decimal) além do tipo **int** (inteiro). O tipo apropriado dependerá da natureza do valor, bem como de possíveis orientações contidas na atividade. Em todos os exercícios os números decimais precisarão ser formatados com uma quantidade específica de casas decimais. Essa formatação será normalmente informada nas tarefas, mas caso não sejam, confira na saída a quantidade de casas usadas. Normalmente essa quantidade será duas casas decimais.

Exercício 1.12:

Escreva um programa que peça para o usuário fornecer um valor qualquer que deve ser um número decimal. O programa deve exibir esse número três vezes usando o método `printf`: Na primeira, deve ser exibido o número sem nenhuma formatação. Na segunda, o número deve ser formatado para mostrar duas casas decimais. Por fim, na terceira, o número deve ser formatado para mostrar três casas decimais.

Arquivo com a solução: Exercicio1\$12.java

Entrada

Entre com um valor qualquer: 153.4671

Saída

153.467100

153.47

153.467

Exercício 1.13:

Repita o Exercício 1.6, usando agora números decimais. Os resultados devem ser formatados usando duas casas decimais. Reescreva o programa ao invés de copiá-lo!

Arquivo com a solução: Exercicio1\$13.java

Entrada

```
Primeiro numero: 7.5  
Segundo numero: 3.5
```

Saída

```
7.50 + 3.50 = 11.00  
7.50 - 3.50 = 4.00  
7.50 * 3.50 = 26.25  
7.50 / 3.50 = 2.14
```

Exercício 1.14:

Repita o Exercício 1.7, usando agora números decimais. Os resultados devem ser formatados usando duas casas decimais. Reescreva o programa ao invés de copiá-lo!

Arquivo com a solução: Exercicio1\$14.java

Entrada

Valor do lado: 5.5

Saída

Perímetro = 22.00

Área = 30.25

Exercício 1.15:

Repita o Exercício 1.8, usando agora números decimais. Os resultados devem ser formatados usando duas casas decimais. Reescreva o programa ao invés de copiá-lo!

Arquivo com a solução: Exercicio1\$15.java

Entrada

```
Valor da largura: 5.5  
Valor da altura: 9.5
```

Saída

```
Perimetro = 30.00  
Area = 52.25
```

Exercício 1.16:

Repita o Exercício 1.9, usando agora números decimais. Os resultados devem ser formatados usando duas casas decimais. Reescreva o programa ao invés de copiá-lo!

Arquivo com a solução: Exercicio1\$16.java

Entrada

```
Valor da base: 10.5  
Valor da altura: 5.75
```

Saída

```
Area = 30.19
```

Exercício 1.17:

Repita o Exercício 1.10, usando agora números decimais. Os resultados devem ser formatados usando duas casas decimais. Reescreva o programa ao invés de copiá-lo!

Arquivo com a solução: Exercicio1\$17.java

Entrada

```
Valor da base maior: 10.5  
Valor da base menor: 6.25  
Valor da altura: 6.75
```

Saída

```
Area = 56.53
```

Exercício 1.18:

Repita o Exercício 1.11, usando agora números decimais. Os resultados devem ser formatados usando duas casas decimais. Reescreva o programa ao invés de copiá-lo!

Arquivo com a solução: Exercicio1\$18.java

Entrada

```
Valor da diagonal maior: 12.25  
Valor da diagonal menor: 6.6
```

Saída

```
Area = 40.42
```

Exercício 1.19:

Escreva um programa que peça para o usuário fornecer o valor do raio de um círculo em uma unidade arbitrária. O valor deve ser um número decimal. O programa deve calcular os valores do diâmetro, da circunferência e da área desse círculo. O resultado de cada cálculo deve ser armazenado em uma variável. No final, o programa deve exibir ao usuário os valores obtidos. Lembrando que:

- $D = 2r$
- $C = 2\pi r$
- $A = \pi r^2$
- Onde:
 - D é o diâmetro do círculo;
 - C é a circunferência do círculo;
 - A é a área do círculo;
 - r é o valor do raio do círculo;
 - π é a constante matemática Pi. Para esse exercício, considere que $\pi = 3.141592$.

Arquivo com a solução: Exercicio1\$19.java

Entrada

Valor do raio do circulo: 10.5

Saída

Diametro = 21.00
Circunferencia = 65.97
Area = 346.36

Exercício 1.20:

Escreva um programa que peça para o usuário fornecer dois números inteiros. O programa deve calcular e exibir a média aritmética desses dois números. Armazene essa média em uma variável.

Arquivo com a solução: Exercicio1\$20.java

Entrada

```
Primeiro numero: 5  
Segundo numero: 10
```

Saída

```
Média aritmética: 7
```

Exercício 1.21:

Escreva um programa que peça para o usuário fornecer um número inteiro. O programa deve calcular exibir o sucessor e o antecessor desse número. Armazene ambos os números em variáveis.

Arquivo com a solução: Exercicio1\$21.java

Entrada

Forneca um numero inteiro: 1992

Saída

Sucessor de 1992: 1993

Antecessor de 1992: 1991

Exercício 1.22:

Escreva um programa que peça para o usuário fornecer o valor de um produto. O programa deve calcular e exibir o preço de venda do produto, com um desconto de 9%, usando duas casas decimais. Armazene o preço de venda do produto em uma variável. Lembre-se que para imprimir um símbolo de porcentagem (%) na saída ao se usar o método printf, você deve precedê-lo de outro símbolo de porcentagem.

Arquivo com a solução: Exercicio1\$22.java

Entrada

Valor do produto: 5.79

Saída

Preco de venda com 9% de desconto: 5.27

Exercício 1.23:

Escreva um programa que peça para o usuário fornecer o ano de seu nascimento e o ano atual. O programa deve calcular e exibir a idade atual aproximada do usuário.

Arquivo com a solução: Exercicio1\$23.java

Entrada

Ano de nascimento: 1985

Ano atual: 2018

Saída

Idade aproximada: 33 anos

Exercício 1.24:

Escreva um programa que calcule e exiba, usando duas casas decimais, o valor líquido do salário de um professor. O programa deve pedir para o usuário fornecer o valor da hora/aula, a quantidade de aulas e a porcentagem de desconto do INSS.

Arquivo com a solução: Exercicio1\$24.java

Entrada

Valor da hora/aula: 20.78
Quantidade de aulas: 40
Porcentagem de desconto do INSS: 26.5

Saída

Salario Liquido: 610.93

Exercício 1.25:

Escreva um programa que peça para o usuário fornecer uma temperatura em graus Fahrenheit. O programa deve calcular a temperatura correspondente em graus Celsius. O resultado do cálculo deve ser armazenado em uma variável. No final, o programa deve exibir ao usuário o valor obtido. Lembrando que:

- $C = \frac{F - 32}{1,8}$
- Onde:
 - C é a temperatura em graus Celsius;
 - F é a temperatura em graus Fahrenheit.

Arquivo com a solução: Exercicio1\$25.java

Entrada

Temperatura em graus Fahrenheit: 125

Saída

125.00 graus Fahrenheit correspondem a 51.67 graus Celsius

Exercício 1.26:

Escreva um programa que peça para o usuário fornecer uma temperatura em graus Celsius. O programa deve calcular a temperatura correspondente em graus Fahrenheit. O resultado do cálculo deve ser armazenado em uma variável. No final, o programa deve exibir ao usuário o valor obtido. Lembrando que:

- $F = 1,8C + 32$
- Onde:
 - F é a temperatura em graus Fahrenheit;
 - C é a temperatura em graus Celsius.

Arquivo com a solução: Exercicio1\$26.java

Entrada

Temperatura em graus Celsius: 36

Saída

36.00 graus Celsius correspondem a 96.80 graus Fahrenheit

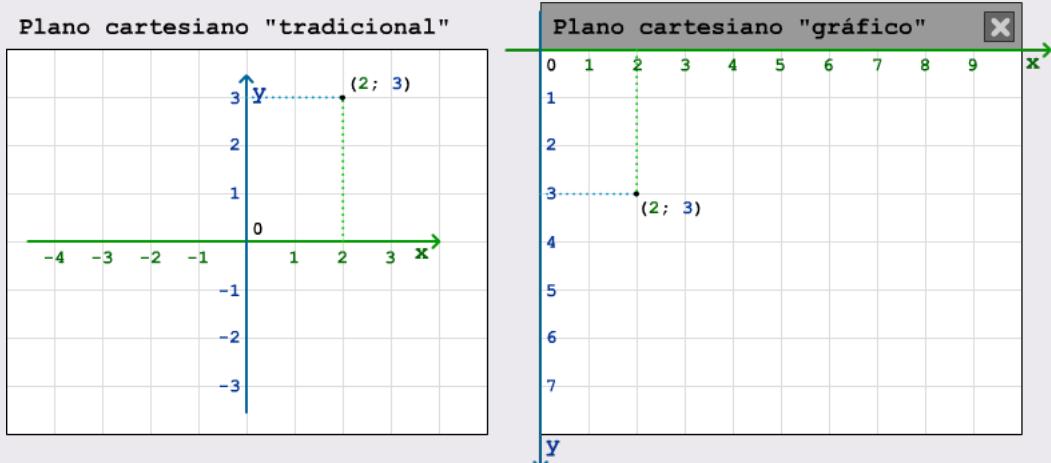
Espero que sua experiência no desenvolvimento das atividades esteja sendo prazerosa. Caso você seja aluno de algum curso técnico ou superior, a carreira que você escolheu na informática/computação para seu futuro será baseada nesse tipo de atividade. Vamos continuar!

1.3 Exercícios Criativos

Nesta edição do livro foram inseridos diversos Exercícios Criativos. Nesses exercícios você terá a oportunidade de aprender alguns conceitos relativos ao processo de “desenho” em interfaces gráficas. Para isso, usaremos uma abstração baseada em um subconjunto da Engine de desenvolvimento de jogos Raylib. Essa abstração, denominada JSimple Game Engine (JSGE²), foi implementada tendo como base a API Java 2D, uma coleção de classes e métodos que permitem ao programador realizar tais desenhos em Java. Sendo assim, preparei um modelo de projeto que você poderá utilizar na execução de tais exercícios. O repositório desse modelo pode ser acessado aqui: <<https://github.com/davidbuzatto/ModeloExerciciosCriativosJSGE>>. Basta baixar todo o repositório, editar o arquivo `Main.java`, compilar e executar a classe.

Antes de começarmos vale a pena entender um conceito muito importante quando estamos lidando com desenhos em interfaces gráficas. De modo geral, o sistema de coordenadas cartesianas é praticamente igual ao que você já está acostumado na matemática. Há duas diferenças primordiais quando estivermos falando em um sistema de duas dimensões. A primeira dessas diferenças é em relação ao eixo y . Enquanto no plano cartesiano tradicional os valores de y crescem para cima, no plano cartesiano dos contextos gráficos o eixo y cresce para baixo. A outra diferença é que a origem (encontro dos eixos x e y) do plano cartesiano padrão é normalmente apresentada no “centro”, enquanto no plano cartesiano gráfico ela está sempre inicialmente situada no canto superior esquerdo. Veja o esquema abaixo.

Planos Cartesianos:



²<<https://github.com/davidbuzatto/JSGE>>

Exercício Criativo 1.1:

Escreva um programa que peça para o usuário fornecer um par ordenado de inteiros, que representa um ponto do plano cartesiano do sistema de coordenadas gráficas. Seu programa deve pintar esse pixel usando a cor que você desejar, em uma tela de cor branca ou preta de 600 pixels de largura por 400 pixels de altura. Para executar essa tarefa utilize o método `drawPixel`, detalhado abaixo:

Método:

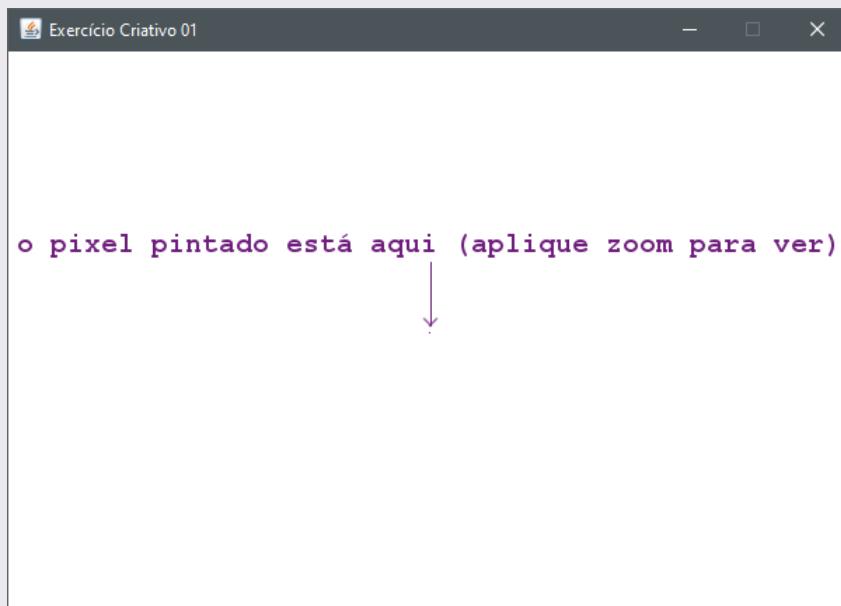
```
1 void drawPixel( double x, double y, Paint paint );
```

- **Nome:** `drawPixel`
- **Descrição:** Pinta um pixel no ponto `(x; y)` usando o `paint` especificado.
- **Entrada/Parâmetro(s):**
 1. `x`: um número decimal que representa a coordenada x do ponto desejado;
 2. `y`: um número decimal que representa a coordenada y do ponto desejado;
 3. `paint`: `paint` para o desenho³.
- **Saída/Retorno:** Esse método não retorna nenhum valor, pois é `void`⁴.

Entrada

```
x: 300  
y: 200
```

Saída:



³Um `paint` pode ser uma cor (tipo `Color`).

⁴Aprenderemos mais sobre isso no decorrer do curso.

Exercício Criativo 1.2:

Escreva um programa que peça para o usuário fornecer dois pares ordenados de inteiros que representam, respectivamente, o vértice inicial e o vértice final de um segmento de reta. Seu programa deve desenhar essa linha usando a cor que você desejar, em uma tela de cor branca ou preta de 600 pixels de largura por 400 pixels de altura. Para executar essa tarefa utilize o método `drawLine`, detalhado abaixo:

Método:

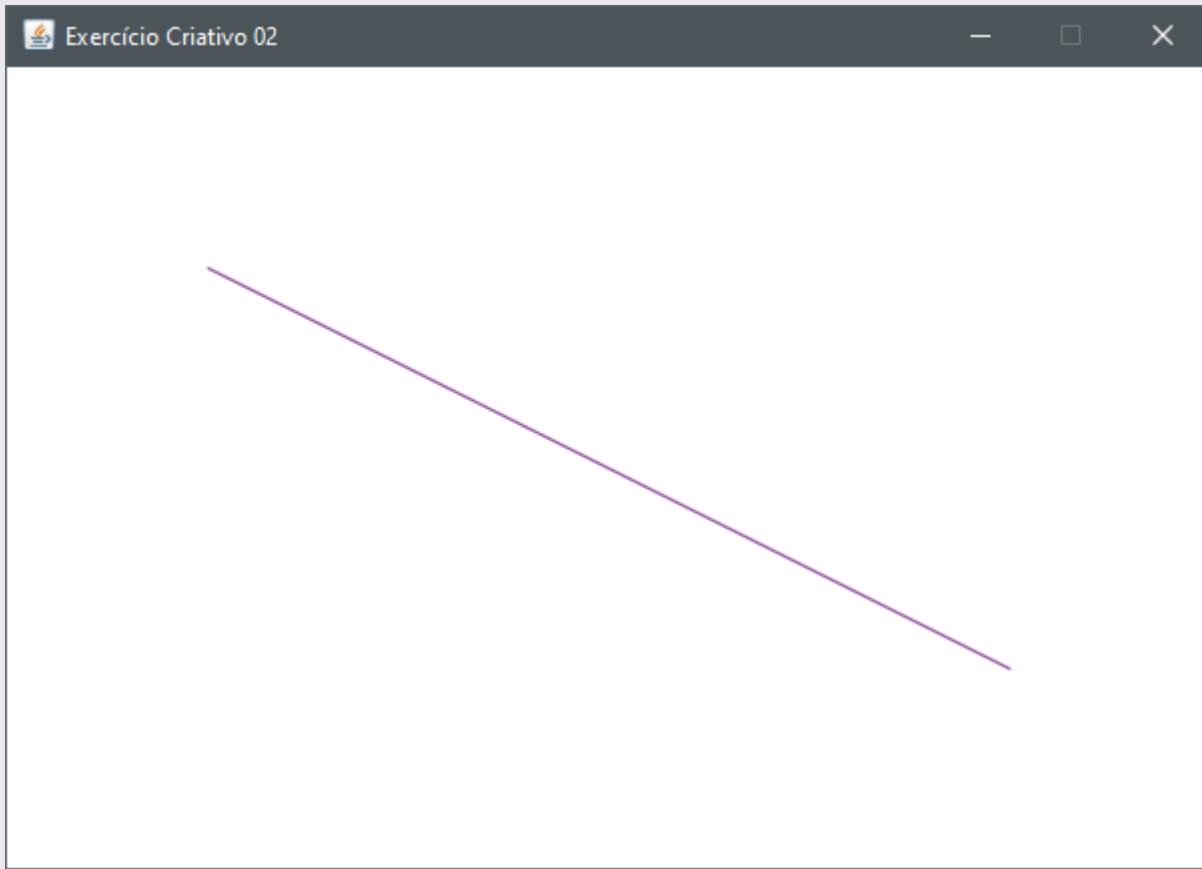
```
1 void drawLine( double startX, double startY,  
2                 double endX, double endY, Paint paint );
```

- **Nome:** `drawLine`
- **Descrição:** Desenha uma linha do ponto `(startX; startY)` até o ponto `(endX; endY)` usando o `paint` especificado.
- **Entrada/Parâmetro(s):**
 1. `startX`: um número decimal que representa a coordenada *x* do ponto inicial;
 2. `startY`: um número decimal que representa a coordenada *y* do ponto inicial;
 3. `endX`: um número decimal que representa a coordenada *x* do ponto final;
 4. `endY`: um número decimal que representa a coordenada *y* do ponto final;
 5. `paint`: `paint` para o desenho.
- **Saída/Retorno:** Esse método não retorna nenhum valor, pois é `void`.

Entrada

```
x inicial: 100  
y inicial: 100  
x final: 500  
y final: 300
```

Saída:

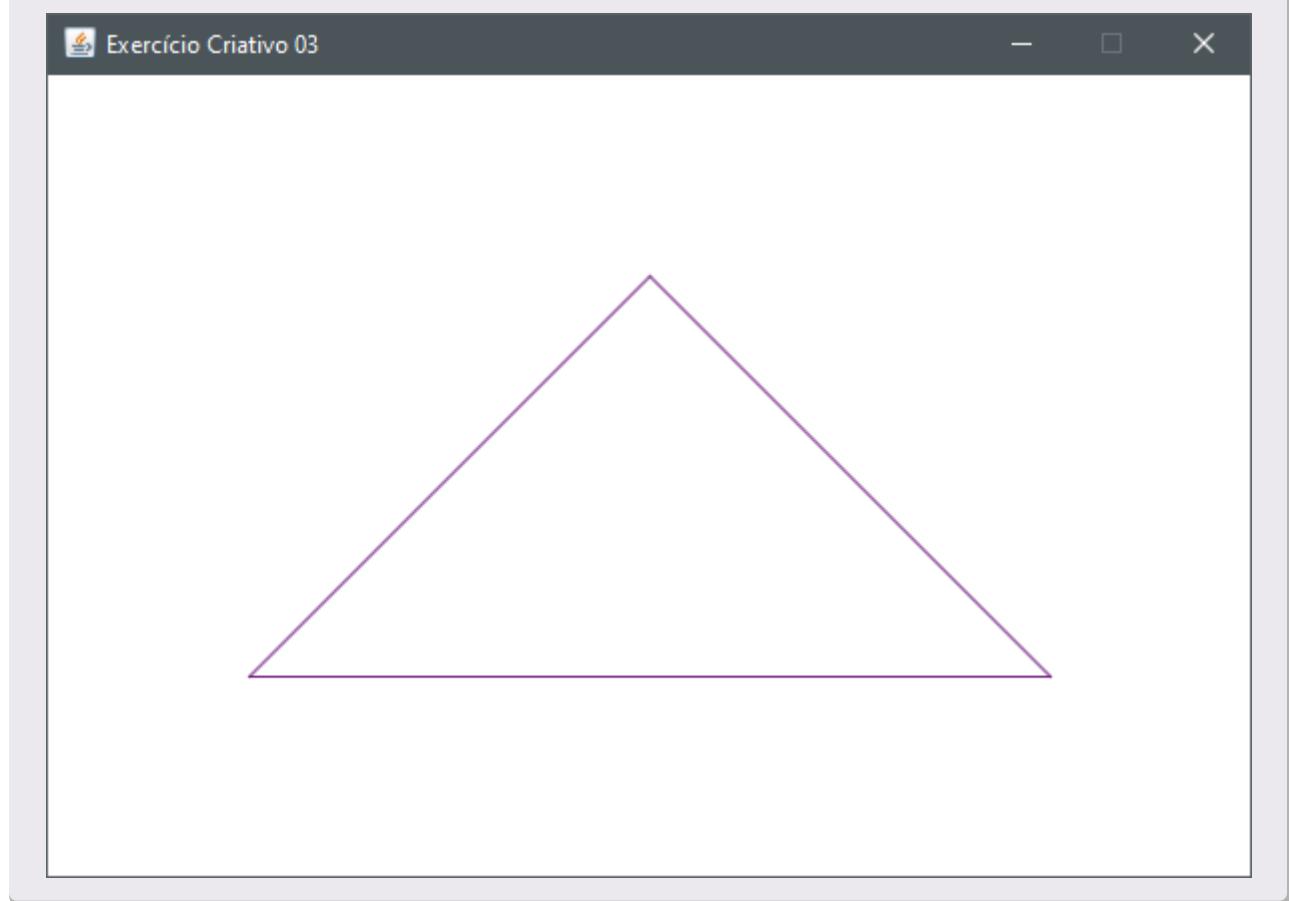


Exercício Criativo 1.3:

Escreva um programa que peça para o usuário fornecer dois pares ordenados de inteiros que representam o segmento de reta que forma a base de um triângulo isósceles ou equilátero. Note que o componente y de ambos os vértices precisa ser igual. Você ainda não sabe como realizar essa restrição, então assuma que o valor fornecido sempre será correto. Além disso, seu programa deve pedir ao usuário o valor da altura, também inteiro, do triângulo. Com esses dados, desenhe o contorno desse triângulo usando a cor que você desejar, em uma tela de cor branca ou preta de 600 pixels de largura por 400 pixels de altura. Para executar essa tarefa utilize o método `drawLine`, apresentado no exercício anterior.

Entrada

```
x inicial: 100
y inicial: 300
x final: 500
y final: 300
altura: 200
```

Saída:

Exercício Criativo 1.4:

Escreva um programa que peça para o usuário fornecer um par ordenado de inteiros que representa o vértice superior esquerdo de um retângulo, além de sua largura e altura, também inteiros. Seu programa deve desenhar o contorno desse retângulo usando a cor que você desejar, em uma tela de cor branca ou preta de 600 pixels de largura por 400 pixels de altura. Para executar essa tarefa utilize o método `drawRectangle`, detalhado abaixo:

Método:

```

1 void drawRectangle( double x, double y,
2                     double width, double height,
3                     Paint paint );

```

- **Nome:** `drawRectangle`
- **Descrição:** Desenha o contorno de um retângulo com o vértice superior esquerdo no ponto $(x; y)$, de largura igual a `width` e altura igual a `height`, usando o `paint` especificado.
- **Entrada/Parâmetro(s):**
 1. `x`: um número decimal que representa a coordenada x do vértice superior esquerdo do retângulo;
 2. `y`: um número decimal que representa a coordenada y do vértice superior esquerdo do retângulo;
 3. `width`: a largura do retângulo;
 4. `height`: a altura do retângulo;
 5. `paint`: `paint` para o desenho.
- **Saída/Retorno:** Esse método não retorna nenhum valor, pois é `void`.

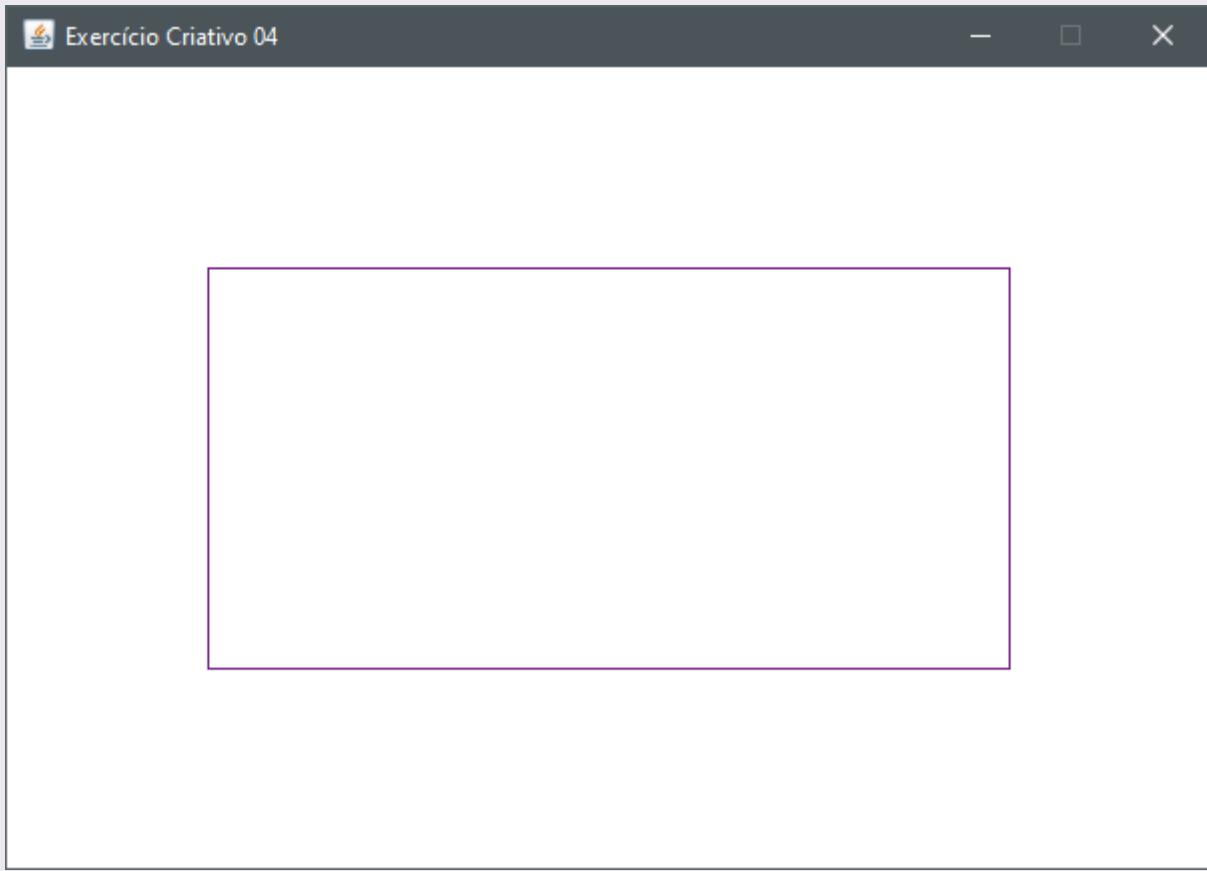
Entrada

```

x: 100
y: 100
largura: 400
altura: 200

```

Saída:



Exercício Criativo 1.5:

Escreva um programa que peça para o usuário fornecer um par ordenado de inteiros que representa o vértice superior esquerdo de um retângulo, além de sua largura e altura, também inteiros. Seu programa deve pintar esse retângulo usando a cor que você desejar, em uma tela de cor branca ou preta de 600 pixels de largura por 400 pixels de altura. Para executar essa tarefa utilize o método `fillRectangle`, detalhado abaixo:

Método:

```

1 void fillRectangle( double x, double y,
2                     double width, double height,
3                     Paint paint );

```

- **Nome:** `fillRectangle`
- **Descrição:** Desenha um retângulo com o vértice superior esquerdo no ponto $(x; y)$, de largura igual a `width` e altura igual a `height`, usando o `paint` especificado para pintar seu interior.
- **Entrada/Parâmetro(s):**
 1. `x`: um número decimal que representa a coordenada x do vértice superior esquerdo do retângulo;
 2. `y`: um número decimal que representa a coordenada y do vértice superior esquerdo do retângulo;
 3. `width`: a largura do retângulo;
 4. `height`: a altura do retângulo;
 5. `paint`: `paint` para o desenho.
- **Saída/Retorno:** Esse método não retorna nenhum valor, pois é `void`.

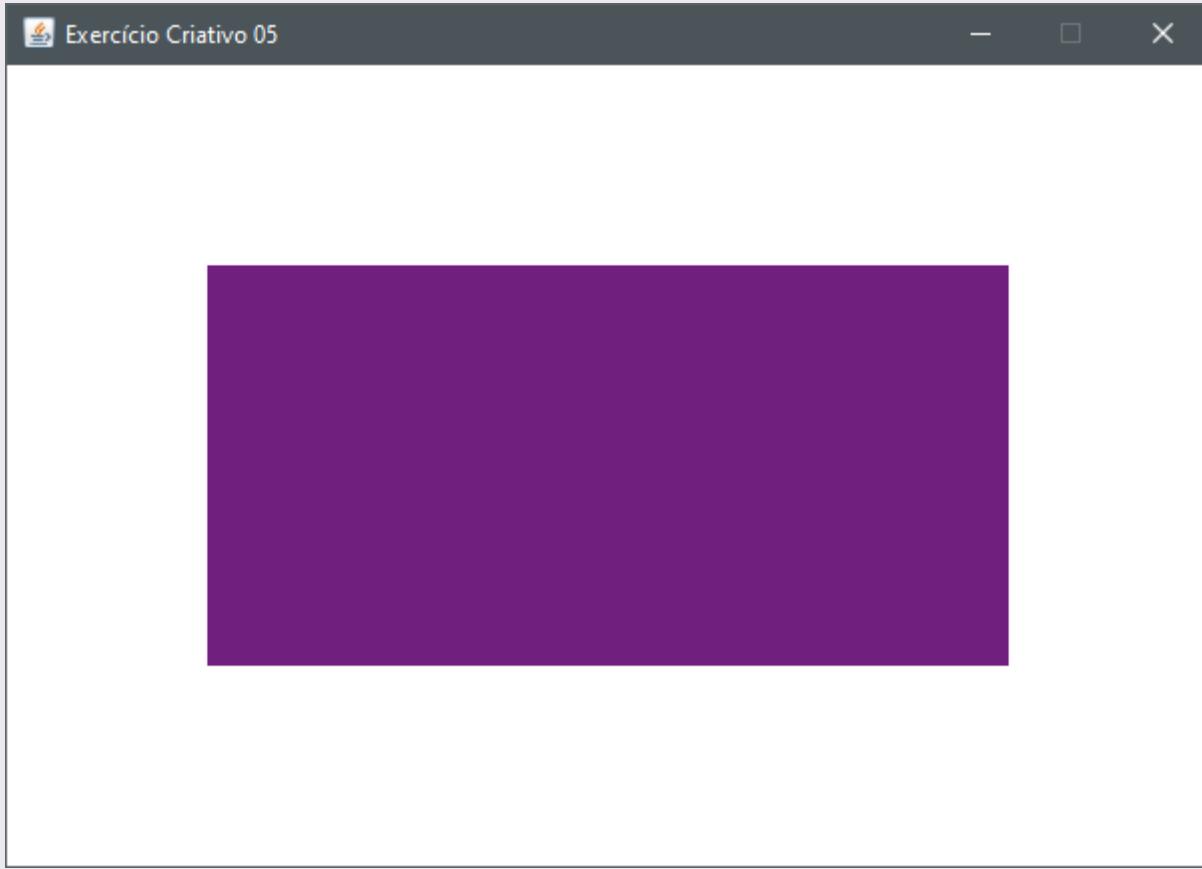
Entrada

```

x: 100
y: 100
largura: 400
altura: 200

```

Saída:



Exercício Criativo 1.6:

Escreva um programa que peça para o usuário fornecer um par ordenado de inteiros que representa o centro de uma circunferência, além de seu raio que também é do tipo inteiro. Seu programa deve desenhar essa circunferência usando a cor que você desejar, em uma tela de cor branca ou preta de 600 pixels de largura por 400 pixels de altura. Para executar essa tarefa utilize o método `drawCircle`, detalhado abaixo:

Método:

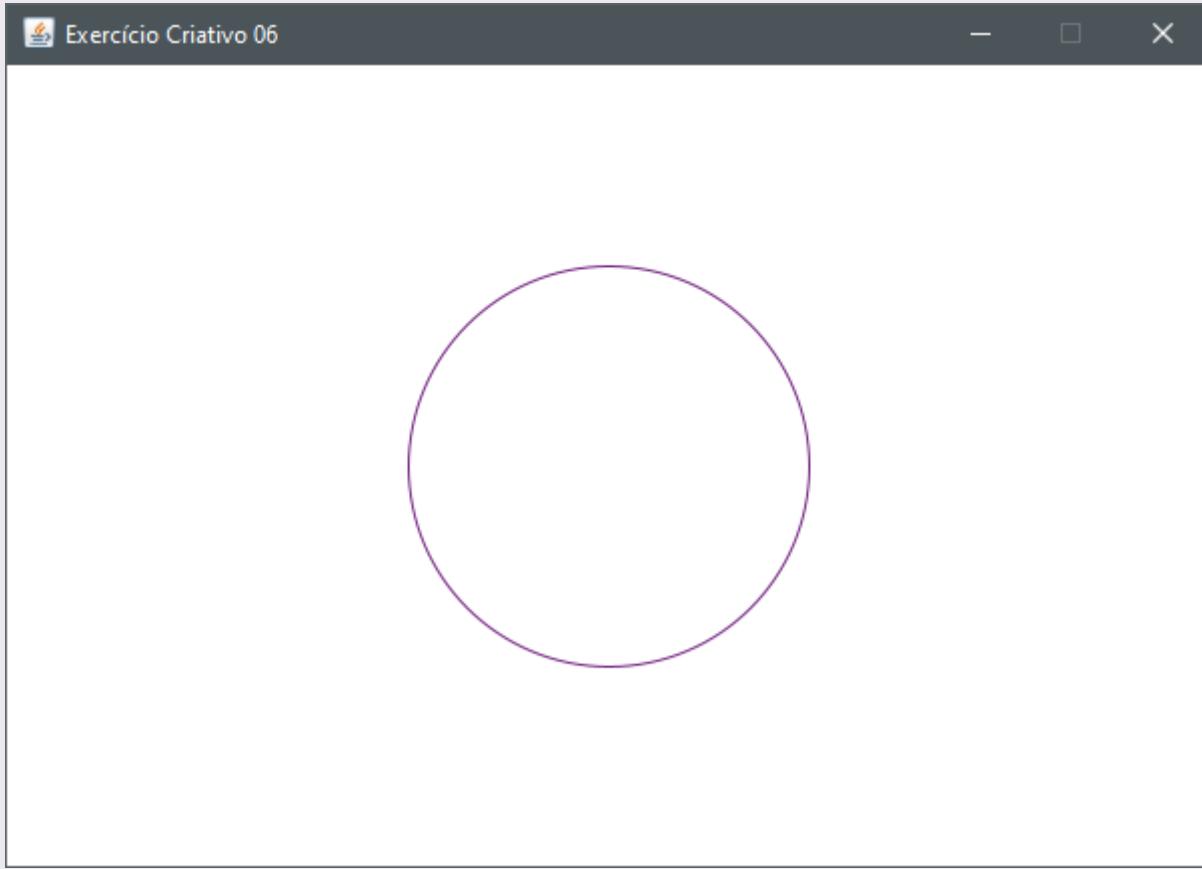
```
1 void drawCircle( double x, double y, double radius, Paint paint );
```

- **Nome:** `drawCircle`
- **Descrição:** Desenha uma circunferência (contorno de um círculo) com o centro no ponto $(x; y)$ e de raio igual a `radius`, usando o `paint` especificado.
- **Entrada/Parâmetro(s):**
 1. `x`: um número decimal que representa a coordenada x do centro da circunferência;
 2. `y`: um número decimal que representa a coordenada y do centro da circunferência;
 3. `radius`: raio da circunferência;
 4. `paint`: `paint` para o desenho.
- **Saída/Retorno:** Esse método não retorna nenhum valor, pois é `void`.

Entrada

```
x centro: 300  
y centro: 200  
raio: 100
```

Saída:



Exercício Criativo 1.7:

Escreva um programa que peça para o usuário fornecer um par ordenado de inteiros que representa o centro de um círculo, além de seu raio que também é do tipo inteiro. Seu programa deve desenhar esse círculo usando a cor que você desejar, em uma tela de cor branca ou preta de 600 pixels de largura por 400 pixels de altura. Para executar essa tarefa utilize o método `fillCircle`, detalhado abaixo:

Método:

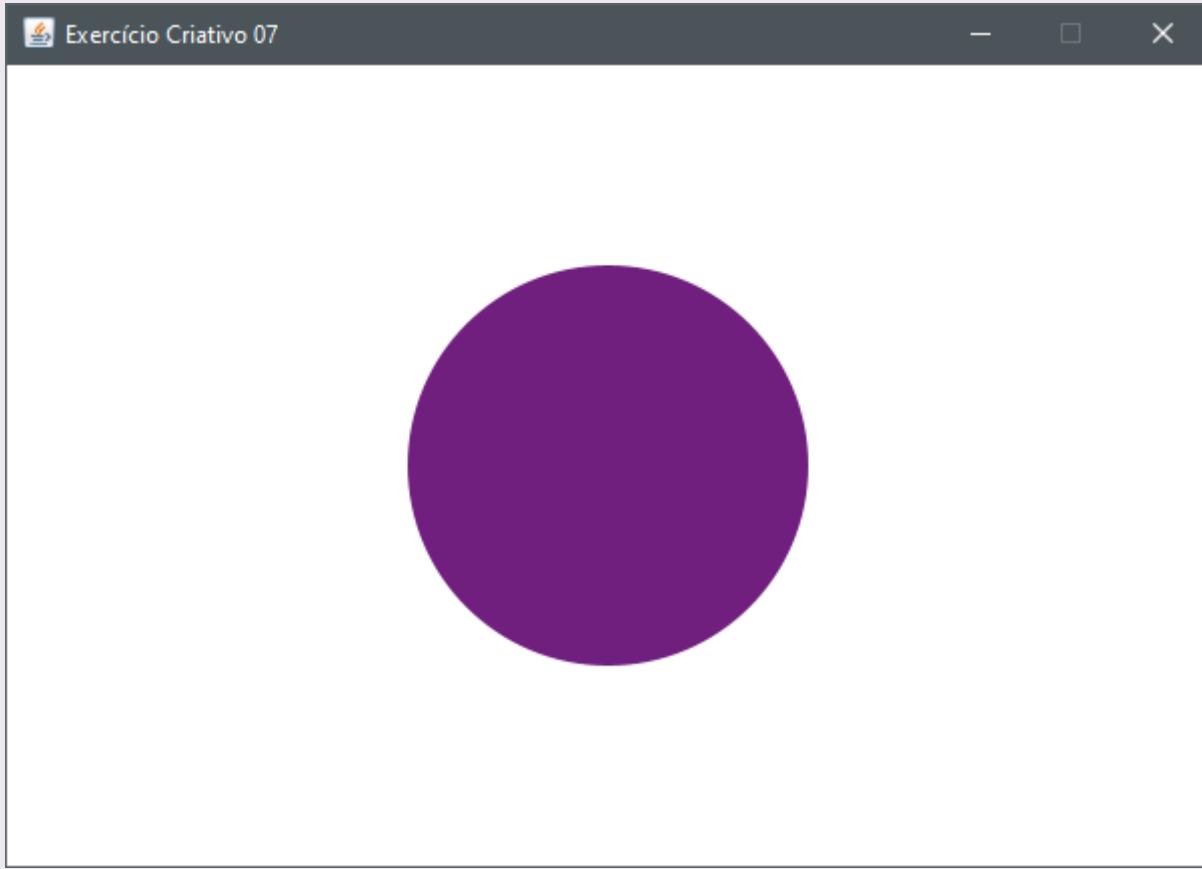
```
1 void fillCircle( double x, double y, double radius, Paint paint );
```

- **Nome:** `fillCircle`
- **Descrição:** Desenha um círculo com o centro no ponto `(x; y)` e de raio igual a `radius`, usando o `paint` especificado.
- **Entrada/Parâmetro(s):**
 1. `x`: um número decimal que representa a coordenada `x` do centro do círculo;
 2. `y`: um número decimal que representa a coordenada `y` do centro do círculo;
 3. `radius`: raio do círculo;
 4. `paint`: `paint` para o desenho.
- **Saída/Retorno:** Esse método não retorna nenhum valor, pois é `void`.

Entrada

```
x centro: 300  
y centro: 200  
raio: 100
```

Saída:



Exercício Criativo 1.8:

Escreva um programa que peça para o usuário fornecer um par ordenado de inteiros que representa o centro de uma elipse, além de seus raios horizontal e vertical, também inteiros. Seu programa deve desenhar o contorno dessa elipse usando a cor que você desejar, em uma tela de cor branca ou preta de 600 pixels de largura por 400 pixels de altura. Para executar essa tarefa utilize o método `drawEllipse`, detalhado abaixo:

Método:

```

1 void drawEllipse( double x, double y,
2                   double radiusH, double radiusV,
3                   Paint paint );

```

- **Nome:** `drawEllipse`
- **Descrição:** Desenha o contorno de uma elipse com o centro em `(x; y)`, de raio horizontal igual a `radiusH` e raio vertical igual a `radiusV`, usando o `paint` especificado.
- **Entrada/Parâmetro(s):**
 1. `x`: um número decimal que representa a coordenada `x` do centro da elipse;
 2. `y`: um número decimal que representa a coordenada `y` do centro da elipse;
 3. `radiusH`: raio horizontal da elipse;
 4. `radiusV`: raio vertical da elipse;
 5. `paint`: `paint` para o desenho.
- **Saída/Retorno:** Esse método não retorna nenhum valor, pois é `void`.

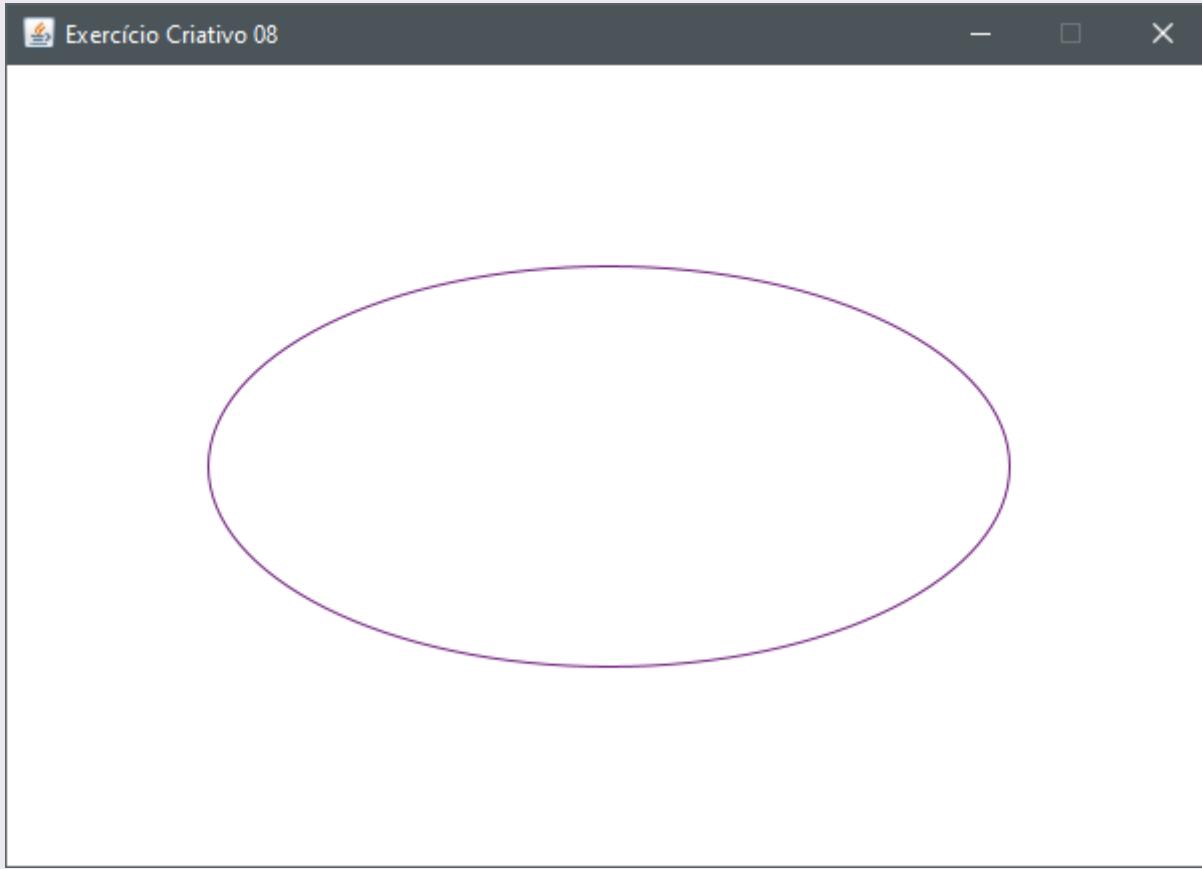
Entrada

```

x centro: 300
y centro: 200
raio horizontal: 200
raio vertical: 100

```

Saída:



Exercício Criativo 1.9:

Escreva um programa que peça para o usuário fornecer um par ordenado de inteiros que representa o centro de uma elipse, além de seus raios horizontal e vertical, também inteiros. Seu programa deve pintar internamente essa elipse usando a cor que você desejar, em uma tela de cor branca ou preta de 600 pixels de largura por 400 pixels de altura. Para executar essa tarefa utilize o método `fillEllipse`, detalhado abaixo:

Método:

```

1 void fillEllipse( double x, double y,
2                   double radiusH, double radiusV,
3                   Paint paint );

```

- **Nome:** `fillEllipse`
- **Descrição:** Desenha uma elipse com o centro no ponto `(x; y)`, de raio horizontal igual a `radiusH` e raio vertical igual a `radiusV`, usando o `paint` especificado para pintar seu interior.
- **Entrada/Parâmetro(s):**
 1. `x`: um número decimal que representa a coordenada `x` do centro da elipse;
 2. `y`: um número decimal que representa a coordenada `y` do centro da elipse;
 3. `radiusH`: raio horizontal da elipse;
 4. `radiusV`: raio vertical da elipse;
 5. `paint`: `paint` para o desenho.
- **Saída/Retorno:** Esse método não retorna nenhum valor, pois é `void`.

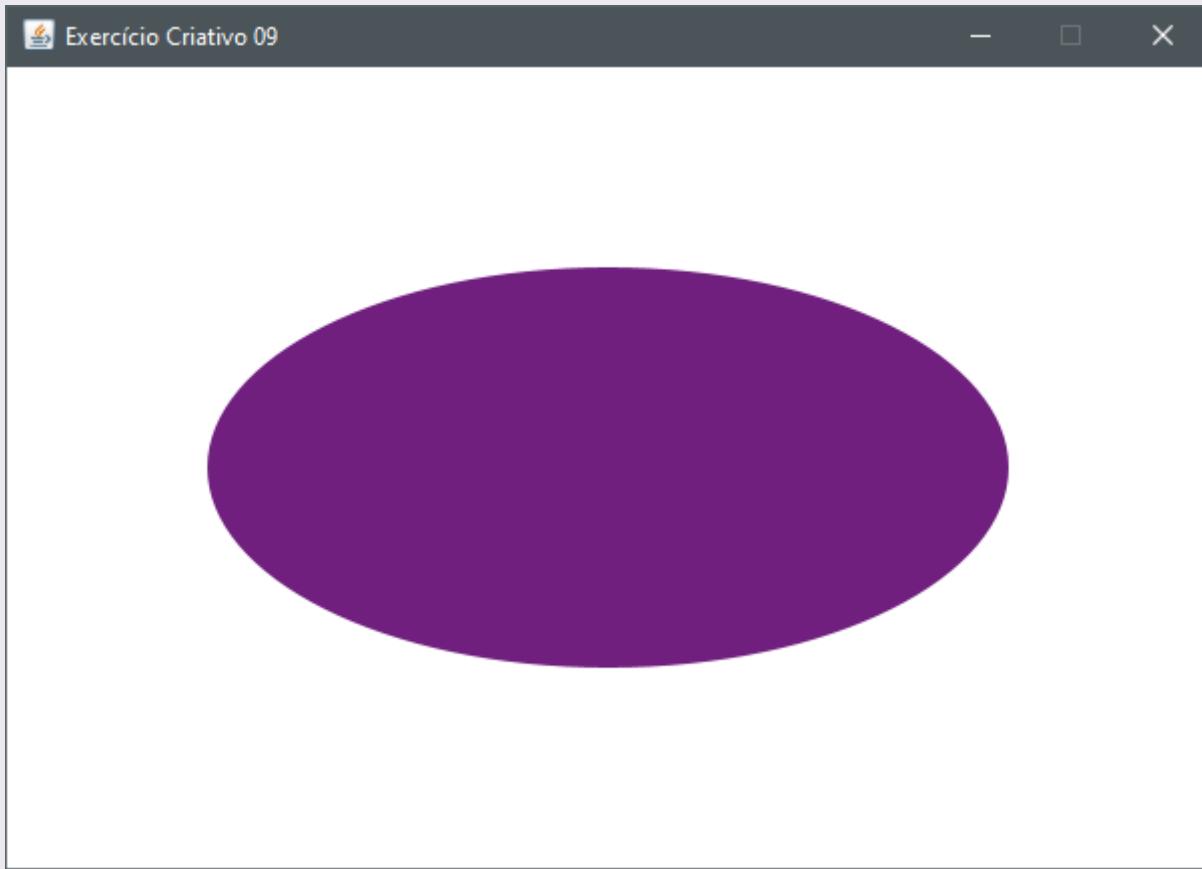
Entrada

```

x centro: 300
y centro: 200
raio horizontal: 200
raio vertical: 100

```

Saída:



ESTRUTURAS CONDICIONAIS

“Que não daria eu, para voltar a mocidade? Aceitaria qualquer condição, menos, é claro, fazer exercícios, acordar cedo e tornar-me respeitável”.

Oscar Wilde



S estruturas condicionais, ou de seleção/decisão, permitem que, a partir da avaliação de uma expressão que gera um valor lógico, ou seja, um valor verdadeiro ou falso, o fluxo de execução do programa seja alterado. Elas têm papel fundamental na construção de algoritmos e, por consequência, na escrita de programas. Neste Capítulo são apresentadas as estruturas condicionais `if` e `switch` que são utilizadas para esse objetivo.

2.1 Estrutura Condicional `if` (se)

Para entendermos melhor o funcionamento das estruturas condicionais iremos fazer um paralelo da sua sintaxe (forma de escrever), com sua execução por meio de diagramas de transição de estados que chamaremos de diagramas de fluxo. Esses diagramas são parecidos com os fluxogramas. Nos próximos três trechos de código você verá números entre parênteses, por exemplo, (1), que são análogos aos números contidos dentro dos estados, representados por círculos, nos diagramas.

| Saiba Mais

Caso queira saber mais sobre fluxogramas, o artigo da Wikipedia sobre o assunto é um bom começo. Veremos muitos tipos de diagramas durante o curso e eles normalmente são fáceis de entender, pois são bastante intuitivos. O link para o artigo sobre fluxogramas é [<https://pt.wikipedia.org/wiki/Fluxograma>](https://pt.wikipedia.org/wiki/Fluxograma).

2.1.1 Exemplo e Diagrama de Fluxo da Estrutura Condicional *if*

A forma mais simples de usarmos a estrutura condicional *if* é utilizá-la para executar algum trecho de código apenas se a condição que ela verifica (*teste*) é verdadeira.

Estrutura do *if* - Verifique a Figura 2.1

```
1 // antes
2 (1)
3     (2)
4 if ( teste ) {
5     (3)
6 }
7 (4)
8 // depois
```

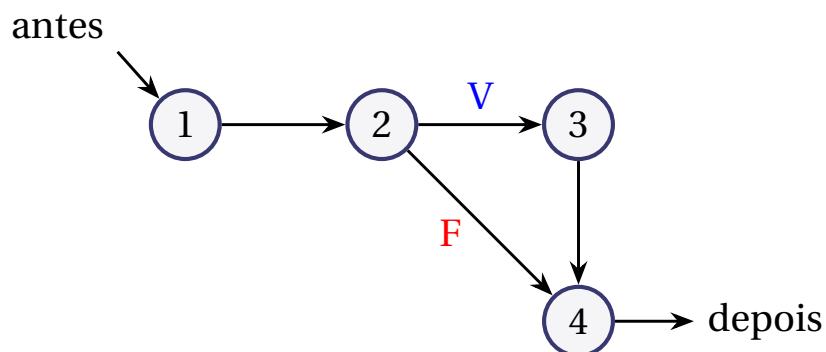


Figura 2.1: Fluxo de execução da estrutura condicional *if*

2.1.2 Exemplo e Diagrama de Fluxo da Estrutura Condicional *if...else* (*se...senão*)

Outra forma de utilizarmos a estrutura condicional *if* é associar também à ela um trecho de código que será executado caso sua condição seja avaliada como falsa. Ou seja, temos um bloco de código que executa caso a condição seja verdadeira e um caso ela seja falsa somente.

Estrutura do *if...else* - Verifique a Figura 2.2

```

1 // antes
2 (1)
3     (2)
4 if ( teste ) {
5     (3)
6 } else {
7     (4)
8 }
9 (5)
10 // depois

```

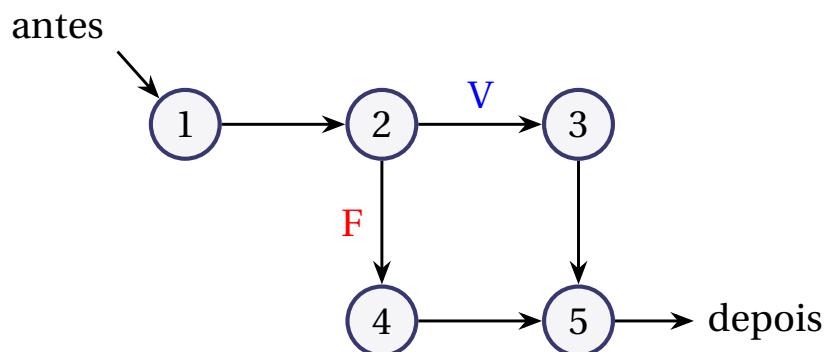


Figura 2.2: Fluxo de execução da estrutura condicional *if...else*

2.1.3 Exemplo e Diagrama de Fluxo da Estrutura Condicional *if...else if...else* (*se...senão se...senão*)

Por fim, também podemos realizar o encadeamento de diversas estruturas *if*, permitindo que possamos testar condições que dependem que outras tenham sido avaliadas como falsas previamente para que sejam executadas.

Estrutura do *if...else if...else* - Verifique a Figura 2.3

```

1 (1) // antes
2      (2)
3 if ( teste1 ) {
4     (3)
5         (4)
6 } else if ( teste2 ) {
7     (5)
8         (6)
9 } else if ( teste3 ) {
10    (7)
11 } else {
12     (8)
13 }
14 (9) // depois

```

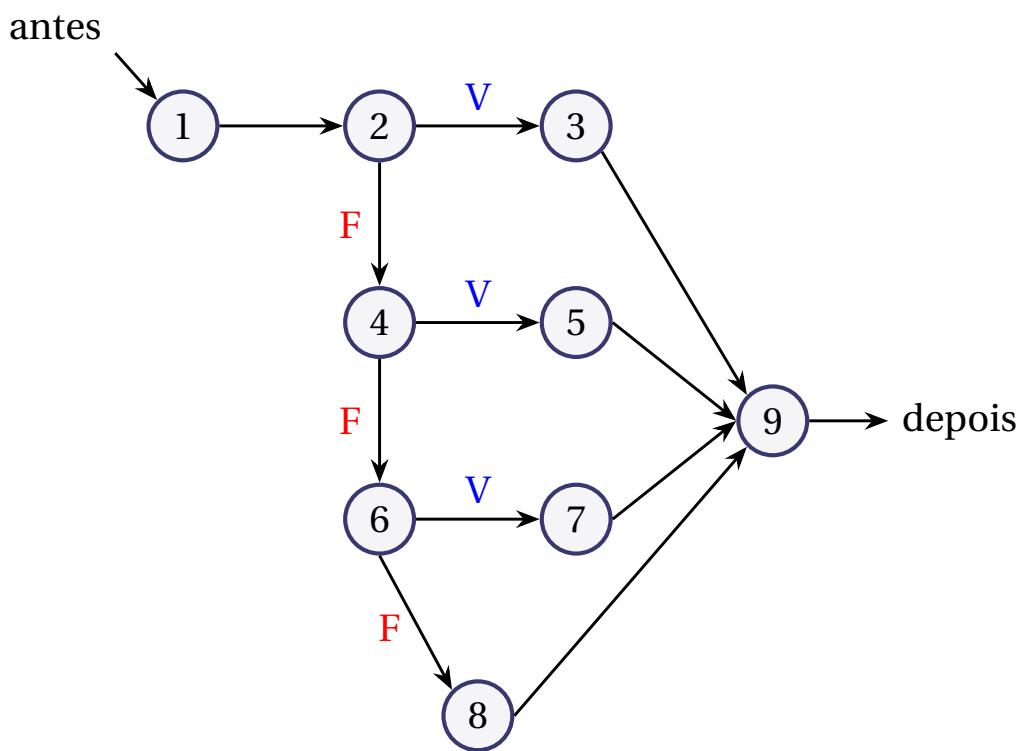


Figura 2.3: Fluxo de execução da estrutura condicional *if...else if...else*

Veja ainda que os símbolos de “abre chave” e “fecha chave” `{ e }` que delimitam os blocos de código da estrutura condicional *if*, bem como de outras estruturas que veremos mais adiante, são de uso opcional caso o bloco de código contenha apenas uma linha, mas...

💡 | Boa Prática

Independentemente da opção de usar ou não chaves para delimitar blocos com apenas uma linha de código é uma boa prática **sempre** utilizar chaves para delimitar blocos de uma linha, evitando assim possíveis erros lógicos.

2.1.4 Operadores de Igualdade, Relacionais e Lógicos

A construção de testes lógicos nas linguagens de programação é feita usando alguns conjuntos de operadores binários, ou seja, que atuam sobre dois operandos. Os operadores de igualdade verificam a igualdade ou a desigualdade entre dois valores/operandos. O operador de igualdade “igual a” da linguagem Java `==` retorna **verdadeiro** como resultado caso dois valores sejam iguais ou **falso** caso não sejam. De modo inverso, o operador de igualdade “diferente de” da linguagem Java `!=` retorna **verdadeiro** caso os dois valores avaliados sejam diferentes ou **falso** caso sejam iguais. Na Tabela 2.1 esses operadores são apresentados.

Operador	Significado
<code>==</code>	Igual a
<code>!=</code>	Diferente de

Tabela 2.1: Operadores de igualdade

Além dos operadores de igualdade, podemos também estabelecer relações entre os operandos. Para isso usamos os operadores relacionais, apresentados na Tabela 2.2. Esses operadores atuam de forma análoga aos operadores de igualdade. Por exemplo, o operador relacional “menor que” da linguagem Java `<` retorna **verdadeiro** caso o primeiro operando seja estritamente menor que o segundo operando ou **falso** caso contrário.

Operador	Significado
<code><</code>	Menor que
<code><=</code>	Menor ou igual a
<code>></code>	Maior que
<code>>=</code>	Maior ou igual a

Tabela 2.2: Operadores relacionais

Os conjuntos de operadores apresentados anteriormente lidam com quaisquer tipos de valores que podem ser comparados entre si. Já os operadores lógicos, apresentados na Tabela 2.3, lidam com operandos que têm valores lógicos, ou seja, **verdadeiro** ou **falso**. As tabelas verdade desses operadores podem ser vistas na Tabela 2.4a, na Tabela 2.4b e na Tabela 2.4c.

Operador	Significado
<code>&&</code>	E
<code> </code>	OU
<code>!</code>	NÃO

Tabela 2.3: Operadores lógicos de curto-circuito

E (<code>&&</code>)		
	V	F
V	V	F
F	F	F

(a) Operador E

OU (<code> </code>)		
	V	F
V	V	V
F	V	F

(b) Operador OU

NÃO (<code>!</code>)	
V	F
F	V

(c) Operador NÃO

Tabela 2.4: Tabelas verdade dos operadores lógicos E, OU e NÃO**⚠ | Atenção!**

Cuidado ao criar expressões que testam a igualdade dentre dois operandos. O operador de igualdade “igual a” é representado por dois sinais de igual juntos `==`, enquanto o operador de atribuição, na linguagem Java, é representado por apenas um sinal de igual `=`. Usar o operador de atribuição ao invés do operador “igual a”, onde se espera um resultado lógico, é um erro sintático que gerará um erro de compilação.

Para exemplificar tal problema, veja o exemplo abaixo:

Erro de sintaxe usando operador de atribuição	
1	<code>int a = 1;</code>
2	<code>int b = 2;</code>
3	
4	<code>// falso</code>
5	<code>if (a == b) {</code>
6	<code>System.out.println("a é igual a b");</code>
7	}
8	
9	<code>// erro de compilação</code>
10	<code>if (a = b)</code>
11	<code>System.out.println("a é igual a b");</code>
12	}

2.1.5 Operador Ternário

O operador ternário, simbolizado pelo caractere `?`, funciona de forma parecida com uma estrutura condicional `if`. Obrigatoriamente, ao utilizar o operador ternário, é necessário que seja gerado algum tipo de retorno. Veja os exemplos a seguir.

Exemplo de uso do operador ternário

```

1  /*
2   * Arquivo: capitulo02/EstruturasCondicionaisOperadorTernario.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6  public class EstruturasCondicionaisOperadorTernario {
7
8      public static void main( String[] args ) {
9
10         int n = 20;
11
12         // n é par? se sim, retorna 1, caso contrário, retorna 0
13         int v = n % 2 == 0 ? 1 : 0;
14
15         System.out.println( v );
16
17     }
18
19 }
```

Código correspondente usando a estrutura condicional if

```

1  /*
2   * Arquivo: capitulo02/EstruturasCondicionaisOperadorTernarioIf.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6  public class EstruturasCondicionaisOperadorTernarioIf {
7
8      public static void main( String[] args ) {
9
10         int n = 20;
11         int v;
12
13         // n é par? se sim, atribui 1 a v, caso contrário, atribui 0
14         if ( n % 2 == 0 ) {
15             v = 1;
16         } else {
17             v = 0;
18         }
19
20         System.out.println( v );
21
22     }
23
24 }
```

Há a possibilidade de se encadear diversos operadores ternários na mesma expressão, entretanto,

essa prática pode acarretar em dificuldade de leitura do código, sendo assim:

🔗 | Boa Prática

Utilize o operador ternário **apenas** em situações simples e fáceis de serem lidas e interpretadas, evitando assim erros de lógica em seus algoritmos.

🔗 | Boa Prática

Evite utilizar mais de um operador ternário em uma expressão, visando sempre facilitar a leitura do seu código.

Veja abaixo como **não** utilizar o operador ternário.

Como não utilizar o operador ternário

```
1  /*
2   * Arquivo: capitulo02/EstruturasCondicionaisOperadorTernarioNao.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6  public class EstruturasCondicionaisOperadorTernarioNao {
7
8      public static void main( String[] args ) {
9
10         int n = 20;
11
12         /*
13          * n é par e menor que zero? retorna 1, caso contrário, 0
14          * n é ímpar e maior que zero? retorna 1, caso contrário, 0
15          */
16         int v = n % 2 == 0 ? n < 0 ? 1 : 0 : n > 0 ? 1 : 0;
17
18         // correspondente usando if
19         if ( n % 2 == 0 ) {
20             if ( n < 0 ) {
21                 v = 1;
22             } else {
23                 v = 0;
24             }
25         } else {
26             if ( n > 0 ) {
27                 v = 1;
28             } else {
29                 v = 0;
30             }
31         }
32
33         System.out.println( v );

```

```

34
35     }
36
37 }
```

2.1.6 O tipo boolean

Na linguagem Java existe, além dos tipos `int`, `char` e `double` outros tipos primitivos de dados, entre eles o tipo `boolean`, que é um tipo lógico capaz de armazenar um valor verdadeiro (`true`) ou falso (`false`). O nome do tipo `boolean` se originou da Lógica de Boole, inventada por George Boole.

| Saiba Mais

Quer saber mais sobre George Boole e a Álgebra Booleana? Acesso os seguintes links:

- George Boole: <https://pt.wikipedia.org/wiki/George_Boole>
- Álgebra Booleana: <<https://brasilescola.uol.com.br/informatica/algebra-booleana.htm>>

Exemplo de uso do tipo boolean

```

1 /*
2  * Arquivo: capitulo02/EstruturasCondicionaisStdbool.java
3  * Autor: Prof. Dr. David Buzatto
4  */
5
6 import java.util.Scanner;
7
8 public class EstruturasCondicionaisBoolean {
9
10    public static void main( String[] args ) {
11
12        // variável do tipo boolean
13        boolean maiorDeIdade;
14        int idade;
15
16        Scanner scan = new Scanner( System.in );
17
18        System.out.print( "Entre com sua idade: " );
19        idade = Integer.parseInt( scan.nextLine() );
20
21        if ( idade < 18 ) {
22            // false é o valor que indica falso
23            maiorDeIdade = false;
24        } else {
25            // true é o valor que indica verdadeiro
26            maiorDeIdade = true;
```

```
27     }
28
29 // o if anterior poderia ser substituído por
30 maiorDeIdade = idade >= 18;
31
32 // ou
33 maiorDeIdade = !( idade < 18 );
34
35 /*
36 * maiorDeIdade armazena verdadeiro ou falso, sendo assim
37 * pode ser usado diretamente no lugar do teste lógico.
38 */
39 if ( maiorDeIdade ) {
40     System.out.println( "Voce eh maior de idade!" );
41 } else {
42     System.out.println( "Voce nao eh maior de idade!" );
43 }
44
45 scan.close();
46
47 }
48
49 }
```

Vamos aos exercícios!

2.1.7 Exercícios

Exercício 2.1:

Escreva um programa que peça para o usuário fornecer um número inteiro. O programa deve exibir se o número fornecido é par ou ímpar.

Arquivo com a solução: Exercicio2\$1.java

Entrada

Entre com um numero: 19

Saída

O numero 19 e ímpar.

Entrada

Entre com um numero: 8

Saída

O numero 8 e par.

Exercício 2.2:

Escreva um programa que peça para o usuário fornecer dois números inteiros. O programa deve exibir esses dois números em ordem crescente.

Arquivo com a solução: Exercicio2\$2.java

Entrada

```
Entre com um numero: 7  
Entre com outro numero: 2
```

Saída

```
Ordem crescente: 2 <= 7
```

Entrada

```
Entre com um numero: -2  
Entre com outro numero: 9
```

Saída

```
Ordem crescente: -2 <= 9
```

Entrada

```
Entre com um numero: 4  
Entre com outro numero: 4
```

Saída

```
Ordem crescente: 4 <= 4
```

Exercício 2.3:

Escreva um programa que peça para o usuário fornecer dois números inteiros. O programa deve exibir esses dois números em ordem decrescente.

Arquivo com a solução: Exercicio2\$3.java

Entrada

```
Entre com um numero: 7  
Entre com outro numero: 2
```

Saída

```
Ordem decrescente: 7 >= 2
```

Entrada

```
Entre com um numero: -30  
Entre com outro numero: 20
```

Saída

```
Ordem decrescente: 20 >= -30
```

Entrada

```
Entre com um numero: 4  
Entre com outro numero: 4
```

Saída

```
Ordem decrescente: 4 >= 4
```

Exercício 2.4:

Escreva um programa que peça para o usuário fornecer três números inteiros. O programa deve exibir esses três números em ordem crescente.

Arquivo com a solução: Exercicio2\$4.java

Entrada

```
N1: 5  
N2: 1  
N3: 9
```

Saída

```
1 <= 5 <= 9
```

Entrada

```
N1: 15  
N2: 8  
N3: -4
```

Saída

```
-4 <= 8 <= 15
```

Entrada

```
N1: -4  
N2: 8  
N3: -4
```

Saída

```
-4 <= -4 <= 8
```

Exercício 2.5:

Escreva um programa que peça para o usuário fornecer três números inteiros. O programa deve exibir esses três números em ordem decrescente.

Arquivo com a solução: Exercicio2\$5.java

Entrada

```
N1: 5  
N2: 1  
N3: 9
```

Saída

```
9 >= 5 >= 1
```

Entrada

```
N1: 15  
N2: 8  
N3: -4
```

Saída

```
15 >= 8 >= -4
```

Entrada

```
N1: -4  
N2: 8  
N3: -4
```

Saída

```
8 >= -4 >= -4
```

Exercício 2.6:

Escreva um programa que peça para o usuário fornecer um número decimal. Se esse número for maior que 20, imprimir sua metade, caso contrário, imprimir seu triplo. Formate a saída dos números decimais usando 2 casas de precisão.

Arquivo com a solução: Exercicio2\$6.java

Entrada

Entre com um valor: 33.5

Saída

A metade de 33.50 é 16.75

Entrada

Entre com um valor: 9.5

Saída

O triplo de 9.50 é 28.50

Exercício 2.7:

Escreva um programa que peça para o usuário fornecer dois números decimais. O programa deve somar esses dois números e se essa soma for maior que 10, os dois números devem ser exibidos. Caso contrário, a subtração dos dois números deve ser mostrada. Formate a saída dos números decimais usando 2 casas de precisão.

Arquivo com a solução: Exercicio2\$7.java

Entrada

```
Entre com um numero: 7  
Entre com outro numero: 8.5
```

Saída

```
Os numeros fornecidos foram 7.00 e 8.50
```

Entrada

```
Entre com um numero: 3  
Entre com outro numero: 2
```

Saída

```
A subtracao entre 3.00 e 2.00 e igual a 1.00
```

Exercício 2.8:

Escreva um programa que peça para o usuário fornecer três números decimais e escrever a soma dos dois maiores. Formate a saída dos números decimais usando 2 casas de precisão.

Arquivo com a solução: Exercicio2\$8.java

Entrada

```
N1: 4  
N2: 2  
N3: 9
```

Saída

```
A soma dos dois numeros maiores fornecidos e 13.00
```

Entrada

```
N1: -1  
N2: 7  
N3: -2
```

Saída

```
A soma dos dois numeros maiores fornecidos e 6.00
```

Entrada

```
N1: 7  
N2: 3  
N3: 7
```

Saída

```
A soma dos dois numeros maiores fornecidos e 14.00
```

Exercício 2.9:

Escreva um programa que peça para o usuário fornecer a quantidade de lados de um polígono regular (inteiro) e a medida do lado (decimal). Calcular e imprimir o seguinte:

- Se o número de lados for igual a 3, escrever: TRIANGULO (sem acento) e o valor do seu perímetro;
- Se o número de lados for igual a 4, escrever: QUADRADO e o valor da sua área;
- Se o número de lados for igual a 5, escrever: PENTAGONO (sem acento);
- Em qualquer outra situação, escrever: Poligono nao identificado (sem acentos).

Formate a saída dos números decimais usando 2 casas de precisão.

Arquivo com a solução: Exercicio2\$9.java

Entrada

Entre com a quantidade de lados: 3
Entre com a medida do lado: 7.5

Saída

TRIANGULO de perimetro 22.50

Entrada

Entre com a quantidade de lados: 4
Entre com a medida do lado: 5.5

Saída

QUADRADO de area 30.25

Entrada

Entre com a quantidade de lados: 5
Entre com a medida do lado: 2.5

Saída

PENTAGONO

Entrada

Entre com a quantidade de lados: 7
Entre com a medida do lado: 3.75

Saída

Poligono nao identificado

Exercício 2.10:

Escreva um programa que leia as medidas dos lados de um triângulo (decimais) e escreva se ele é EQUILATERO, ISOSCELES ou ESCALENO (sem acentos). Observação:

- Triângulo equilátero: Possui 3 lados congruentes (mesma medida);
- Triângulo isósceles: Possui 2 lados congruentes e um diferente;
- Triângulo escaleno: Possui 3 lados diferentes;
- Caso os valores fornecidos não representem um triângulo válido, apresente uma mensagem de erro.

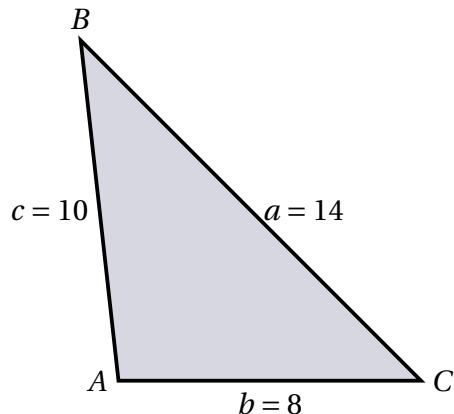
A condição de existência de um triângulo, para os lados a , b e c , é a seguinte:

$$|a - b| < c < a + b \wedge$$

$$|a - c| < b < a + c \wedge$$

$$|b - c| < a < b + c = \text{VERDADEIRO}$$

Exemplo para $a = 14$, $b = 8$, $c = 10$:



$$|14 - 8| < 10 < 14 + 8 \wedge$$

$$|14 - 10| < 8 < 14 + 10 \wedge$$

$$|8 - 10| < 14 < 8 + 10 = \text{VERDADEIRO}$$

Arquivo com a solução: Exercicio2\$10.java

Entrada

```
a: 5
b: 5
c: 5
```

Saída

```
Triangulo EQUILATERO
```

Entrada

a: 6.5
b: 9
c: 6.5

Saída

Triangulo ISOSCELES

Entrada

a: 6.5
b: 7
c: 3.5

Saída

Triangulo ESCALENO

Entrada

a: 15.8
b: 5.5
c: 3.5

Saída

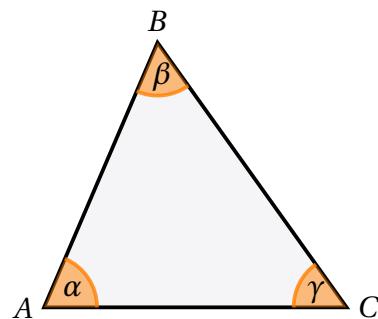
As medidas fornecidas dos lados nao representam um triangulo valido!

Exercício 2.11:

Escreva um programa que leia o valor de 3 ângulos internos (α , β e γ) (decimais) de um triângulo e escreva se o triângulo é ACUTANGULO, RETANGULO ou OBTUSANGULO (sem acentos). Observação:

- Triângulo retângulo: possui um ângulo reto (90 graus);
- Triângulo obtusângulo: possui um ângulo obtuso (ângulo maior que 90 graus);
- Triângulo acutângulo: possui 3 ângulos agudos (ângulo menor que 90 graus);
- Caso os valores fornecidos não representem um triângulo válido, apresente uma mensagem de erro.

Para ser um triângulo, a soma dos três ângulos internos deve ser igual a 180 graus, ou seja, $\alpha + \beta + \gamma = 180^\circ$.



Arquivo com a solução: Exercicio2\$11.java

Entrada

```
alfa: 90
beta: 60
gama: 30
```

Saída

```
Triangulo RETANGULO
```

Entrada

```
alfa: 70
beta: 70
gama: 40
```

Saída

```
Triangulo ACUTANGULO
```

Entrada

```
alfa: 30  
beta: 120  
gama: 30
```

Saída

```
Triangulo OBTUSANGULO
```

Entrada

```
alfa: 90  
beta: 60  
gama: 60
```

Saída

```
As medidas fornecidas dos angulos nao representam um triangulo valido!
```

Exercício 2.12:

Escreva um programa que leia a idade de dois homens e duas mulheres, todos valores inteiros. Calcule e escreva a soma das idades do homem mais velho com a mulher mais nova e o produto das idades do homem mais novo com a mulher mais velha.

Arquivo com a solução: Exercicio2\$12.java

Entrada

```
Idade Homem 1: 20  
Idade Homem 2: 25  
Idade Mulher 1: 40  
Idade Mulher 2: 15
```

Saída

```
Idade homem mais velho + idade mulher mais nova: 40  
Idade homem mais novo * idade mulher mais velha: 800
```

Exercício 2.13:

Escreva um programa que leia as notas das duas avaliações normais e a nota da avaliação optativa. Caso o aluno não tenha feito a optativa deve ser fornecido um valor negativo. Calcular a média do semestre considerando que a prova optativa substitui a nota mais baixa entre as duas primeiras avaliações, caso, é claro, ela seja maior que uma das duas notas. Escrever a média e uma mensagem que indique se o aluno foi aprovado, reprovado ou está em exame. Formate a saída dos números decimais usando 2 casas de precisão. Considere que se M é a média:

- Aprovado: $M \geq 6,0$;
- Exame: $4,0 \leq M < 6,0$;
- Reprovado: $M < 4,0$

Arquivo com a solução: Exercicio2\$13.java

Entrada

```
Nota Av. 1: 6.5  
Nota Av. 2: 7.5  
Nota Optativa: -1
```

Saída

```
Media: 7.00  
Aprovado!
```

Entrada

```
Nota Av. 1: 3  
Nota Av. 2: 4  
Nota Optativa: 6
```

Saída

```
Media: 5.00  
Exame.
```

Entrada

```
Nota Av. 1: 5  
Nota Av. 2: 1  
Nota Optativa: 2
```

Saída

```
Media: 3.50  
Reprovado...
```

Exercício 2.14:

Escreva um programa que peça para o usuário fornecer seu peso em quilogramas e sua altura em metros, ambos números decimais. O programa deve calcular o IMC (Índice de Massa Corpórea) do usuário e no final deve exibir, além do índice, qual a situação do usuário na forma de uma mensagem, sem acentos, baseando-se nas seguintes regras:

- $IMC < 17,0$: Voce esta muito abaixo do peso ideal!
- $17,0 \leq IMC < 18,5$: Voce esta abaixo do peso ideal!
- $18,5 \leq IMC < 25,0$: Parabens! Voce esta em seu peso normal!
- $25,0 \leq IMC < 30,0$: Atencao, voce esta acima de seu peso (sobrepeso)!
- $30,0 \leq IMC < 35,0$: Cuidado! Obesidade grau I!
- $35,0 \leq IMC < 40,0$: Cuidado! Obesidade grau II!
- $IMC \geq 40,0$: Muito cuidado!!! Obesidade grau III!

Para o cálculo do IMC utilize:

- $IMC = \frac{p}{h^2}$
- Onde:
 - IMC é o índice de massa corpórea;
 - p é o valor do peso (na verdade, massa) em quilogramas;
 - h é o valor da altura em metros.

Formate a saída dos números decimais usando 2 casas de precisão.

Arquivo com a solução: Exercicio2\$14.java

Entrada

```
Entre com seu peso em quilogramas: 82
Entre com sua altura em metros: 1.8
```

Saída

```
IMC: 25.31
Atencao, voce esta acima de seu peso (sobrepeso)!
```

Entrada

```
Entre com seu peso em quilogramas: 50
Entre com sua altura em metros: 1.7
```

Saída

```
IMC: 17.30
Voce esta abaixo do peso ideal!
```

Entrada

Entre com seu peso em quilogramas: 120

Entre com sua altura em metros: 1.82

Saída

IMC: 36.23

Cuidado! Obesidade grau II!

Exercício 2.15:

Escreva um programa que peça para o usuário fornecer sua idade em anos e que exiba a classe eleitoral, sem acentos, desse usuário, baseando-se nas seguintes regras:

- Idade abaixo de 16 anos: Nao eleitor;
- Idade maior ou igual a 18 anos e menor ou igual a 65 anos: Eleitor obrigatorio;
- Idade maior ou igual a 16 anos e menor que 18 anos ou maior que 65 anos: Eleitor facultativo.

Arquivo com a solução: Exercicio2\$15.java

Entrada

Entre com sua idade: 18

Saída

Eleitor obrigatorio.

Entrada

Entre com sua idade: 29

Saída

Eleitor obrigatorio.

Entrada

Entre com sua idade: 15

Saída

Nao eleitor.

Entrada

Entre com sua idade: 17

Saída

Eleitor facultativo.

Entrada

Entre com sua idade: 70

Saída

Eleitor facultativo.

Exercício 2.16:

Escreva um programa que peça para o usuário fornecer um número no intervalo de 1, inclusive, e 3999, inclusive. Caso o valor fornecido esteja fora desse intervalo, o programa deve avisar o usuário e terminar. Caso contrário, o programa deve exibir o número romano correspondente ao número arábico fornecido. Lembrando que:

- 1 = I;
- 5 = V;
- 10 = X;
- 50 = L;
- 100 = C;
- 500 = D;
- 1000 = M;

Arquivo com a solução: Exercicio2\$16.java

Entrada

Entre com um numero entre 1 e 3999: 4

Saída

4 = IV

Entrada

Entre com um numero entre 1 e 3999: 9

Saída

9 = IX

Entrada

Entre com um numero entre 1 e 3999: 27

Saída

27 = XXVII

Entrada

Entre com um numero entre 1 e 3999: 251

Saída

251 = CCLI

Entrada

Entre com um numero entre 1 e 3999: 2796

Saída

2796 = MMDCCXCVI

Entrada

Entre com um numero entre 1 e 3999: 5000

Saída

Numero incorreto!

2.2 Estrutura Condisional *switch* (escolha)

2.2.1 Exemplo e Diagrama de Fluxo da Estrutura Condicional *switch*

Estrutura do *switch* - Verifique a Figura 2.4

```

1 (1) // antes
2 switch ( valor ) {
3     case valor1: (2)
4         (3)
5         break;
6     case valor2: (4)
7         (5)
8         break;
9     case valor3: (6)
10        (7)
11        break;
12     default:
13         (8)
14         break;
15 }
16 (9) // depois

```

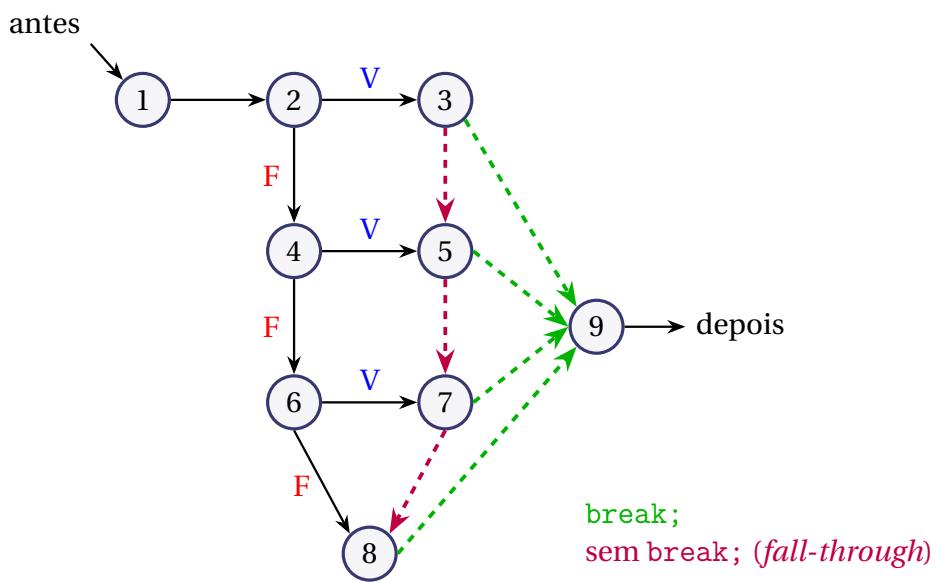


Figura 2.4: Fluxo de execução da estrutura condicional *switch*

A outra estrutura condicional que a linguagem de programação Java suporta é a *switch* (escolha). A diferença entre ela e o *if* é que na estrutura *switch* usa-se um valor, normalmente contido em uma variável, para ser comparado em cada cláusula `case` contida dentro do bloco. Caso o valor da variável passada seja igual ao valor esperado por algum dos *cases*, o bloco associado ao `case` em questão será executado. É necessário que se use a instrução `break` para que a execução de um bloco pare antes do próximo `case` (ou `default`), caso contrário, as próximas linhas de código serão executadas até encontrar um `break` ou até terminar o bloco do *switch*. Esse comportamento

é chamado de *fall-through*. Outro detalhe é que a ordem dos *cases* não importa e que a cláusula `default` é opcional, além de ser usada para casar com qualquer valor diferente dos valores esperados por todos os cases especificados. Os tipos mais utilizados nas cláusulas `case` são `int`, `char`, `String` e tipos enumerados, sendo que esses dois últimos ainda serão objeto dos nossos estudos.

Vamos aos exercícios!

2.2.2 Exercícios

Exercício 2.17:

Escreva um programa que peça para o usuário fornecer um número inteiro. Use um switch para verificar se o número é igual a 2, ou 4, ou 6, ou 8. Caso seja um desses números, exiba uma mensagem informando ao usuário o número que foi digitado. Caso não seja nenhum dos números esperados, informe o usuário que o valor inserido é inválido.

Arquivo com a solução: Exercicio2\$17.java

Entrada

Entre com um valor inteiro: 6

Saída

O valor fornecido foi 6.

Entrada

Entre com um valor inteiro: 7

Saída

Valor invalido.

Exercício 2.18:

Escreva um programa que peça para o usuário fornecer dois números decimais. Após a inserção de tais números, o programa deve mostrar ao usuário um menu, onde ele poderá escolher entre as quatro operações básicas (adição, subtração, multiplicação e divisão). Fazer a leitura dessa opção como um caractere (tipo char). Dependendo da operação escolhida, o programa deve executar o cálculo correspondente e exibir ao usuário o resultado. Caso o usuário forneça uma opção inválida, o programa deve exibir uma mensagem dizendo que a opção é inválida e deve terminar sua execução. Formate a saída dos números decimais usando 2 casas de precisão.

Arquivo com a solução: Exercicio2\$18.java

Entrada

```
N1: 28  
N2: 8  
Escolha uma operacao de acordo com o menu:  
+) Adicao;  
-) Subtracao;  
*) Multiplicacao;  
/) Divisao.  
Operacao: +
```

Saída

```
28.00 + 8.00 = 36.00
```

Entrada

```
N1: 16  
N2: 3  
Escolha uma operacao de acordo com o menu:  
+) Adicao;  
-) Subtracao;  
*) Multiplicacao;  
/) Divisao.  
Operacao: /
```

Saída

```
16.00 / 3.00 = 5.33
```

Entrada

```
N1: 13  
N2: 76  
Escolha uma operacao de acordo com o menu:  
+) Adicao;  
-) Subtracao;  
*) Multiplicacao;  
/) Divisao.  
Operacao: x
```

Saída

```
Opcão invalida!
```

Exercício 2.19:

Escreva um programa que exiba um menu ao usuário, onde ele poderá escolher entre converter um valor em graus Celsius para graus Fahrenheit, ou então converter um valor em graus Fahrenheit para graus Celsius. Os valores das temperaturas são valores decimais. Caso o usuário forneça uma opção inválida, o programa deve exibir uma mensagem dizendo que a opção é inválida e deve terminar sua execução. Lembrando que:

- $C = \frac{F - 32}{1,8}$
- $F = 1,8C + 32$
- Onde:
 - C é a temperatura em graus Celsius;
 - F é a temperatura em graus Fahrenheit.

Formate a saída dos números decimais usando 2 casas de precisão.

Arquivo com a solução: Exercicio2\$19.java

Entrada

Escolha uma operacao de acordo com o menu:

- C) Celsius -> Fahrenheit;
- F) Fahrenheit -> Celsius.

Opcão: C

Entre com a temperatura em graus Celsius: 38.5

Saída

38.50 graus Celsius correspondem a 101.30 graus Fahrenheit

Entrada

Escolha uma operacao de acordo com o menu:

- C) Celsius -> Fahrenheit;
- F) Fahrenheit -> Celsius.

Opcão: F

Entre com a temperatura em graus Fahrenheit: 125.7

Saída

125.70 graus Fahrenheit correspondem a 52.06 graus Celsius

Entrada

Escolha uma operação de acordo com o menu:

- C) Celsius -> Fahrenheit;
- F) Fahrenheit -> Celsius.

Opcão: x

Saída

Opcão invalida!

Vamos aos exercícios criativos?

2.3 Exercícios Criativos

Exercício Criativo 2.1:

Escreva um programa que peça para o usuário fornecer dois valores inteiros. Seu programa deve apresentar esses valores em ordem crescente usando uma representação em barras. Utilize as cores que você desejar em uma tela de cor branca ou preta de 600 pixels de largura por 400 pixels de altura. Para executar essa tarefa, você precisará desenhar textos na tela. Para isso, use os métodos `drawText` e `String.format`¹, detalhados abaixo:

Método:

```
1 void drawText( String text,
2                 double x, double y,
3                 int fontSize, Paint paint );
```

- **Nome:** `drawText`
- **Descrição:** Desenha um texto a partir do ponto `(x; y)` usando um tamanho de fonte e um `paint` especificado.
- **Entrada/Parâmetro(s):**
 1. `text`: o texto a ser desenhado, podendo conter pulos de linha;
 2. `x`: um número decimal que representa a coordenada `x` do ponto desejado;
 3. `y`: um número decimal que representa a coordenada `y` do ponto desejado;
 4. `fontSize`: um inteiro que representa o tamanho da fonte a ser utilizado;
 5. `paint`: `paint` para o desenho.
- **Saída/Retorno:** Esse método não retorna nenhum valor, pois é `void`.

Método:

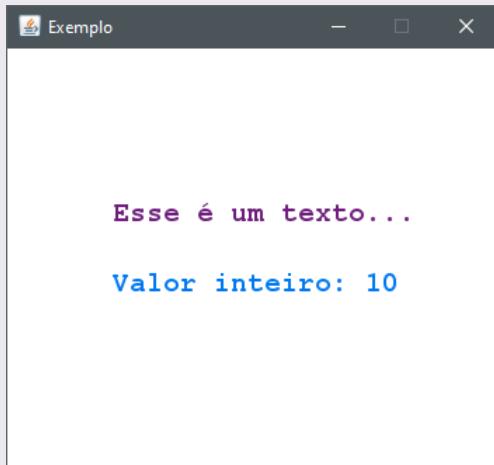
```
1 String.format( String text, Object... args );
```

- **Nome:** `String.format`
- **Descrição:** Processa um padrão e retorna um texto formatado. Funciona de forma semelhante ao método `System.out.printf()`, entretanto, ao invés de direcionar o texto formatado para a saída, ela o gera e o retorna.
- **Entrada/Parâmetro(s):**
 1. `text`: o texto que será usado como padrão;
 2. `args`: valores separados por vírgula que serão usados para preencher os especificadores de formato inseridos no padrão.
- **Saída/Retorno:** Essa função retorna o texto formatado.

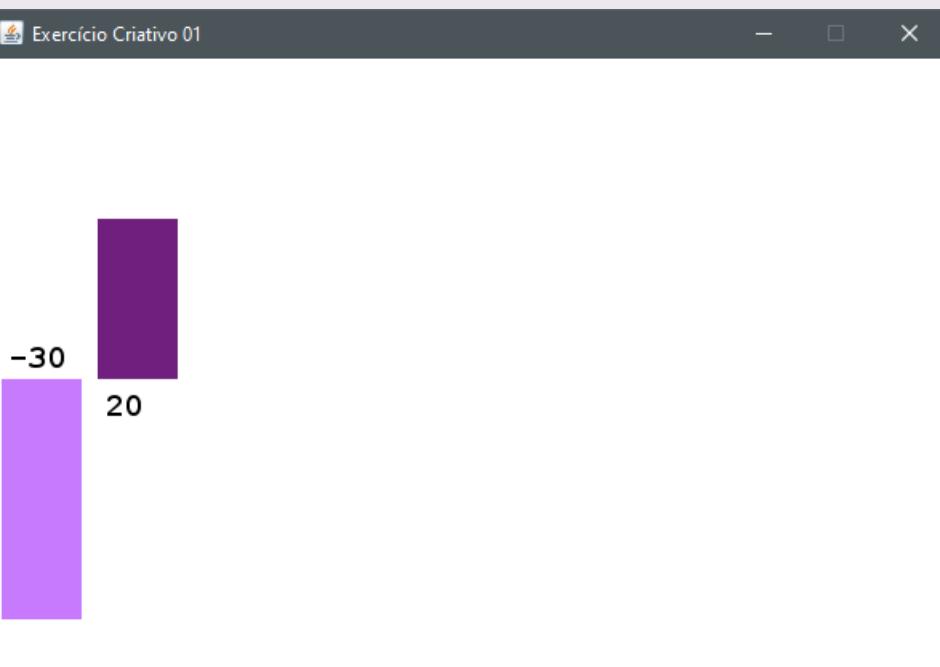
¹O método `format` da classe `String` faz parte da linguagem Java, não sendo uma funcionalidade específica da JSGE.

Exemplo:

```
1 int valor = 10;
2 drawText( "Esse é um texto...", 75, 120, 20, DARKPURPLE );
3 drawText(
4     String.format( "Valor inteiro: %d", valor ), 75, 170, 20, BLUE
5 );
```

**Entrada**

N1: 20
N2: -30

Saída:

Exercício Criativo 2.2:

Escreva um programa que peça para o usuário fornecer três valores inteiros. Seu programa deve apresentar esses valores em ordem crescente usando uma representação em barras. Utilize as cores que você desejar em uma tela de cor branca ou preta de 600 pixels de largura por 400 pixels de altura.

Entrada

N1: 20

N2: 30

N3: 10

Saída:

Exercício Criativo 2.3:

Escreva um programa que desenhe no centro da tela um quadrado que tem a medida do lado igual a 200 pixels. Esse quadrado deve ser pintado utilizando uma cor escolhida pelo usuário (vermelho, verde ou azul). Caso o usuário forneça um valor incorreto, a cor amarela deve ser utilizada. A tela do seu programa deve ser da cor branca com dimensões de 600 pixels de largura por 400 pixels de altura. As dimensões da tela podem ser obtidas através dos dois métodos abaixo:

Método:

```
1 int getScreenWidth();
```

- **Nome:** `getScreenWidth`
- **Descrição:** Retorna a largura atual da tela.
- **Entrada/Parâmetro(s):** Nenhum.
- **Saída/Retorno:** A largura atual da tela `(int)`.

Método:

```
1 int getScreenHeight();
```

- **Nome:** `getScreenHeight`
- **Descrição:** Retorna a altura atual da tela.
- **Entrada/Parâmetro(s):** Nenhum.
- **Saída/Retorno:** A altura atual da tela `(int)`.

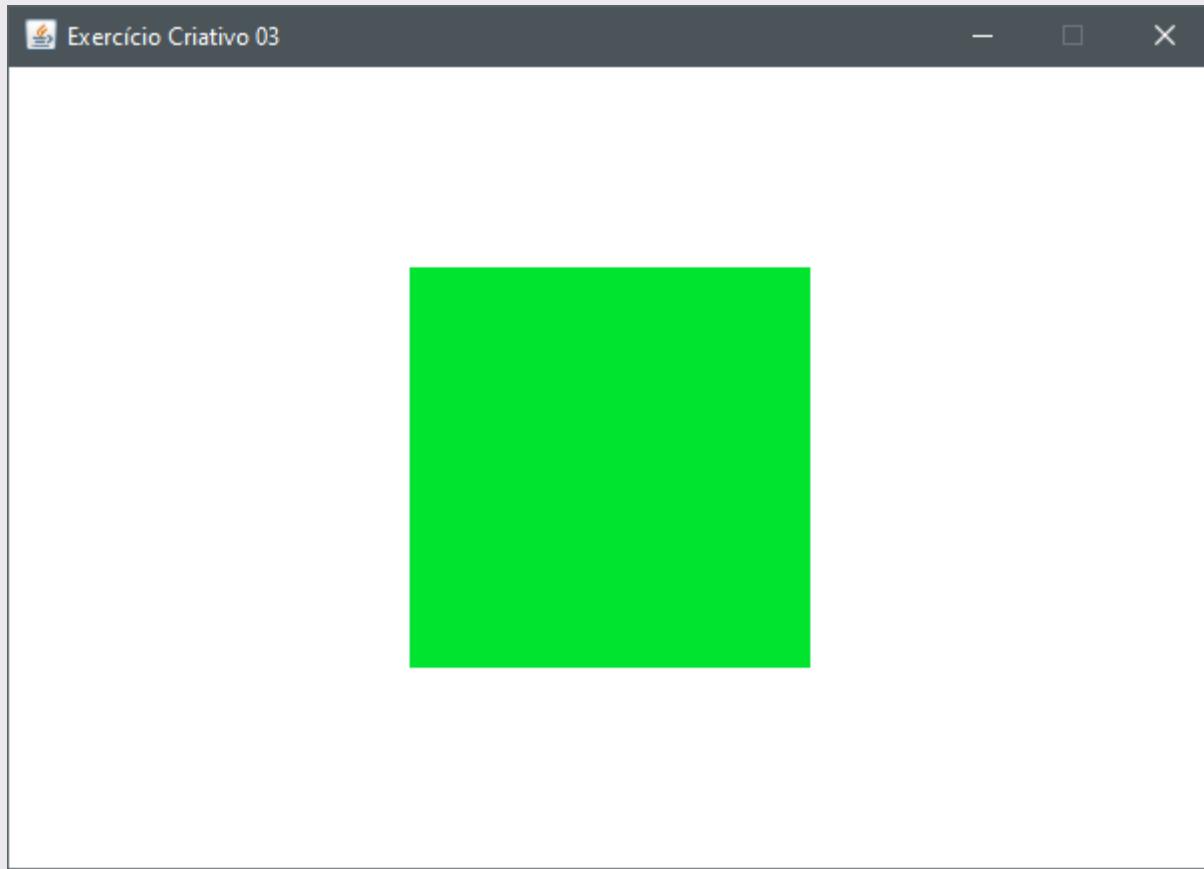
Entrada

Cores disponíveis:

- 1) Vermelho
- 2) Verde
- 3) Azul

Escolha uma cor: 2

Saída:



Exercício Criativo 2.4:

Escreva um programa que solicite ao usuário o valor de uma temperatura em graus Celsius. Esse valor deve ser apresentado em uma tela de fundo branco de 600x400 pixels, usando uma escala pintada com um gradiente linear. Essa escala deve ter dimensões 400x80 pixels. As temperaturas que serão representadas na escala variam de acordo com o intervalo $[-20, 0..180, 0]$. O desenho de retângulos pintados usando gradientes lineares horizontais pode ser feito utilizando o método estático `getHorizontalGradientPaint` da classe `PaintUtils` da JSGE. Você deve usar esse método para criar um paint que, ao invés de ser uma cor sólida, é um gradiente! Veja o detalhe abaixo:

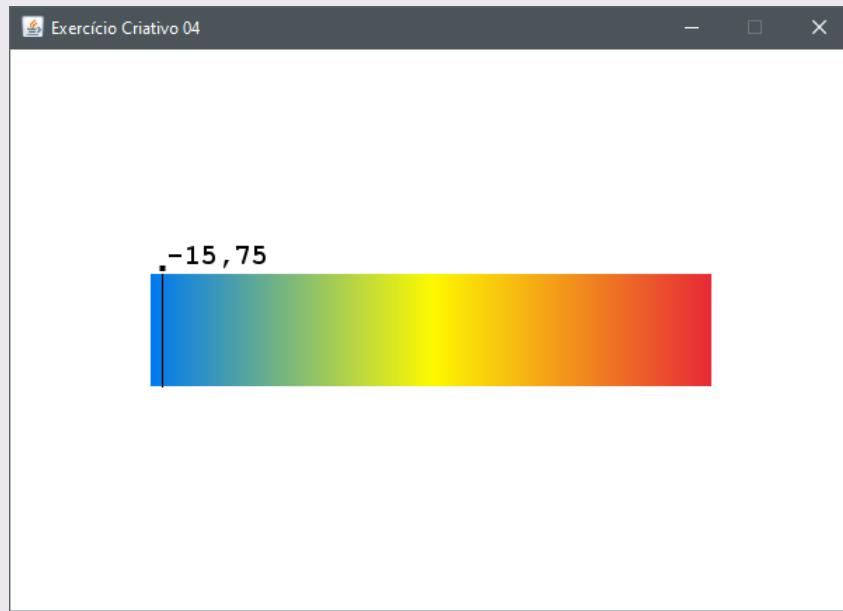
Métodos:

```
1 Paint PaintUtils.getHorizontalGradientPaint(  
2     double x, double y,  
3     double width, double height,  
4     Color startColor, Color endColor  
5 );
```

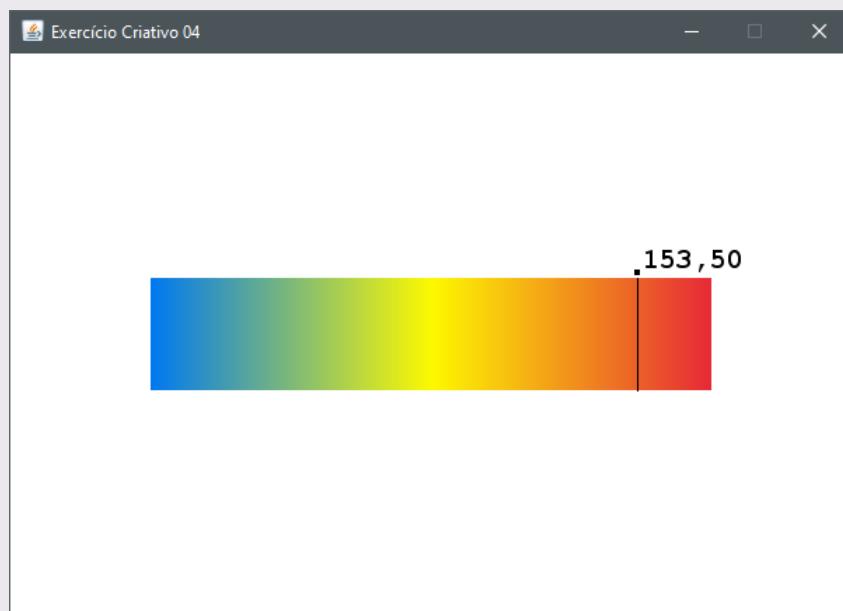
- **Nome:** `getHorizontalGradientPaint`
- **Descrição:** Cria um gradiente linear horizontal utilizando uma cor de início e uma cor de fim.
- **Entrada/Parâmetro(s):**
 1. `x`: um número decimal que representa a coordenada x do limite da cor inicial;
 2. `y`: um número decimal que representa a coordenada y do limite da cor inicial;
 3. `width`: a largura do retângulo que delimita o gradiente, indicando a transição de x até $x + width$, ou seja, da cor inicial até a cor final;
 4. `height`: a altura do retângulo que delimita o gradiente;
 5. `startColor`: a primeira cor do gradiente de pintura;
 6. `endColor`: a segunda cor do gradiente de pintura.
- **Saída/Retorno:** Esse método retorna uma instância de um objeto do tipo `Paint` que será utilizado como o paint do desenho da forma desejada, nesse exemplo, de um retângulo.

Entrada

Temperatura: -15.75

Saída:**Entrada**

Temperatura: 153.5

Saída:

ESTRUTURAS DE REPETIÇÃO

“Bem. E daí? Daí, nada. Quanto a mim, autor de uma vida, me dou mal com a repetição: a rotina me afasta de minhas possíveis novidades”.

Clarice Lispector



S estruturas de repetição permitem que seja possível escrever trechos de código que serão executados repetidamente, a partir da satisfação de uma determinada condição. Neste Capítulo serão apresentadas as principais estruturas de repetição que existem na linguagem de programação Java.

3.1 Estrutura de Repetição *for*

A primeira estrutura de repetição que veremos é a estrutura *for*. O *for* é em geral usado quando sabemos previamente quantas vezes o bloco de código associado à estrutura será executado. Isso não é uma regra absoluta, mas normalmente é assim que é aplicado. Essa quantidade de vezes que o bloco executará, em geral, é relacionada à uma quantidade inteira ou ao tamanho de alguma estrutura de dados. O *for* é construído na maioria das vezes usando apenas uma variável de controle que existirá somente dentro do bloco de código, mas podemos, dependendo da situação, permitir que essa variável seja enxergada também fora do bloco do *for*, além de podermos ter mais de uma variável de controle. Novamente, o uso de apenas uma variável que tenha o escopo no bloco do *for* não é uma regra absoluta, mas é o usual. Outro padrão adotado é que as variáveis que controlam o *for* são normalmente nomeadas de *i*, *j*, *k*, ..., *n*, pois têm base matemática e histórica. Do lado da matemática, devido a uma boa parte da programação se dar na tradução de problemas matemáticos e na solução dos mesmos e do lado histórico por causa da linguagem de programação FORTRAN, onde variáveis do tipo inteiro devem iniciar o nome dos identificadores usando as letras

de I a N. Em alguns exercícios veremos esse paralelo da representação de equações matemáticas em linguagem de programação.

3.1.1 Operadores Unários de Incremento e Decremento

Usualmente a cada iteração (repetição) do *for* nós precisaremos atualizar o valor da variável de controle. Isso se dá normalmente incrementando (aumentando) ou decrementando (diminuindo) o valor da mesma em uma unidade. Mais uma vez, isso não é uma regra imutável, mas na grande maioria das vezes é isso que precisará ser feito. Sendo assim, existem dois operadores unários (aplicados a apenas um operando) que são usados. Após a apresentação da sintaxe e do diagrama de fluxo do *for*, há uma listagem de código mostrando o uso dos mesmos. Esses operadores podem ser vistos na Tabela 3.1.

Operador	Significado
<code>++</code>	Incremento (soma um) pré e pós fixado
<code>--</code>	Decremento (subtrai um) pré e pós fixado

Tabela 3.1: Operadores unários de incremento e decremento

🔗 | Boas Práticas

- Nomeie as variáveis de controle das estruturas de repetição *for* como i, j, k, ..., n;
- Sempre que possível utilize apenas uma variável para o controle da execução do *for*;
- Sempre utilize chaves para delimitar o bloco de código do *for*, mesmo que ele contenha apenas uma linha de código;
- Realize o teste de mesa para verificar o que o seu algoritmo com repetição está fazendo de modo a encontrar problemas.

🔗 | Boas Práticas

- Atualize o valor das variáveis de controle preferencialmente na seção de passo;
- Procure utilizar as três seções do *for*, mesmo que as três sejam opcionais.

⚠️ | Atenção!

As seções de inicialização, teste e passo do *for* são **separadas por ponto e vírgula**.

3.1.2 Exemplo e Diagrama de Fluxo da Estrutura de Repetição *for* (para)

Estrutura do *for* - Verifique a Figura 3.1

```

1 // antes
2 (1)
3         (2)      (3)      (4)
4 for ( inicialização; teste; passo ) {
5     (5)
6 }
7 (6)
8 // depois
  
```

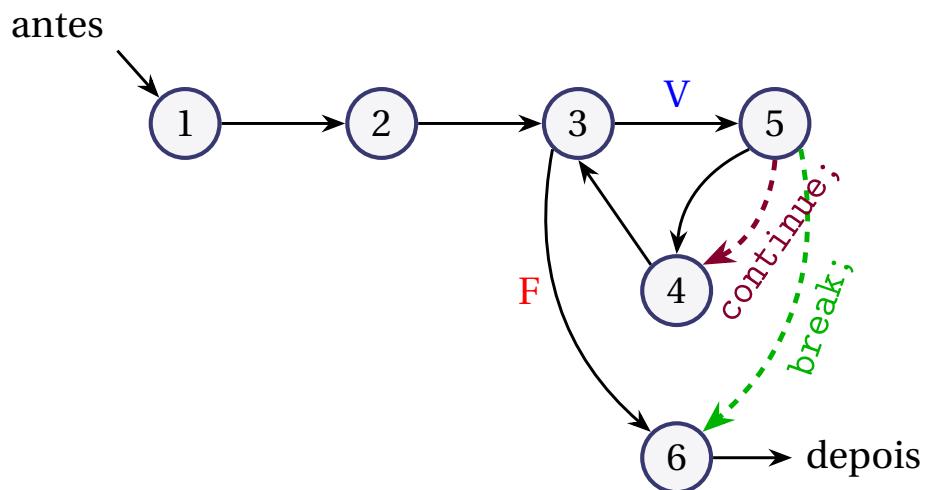


Figura 3.1: Fluxo de execução da estrutura de repetição *for*

Trecho de código exemplificando o funcionamento do operador unário de incremento

```

1 /*
2 * Arquivo: capitulo03/EstruturasDeRepeticaoOperadoresIncrDecr.java
3 * Autor: Prof. Dr. David Buzatto
4 */
5
6 public class EstruturasDeRepeticaoOperadoresIncrDecr {
7
8     public static void main( String[] args ) {
9
10        /*
11         * O funcionamento do operador unário de decremento (--).
12         * é análogo ao operador unário de incremento (++).
13         */
14        int i = 0;
15
16        System.out.println( i );    // imprime 0
17
  
```

```
18     i++; // incremento pós-fixado
19     System.out.println( i ); // imprime 1
20
21     ++i; // incremento pré-fixado
22     System.out.println( i ); // imprime 2
23
24     /*
25      * Incremento pós-fixado em uma expressão:
26      *     usa o valor, depois incrementa.
27      */
28     System.out.println( i++ ); // imprime 2
29     System.out.println( i ); // imprime 3
30
31     /*
32      * Incremento pré-fixado em uma expressão:
33      *     incrementa depois usa o valor.
34      */
35     System.out.println( ++i ); // imprime 4
36     System.out.println( i ); // imprime 4
37
38 }
39
40 }
```

⚠ | Atenção!

Tome cuidado com a geração de laços/*loops* infinitos, ou seja, laços que nunca terminam. Preste sempre atenção no estado das variáveis que controlam as estruturas de repetição. Caso ocorra algum *loop* infinito durante a execução do seu programa, investigue seu algoritmo e o estado da ou das variáveis de controle das suas estruturas de repetição.

Vamos aos exercícios!

3.1.3 Exercícios

Exercício 3.1:

Escreva um programa que imprima os números inteiros de 0, inclusive, a 20, inclusive, usando a estrutura de repetição for.

Arquivo com a solução: `Exercicio3$1.java`

Saída

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Exercício 3.2:

Escreva um programa que imprima os números inteiros pares que estão no intervalo entre 0, inclusive, e 50, inclusive, usando a estrutura de repetição `for`.

Arquivo com a solução: Exercicio3\$2.java

Saída

```
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50
```

Exercício 3.3:

Escreva um programa que imprima os números inteiros de 20, inclusive, a 0, inclusive, usando a estrutura de repetição for

Arquivo com a solução: Exercicio3\$3.java

Saída

```
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

Exercício 3.4:

Escreva um programa que apresente o quadrado dos números inteiros de 15, inclusive, a 30, inclusive, usando a estrutura de repetição `for`

Arquivo com a solução: Exercicio3\$4.java

Saída

```
225 256 289 324 361 400 441 484 529 576 625 676 729 784 841 900
```

Exercício 3.5:

Escreva um programa que peça para o usuário entrar com um número inteiro maior ou igual a 0. Se um valor incorreto for digitado, o programa deve avisar o usuário imprimindo na tela a mensagem “Valor incorreto (negativo)” e terminar. Caso o número seja correto, o programa deve exibir os números de 0 ao número digitado.

Arquivo com a solução: Exercicio3\$5.java

Entrada

```
Forneca um numero maior ou igual a zero: 7
```

Saída

```
0 1 2 3 4 5 6 7
```

Entrada

```
Forneca um numero maior ou igual a zero: -5
```

Saída

```
Valor incorreto (negativo)
```

Exercício 3.6:

Escreva um programa que peça para o usuário entrar com um número inteiro maior ou igual a 0. Se um valor incorreto for digitado, o programa deve avisar o usuário imprimindo na tela a mensagem “Valor incorreto (negativo)” e terminar. Caso o número seja correto, o programa deve exibir os números do número digitado até 0.

Arquivo com a solução: Exercicio3\$6.java

Entrada

```
Forneca um numero maior ou igual a zero: 7
```

Saída

```
7 6 5 4 3 2 1 0
```

Entrada

```
Forneca um numero maior ou igual a zero: -10
```

Saída

```
Valor incorreto (negativo)
```

Exercício 3.7:

Escreva um programa que peça para o usuário entrar com um número inteiro menor ou igual a 0. Se um valor incorreto for digitado, o programa deve avisar o usuário imprimindo na tela a mensagem “Valor incorreto (positivo)” e terminar. Caso o número seja correto, o programa deve exibir os números do número digitado até 0.

Arquivo com a solução: Exercicio3\$7.java

Entrada

```
Forneca um numero menor ou igual a zero: -10
```

Saída

```
-10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0
```

Entrada

```
Forneca um numero menor ou igual a zero: 20
```

Saída

```
Valor incorreto (positivo)
```

Exercício 3.8:

Escreva um programa que peça para o usuário entrar com um número inteiro menor ou igual a 0. Se um valor incorreto for digitado, o programa deve avisar o usuário imprimindo na tela a mensagem “Valor incorreto (positivo)” e terminar. Caso o número seja correto, o programa deve exibir os números de 0 ao número digitado.

Arquivo com a solução: Exercicio3\$8.java

Entrada

```
Forneca um numero menor ou igual a zero: -9
```

Saída

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

Entrada

```
Forneca um numero menor ou igual a zero: 15
```

Saída

```
Valor incorreto (positivo)
```

Exercício 3.9:

Escreva um programa que peça para o usuário fornecer um número inteiro. O programa deve exibir a tabuada de 0 a 10 desse número.

Arquivo com a solução: Exercicio3\$9.java

Entrada

Tabuada do Numero: 5

Saída

```
5 x 0 = 0
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

Exercício 3.10:

Escreva um programa que apresente se cada número inteiro no intervalo entre 45, inclusive, e 60, inclusive, é divisível ou não por 4. Utilize a estrutura de repetição `for`.

Arquivo com a solução: Exercicio3\$10.java

Saída

```
45: indivisivel
46: indivisivel
47: indivisivel
48: divisivel
49: indivisivel
50: indivisivel
51: indivisivel
52: divisivel
53: indivisivel
54: indivisivel
55: indivisivel
56: divisivel
57: indivisivel
58: indivisivel
59: indivisivel
60: divisivel
```

Exercício 3.11:

Escreva um programa que peça para o usuário fornecer dois números inteiros. Se o primeiro número for menor ou igual ao segundo, o programa deve exibir todos os números no intervalo entre os números digitados em ordem crescente. Caso o primeiro número seja maior que o segundo, o programa deve exibir todos os números no intervalo entre os números digitados em ordem decrescente.

Arquivo com a solução: Exercicio3\$11.java

Entrada

N1: 2
N2: 10

Saída

2 3 4 5 6 7 8 9 10

Entrada

N1: 30
N2: 20

Saída

30 29 28 27 26 25 24 23 22 21 20

Exercício 3.12:

Escreva um programa que peça para o usuário fornecer dois números inteiros. O programa deve contar e exibir a quantidade de números pares que existem no intervalo compreendido entre os dois números informados, considerando esses dois números. Fique atento à ordem de entrada dos números.

Arquivo com a solução: Exercicio3\$12.java

Entrada

N1: 5
N2: 100

Saída

Numeros pares entre 5 e 100: 48

Entrada

N1: 20
N2: -30

Saída

Numeros pares entre -30 e 20: 26

Exercício 3.13:

Escreva um programa que conte quantos números inteiros múltiplos de 2, múltiplos de 3 e múltiplos de 4 existem no intervalo de dois números inteiros fornecidos pelo usuário. Esses contadores devem ser exibidos no final. Fique atento à ordem de entrada dos números.

Arquivo com a solução: Exercicio3\$13.java

Entrada

```
N1: -50  
N2: 200
```

Saída

```
Multiplos de 2: 126  
Multiplos de 3: 83  
Multiplos de 4: 63
```

Entrada

```
N1: 50  
N2: -100
```

Saída

```
Multiplos de 2: 76  
Multiplos de 3: 50  
Multiplos de 4: 38
```

Exercício 3.14:

Escreva um programa que calcule o somatório dos valores compreendidos entre dois números inteiros fornecidos pelo usuário, incluindo tais números no cálculo. Fique atento à ordem de entrada dos números.

Arquivo com a solução: Exercicio3\$14.java

Entrada

N1: -5

N2: 50

Saída

Somatorio entre -5 e 50: 1260

Entrada

N1: 80

N2: -10

Saída

Somatorio entre -10 e 80: 3185

Exercício 3.15:

Escreva um programa que peça para o usuário fornecer um número inteiro positivo e que calcule o fatorial desse número. Caso o número seja negativo, o programa deve avisar o usuário, usando a mensagem “Nao ha fatorial de numero negativo.” (sem acentos) e terminar. Caso contrário o programa deve calcular o fatorial do número digitado e exibi-lo. Lembrando que:

- $n! = \prod_{i=1}^n i, n \in \mathbb{N}^*$
- Onde:
 - $n!$ é o fatorial de n .

Arquivo com a solução: Exercicio3\$15.java

Entrada

Numero: 5

Saída

5! = 120

Entrada

Numero: -10

Saída

Nao ha fatorial de numero negativo.

Exercício 3.16:

Escreva um programa que exiba os vinte primeiros termos da série de Fibonacci. A série de Fibonacci inicia com 1 e 1, sendo os próximos termos gerados pela soma dos dois últimos termos. Os nove primeiros termos da série de Fibonacci são: 1 1 2 3 5 8 13 21 34. Obs: O primeiro termo tem posição 0, o segundo termo tem posição 1, o terceiro termo tem posição 2 e assim por diante.

Arquivo com a solução: Exercicio3\$16.java

Saída

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

Exercício 3.17:

Escreva um programa que peça ao usuário o termo da série de Fibonacci que ele quer que seja obtido e que então exiba esse termo.

Arquivo com a solução: Exercicio3\$17.java

Entrada

Termo desejado: 0

Saída

Fibonacci de 0 e 1

Entrada

Termo desejado: 1

Saída

Fibonacci de 1 e 1

Entrada

Termo desejado: 5

Saída

Fibonacci de 5 e 8

Entrada

Termo desejado: 15

Saída

Fibonacci de 15 e 987

Exercício 3.18:

Escreva um programa que exiba o seguinte desenho usando, obrigatoriamente, a estrutura de repetição `for`. Você só pode utilizar as seguintes construções para gerar a saída:

- `System.out.print("*")` e
- `System.out.println()`

Arquivo com a solução: `Exercicio3$18.java`

Saída

```
*  
**  
***  
****  
*****
```

Exercício 3.19:

Escreva um programa que exiba o seguinte desenho usando, obrigatoriamente, a estrutura de repetição `for`. Você só pode utilizar as seguintes construções para gerar a saída:

- `System.out.print("*")` e
- `System.out.println()`

Arquivo com a solução: `Exercicio3$19.java`

Saída

```
*  
**  
***  
****  
*****  
****  
***  
**  
*
```

Exercício 3.20:

Escreva um programa que exiba o seguinte desenho usando, obrigatoriamente, a estrutura de repetição `for`. Os asteriscos quase pretos indicam espaços. Você só pode utilizar as seguintes construções para gerar a saída:

- `System.out.print("*"),`
- `System.out.print(" ")` e
- `System.out.println()`

Arquivo com a solução: Exercicio3\$20.java

Saída

```
*  
**  
***  
****  
*****  
  
*****  
****  
***  
**  
*  
  
*****  
*****  
*****  
*****  
*****  
  
*****  
*****  
***  
****  
*****  
*****
```

Exercício 3.21:

Escreva um programa que leia uma altura em inteiro e imprima um triângulo de asteriscos, baseado nessa altura. Uma altura negativa deve resultar em um triângulo de cabeça para baixo. Uma altura igual a zero não produzirá um triângulo. Os asteriscos quase pretos indicam espaços. Você só pode utilizar as seguintes construções para gerar a saída:

- `System.out.print("*")`,
- `System.out.print(" ")` e
- `System.out.println()`

Arquivo com a solução: Exercicio3\$21.java

Entrada

```
Altura: 3
```

Saída

```
***  
***  
***
```

Entrada

```
Altura: 11
```

Saída

```
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****
```

Entrada

```
Altura: -5
```

Saída

```
*****  
* * * * *  
* * * * *  
* * * * *  
*****
```

Exercício 3.22:

Escreva um programa que peça para o usuário fornecer 5 números positivos. Caso algum seja menor ou igual a zero, após o usuário fornecer todos os números, o programa deve avisar o usuário e terminar com a mensagem “Forneca apenas numeros positivos.”. Os valores fornecidos devem ser usados para desenhar um gráfico de barras usando o símbolo asterisco. Os asteriscos quase pretos indicam espaços.

Arquivo com a solução: Exercicio3\$22.java

Entrada

```
N1: 3  
N2: 5  
N3: 10  
N4: 2  
N5: 6
```

Saída

```
0010*****  
0009*****  
0008*****  
0007*****  
0006*****  
0005*****  
0004*****  
0003*****  
0002*****  
0001*****
```

Entrada

```
N1: 4  
N2: 8  
N3: 0  
N4: -7  
N5: 2
```

Saída

Forneca apenas numeros positivos.

💡 | Dica

Para formatar um valor inteiro usando zeros à esquerda para preenchimento, use o especificador de formato `"%0nd"`, onde `n` é a quantidade de casas que o valor e os zeros ocuparão. Por exemplo, uma variável que contém o valor 15, ao ser formatada usando o especificador

de formato `"%04d"`, resultará em 0015, ou seja, o número inteiro 15, precedido de dois zeros, ocupando quatro casas. Veja o código abaixo:

```
1 int n = 15;  
2 System.out.printf( "%04d", n );      // imprime 0015
```

Exercício 3.23:

Escreva um programa para ler as notas de 10 alunos de uma turma e calcular e exibir a média aritmética destas notas. Armazene a média em uma variável. As notas são números decimais. Utilize a estrutura de repetição `for` para coletar as notas.

Arquivo com a solução: Exercicio3\$23.java

Entrada

```
Forneca a nota de 10 alunos:  
Nota 01: 6  
Nota 02: 8  
Nota 03: 9  
Nota 04: 8.75  
Nota 05: 7  
Nota 06: 5  
Nota 07: 6  
Nota 08: 7.5  
Nota 09: 8  
Nota 10: 9
```

Saída

```
A media aritmetica das dez notas e: 7.43
```

3.2 Estruturas de Repetição *while* (enquanto) e *do... while* (faça ... enquanto)

Assim como o *for*, o *while* e o *do... while* são usados para realizar iterações, mas a diferença de aplicação é que enquanto no *for* sabemos quantas vezes ele executará com base em uma quantidade, no *while* e no *do... while* nós não temos certeza quantas vezes os laços executarão. Mais uma vez, essa regra não é absoluta, pois podemos fazer a mesma coisa com as três estruturas de repetição, “simulando” uma na outra. Há professores que ensinam as três, os modos de uso, mas depois pedem para os alunos escreverem programas usando *while* onde se deveria usar o *for* usualmente, além de outras combinações, mas eu acho isso uma perda de tempo e pode criar confusão, ainda mais em um momento que o aprendizado fatalmente começa a ter uma certa dificuldade. A diferença entre as estruturas *while* e *do... while* é que no *while* o teste que será feito ocorre antes da execução do bloco de código associado, ou seja, caso o teste gere um valor falso na primeira vez em que for executado, o bloco de código não executará. No *do... while* é garantido que o bloco de código executa pelo menos uma vez, pois o teste ocorre após a execução do bloco de código.

🔗 | Boa Prática

Sempre utilize a estrutura de repetição apropriada para o problema que está sendo resolvido.

💡 | Dicas

- Se a quantidade de iterações é sabida, seja de forma fixa, ou baseada em um valor numérico ou no tamanho de algo que precisa ser processado, utilize *for*.
- Se a quantidade de iterações não é sabida, utilize *while* ou *do... while* e:
 - Se você precisa que o **bloco de código execute após o teste, use *while***;
 - Se o **bloco de código precisa ser executado antes do teste, use *do... while***.

3.2.1 Exemplo e Diagrama de Fluxo da Estrutura de Repetição *while* e *do...while*

Estrutura do *while* - Verifique a Figura 3.2a

```

1 // antes
2 (1)
3     (2)
4 while ( teste ) {
5     (3)
6 }
7 (4) // depois

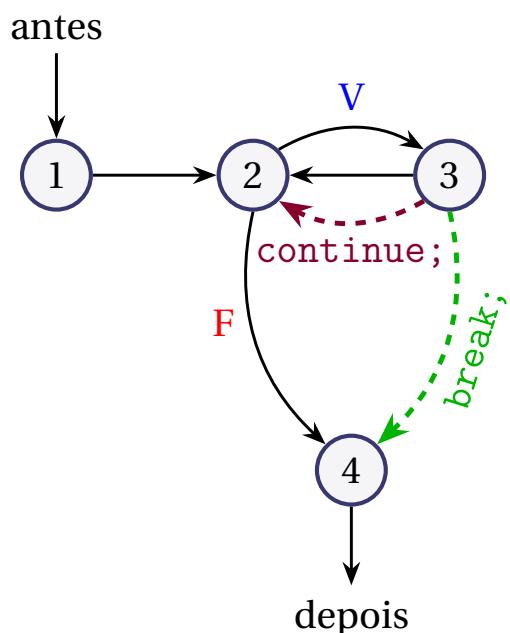
```

Estrutura do *do...while* - Verifique a Figura 3.2b

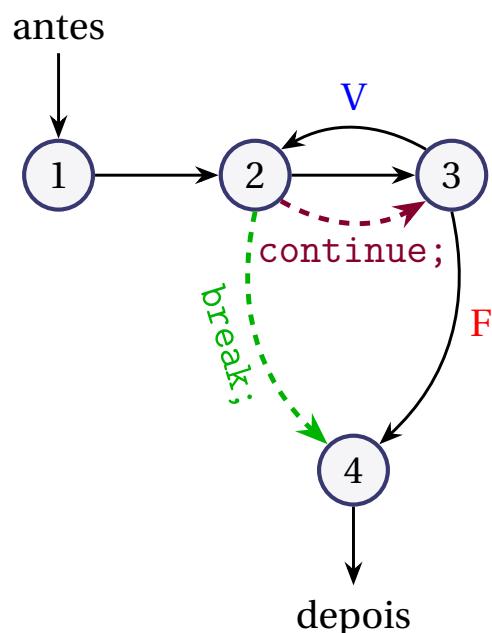
```

1 // antes
2 (1)
3 do {
4     (2)
5 } while ( teste );
6 (4) // depois

```



(a) Fluxo do *while*



(b) Fluxo do *do...while*

Figura 3.2: Fluxos de execução das estruturas de repetição *while* e *do...while*

Vamos aos exercícios!

3.2.2 Exercícios

Exercício 3.24:

Escreva um programa que solicite a idade de várias pessoas e imprima o total de pessoas com menos de 21 anos e o total de pessoas com mais de 50 anos. O programa deve terminar e exibir os resultados quando a idade fornecida for negativa.

Arquivo com a solução: Exercicio3\$24.java

Entrada

```
Idade da pessoa 01: 10  
Idade da pessoa 02: 55  
Idade da pessoa 03: -1
```

Saída

```
Total de pessoas menores de 21 anos: 1  
Total de pessoas com mais de 50 anos: 1
```

Entrada

```
Idade da pessoa 01: 9  
Idade da pessoa 02: 15  
Idade da pessoa 03: 57  
Idade da pessoa 04: 20  
Idade da pessoa 05: 23  
Idade da pessoa 06: 19  
Idade da pessoa 07: 43  
Idade da pessoa 08: 66  
Idade da pessoa 09: -10
```

Saída

```
Total de pessoas menores de 21 anos: 4  
Total de pessoas com mais de 50 anos: 2
```

3.2. ESTRUTURAS DE REPETIÇÃO WHILE (ENQUANTO) E DO... WHILE (FAÇA ... ENQUANTO)

Exercício 3.25:

Escreva um programa que efetue a leitura sucessiva de valores numéricos decimais e apresente no final o somatório, a média e a quantidade de valores lidos, armazenada como inteiro. O programa deve continuar lendo os números até que seja fornecido um número negativo. Esse número negativo não deve entrar nos cálculos! Formate a saída dos números decimais usando 2 casas de precisão. Caso o número negativo seja o primeiro a ser fornecido, o programa deve exibir um somatório igual a zero, uma média igual a zero e uma quantidade igual a zero.

Arquivo com a solução: Exercicio3\$25.java

Entrada

```
Entre com um valor: 4  
Entre com um valor: 8  
Entre com um valor: -1
```

Saída

```
Somatorio: 12.00  
Media: 6.00  
Quantidade: 2
```

Entrada

```
Entre com um valor: 5  
Entre com um valor: 8  
Entre com um valor: 10  
Entre com um valor: 15  
Entre com um valor: 2  
Entre com um valor: 9  
Entre com um valor: 3  
Entre com um valor: 2  
Entre com um valor: -1
```

Saída

```
Somatorio: 54.00  
Media: 6.75  
Quantidade: 8
```

Entrada

```
Entre com um valor: -5
```

Saída

Somatorio: 0.00

Media: 0.00

Quantidade: 0

3.2. ESTRUTURAS DE REPETIÇÃO WHILE (ENQUANTO) E DO... WHILE (FAÇA ... ENQUANTO)

Exercício 3.26:

Escreva um programa que efetue a leitura sucessiva de valores numéricos inteiros e apresente no final o menor e o maior número que foram fornecidos. O programa deve continuar lendo os números até que seja fornecido um número negativo, que por sua vez não deve ser apresentado como menor ou maior número. Caso o número negativo seja o primeiro a ser fornecido, o programa deve exibir tanto o menor quanto o maior número como zero. Pense no que precisa ser feito para inicializar apropriadamente os valores das variáveis menor e maior.

Arquivo com a solução: Exercicio3\$26.java

Entrada

```
Entre com um valor: 7
Entre com um valor: 15
Entre com um valor: 3
Entre com um valor: 29
Entre com um valor: 2
Entre com um valor: 103
Entre com um valor: 0
Entre com um valor: 34
Entre com um valor: -1
```

Saída

```
Menor numero: 0
Maior numero: 103
```

Entrada

```
Entre com um valor: 5
Entre com um valor: -1
```

Saída

```
Menor numero: 5
Maior numero: 5
```

Entrada

```
Entre com um valor: -5
```

Saída

```
Menor numero: 0
Maior numero: 0
```

Exercício 3.27:

Escreva um programa para ler um número indeterminado de dados de pesos de pessoas como números decimais. O último dado, que não entrará nos cálculos, deve ser um valor negativo. O programa deve calcular e imprimir a média aritmética dos pesos das pessoas que possuem mais de 60kg e o peso do indivíduo mais pesado. Formate a saída dos números decimais usando 2 casas de precisão. Caso o número negativo seja o primeiro a ser fornecido, o programa deve exibir a média e o peso mais pesado ambos como zero.

Arquivo com a solução: Exercicio3\$27.java

Entrada

```
Entre com o peso da pessoa 01: 55.8  
Entre com o peso da pessoa 02: 102.7  
Entre com o peso da pessoa 03: 86.3  
Entre com o peso da pessoa 04: -1
```

Saída

```
Media dos pesos acima de 60kg: 94.50  
A pessoa mais pesada possui 102.70kg
```

Entrada

```
Entre com o peso da pessoa 01: -1
```

Saída

```
Media dos pesos acima de 60kg: 0.00  
A pessoa mais pesada possui 0.00kg
```

Entrada

```
Entre com o peso da pessoa 01: 30.0  
Entre com o peso da pessoa 02: -1
```

Saída

```
Media dos pesos acima de 60kg: 0.00  
A pessoa mais pesada possui 30.00kg
```

3.2. ESTRUTURAS DE REPETIÇÃO WHILE (ENQUANTO) E DO... WHILE (FAÇA ... ENQUANTO)

Entrada

Entre com o peso da pessoa 01: 90.0

Entre com o peso da pessoa 02: -1

Saída

Média dos pesos acima de 60kg: 90.00

A pessoa mais pesada possui 90.00kg

Exercício 3.28:

Escreva um programa para ler o saldo inicial de uma conta bancária, um valor decimal. A seguir ler um número indeterminado de pares de valores indicando respectivamente o tipo da operação (codificado da seguinte forma: 1.Depósito 2.Retirada e 3.Fim) e o valor que será movimentado. Quando for informado para o tipo da operação o código 3, o programa deve ser encerrado e impresso o saldo final da conta com as seguintes mensagens: “Sem saldo.” caso o saldo seja zero, “Conta devedora.”, se o saldo for negativo ou “Conta preferencial.”, se o saldo seja positivo. Caso seja fornecido um tipo incorreto de operação, ou seja, diferente de 1, 2 ou 3, o programa deve exibir ao usuário a mensagem “Operacao invalida.” e solicitar novamente a operação. Formate a saída dos números decimais usando 2 casas de precisão.

Arquivo com a solução: Exercicio3\$28.java

Entrada

```
Saldo inicial: 3000
Operacoes:
    1) Deposito;
    2) Saque;
    3) Fim.
Operacao desejada: 1
Valor a depositar: 500
Operacao desejada: 1
Valor a depositar: 300
Operacao desejada: 1
Valor a depositar: 100
Operacao desejada: 2
Valor a sacar: 2555
Operacao desejada: 3
```

Saída

```
Saldo final: R$1345.00
Conta preferencial.
```

3.2. ESTRUTURAS DE REPETIÇÃO WHILE (ENQUANTO) E DO... WHILE (FAÇA ... ENQUANTO)

Entrada

Saldo inicial: 1000

Operações:

- 1) Depósito;
- 2) Saque;
- 3) Fim.

Operação desejada: 2

Valor a sacar: 500

Operação desejada: 2

Valor a sacar: 300

Operação desejada: 2

Valor a sacar: 300

Operação desejada: 3

Saída

Saldo final: -R\$100.00

Conta devolvedora.

Entrada

Saldo inicial: 2000

Operações:

- 1) Depósito;
- 2) Saque;
- 3) Fim.

Operação desejada: 2

Valor a sacar: 1500

Operação desejada: 2

Valor a sacar: 500

Operação desejada: 3

Saída

Saldo final: R\$0.00

Sem saldo.

Exercício 3.29:

Escreva um programa para ler 2 valores inteiros e imprimir o resultado da divisão do primeiro pelo segundo. Se o segundo valor informado for zero, deve ser impressa uma mensagem de “Nao existe divisao inteira por zero!” (sem acentos) e lido um novo valor. Ao final do programa, deve ser impressa a seguinte mensagem: “Voce deseja realizar outro calculo? (S/N): ” Se a resposta for ‘S’ o programa deverá retornar ao começo, repetindo o processo, caso contrário deverá encerrar a sua execução imprimindo quantos cálculos foram feitos. Note que para cada dois valores fornecidos seu programa já deve gerar a saída correspondente! Essa característica é apresentada tanto na entrada, quanto na saída, usando-se cores. Seu programa não deve alterar cor alguma.

Arquivo com a solução: Exercicio3\$29 .java

Entrada

```
N1: 10
N2: 5
Voce deseja realizar outro calculo? (S/N): S
N1: 11
N2: 3
Voce deseja realizar outro calculo? (S/N): S
N1: 15
N2: 0
Entre novamente com N2: 0
Entre novamente com N2: 5
Voce deseja realizar outro calculo? (S/N): N
```

Saída

```
10 / 5 = 2
11 / 3 = 3
Nao existe divisao inteira por zero!
Nao existe divisao inteira por zero!
15 / 5 = 3
```

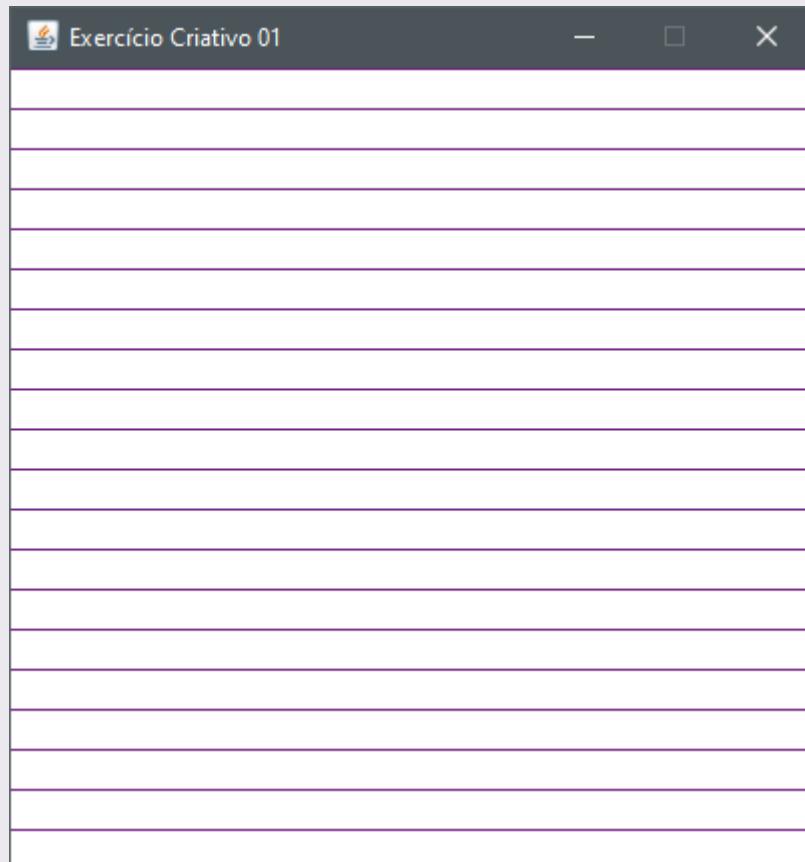
Chegou a hora dos exercícios criativos!

3.3 Exercícios Criativos

Exercício Criativo 3.1:

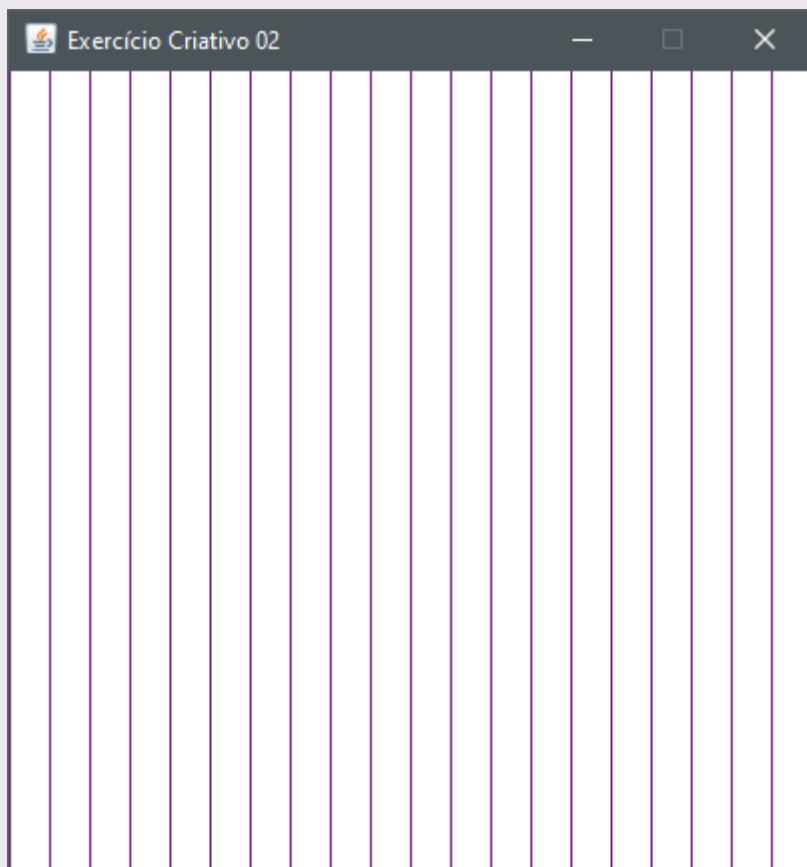
Escreva um programa que desenhe o padrão abaixo. Você está livre para usar as cores que quiser. A única restrição é que sua tela deve ter dimensões 400x400 pixels. Se as dimensões da sua tela forem alteradas o padrão continua a ser desenhado? No exemplo, a distância entre as linhas é de 20 pixels.

Saída:



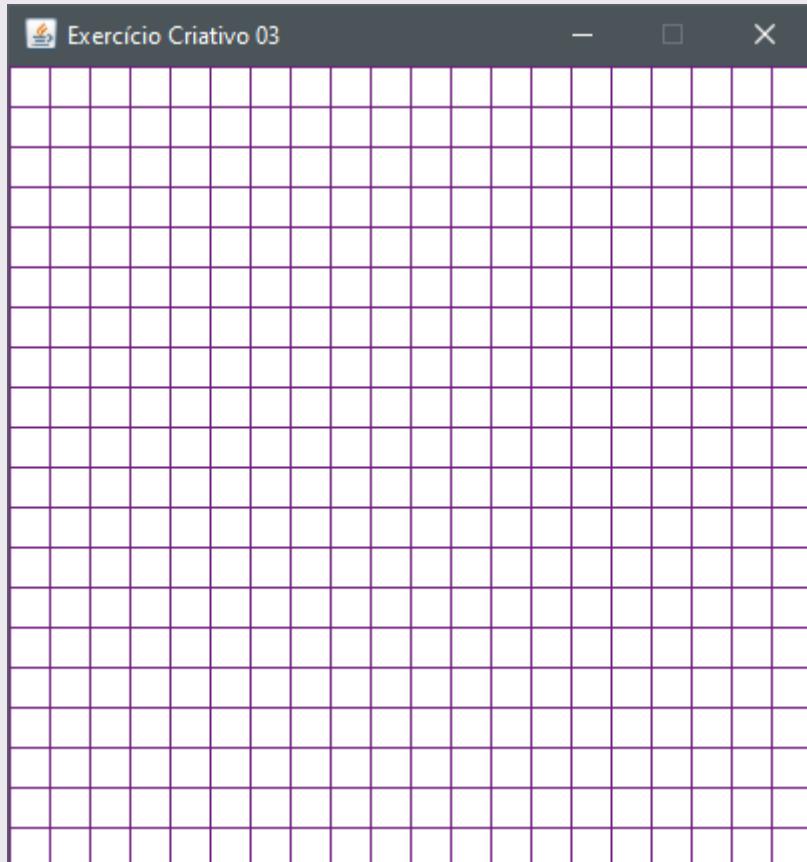
Exercício Criativo 3.2:

Escreva um programa que desenhe o padrão abaixo. Você está livre para usar as cores que quiser. A única restrição é que sua tela deve ter dimensões 400x400 pixels. Se as dimensões da sua tela forem alteradas o padrão continua a ser desenhado? No exemplo, a distância entre as linhas é de 20 pixels.

Saída:

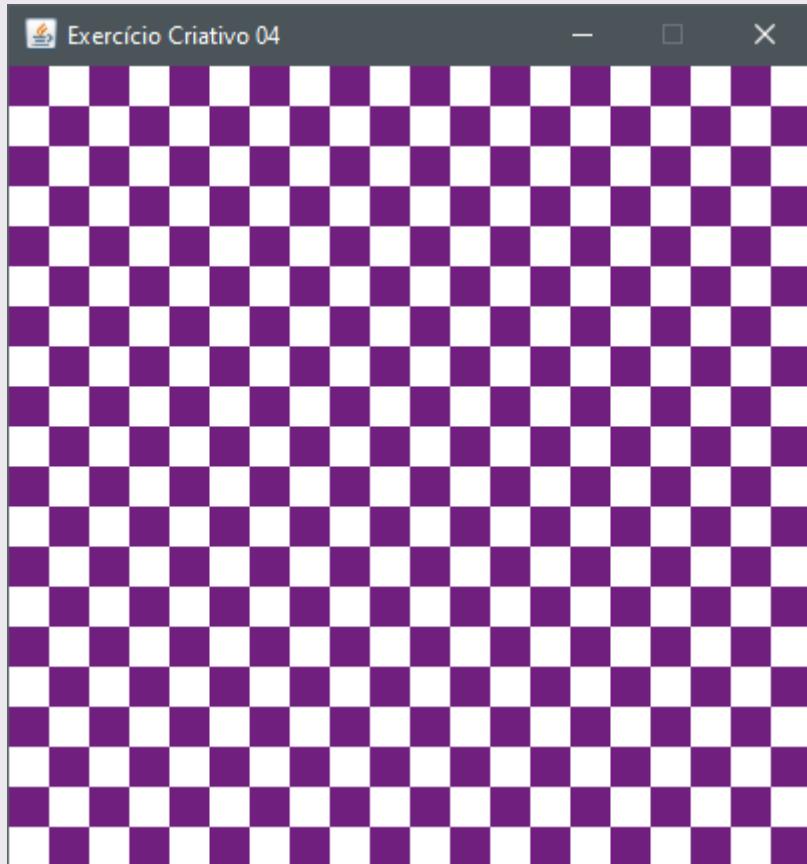
Exercício Criativo 3.3:

Escreva um programa que desenhe o padrão abaixo. Você está livre para usar as cores que quiser. A única restrição é que sua tela deve ter dimensões 400x400 pixels. Se as dimensões da sua tela forem alteradas o padrão continua a ser desenhado? No exemplo, a distância entre as linhas é de 20 pixels.

Saída:

Exercício Criativo 3.4:

Escreva um programa que desenhe o padrão abaixo. Você está livre para usar as cores que quiser. A única restrição é que sua tela deve ter dimensões 400x400 pixels. Se as dimensões da sua tela forem alteradas o padrão continua a ser desenhado? No exemplo, cada quadrado tem lados medindo 20 pixels.

Saída:

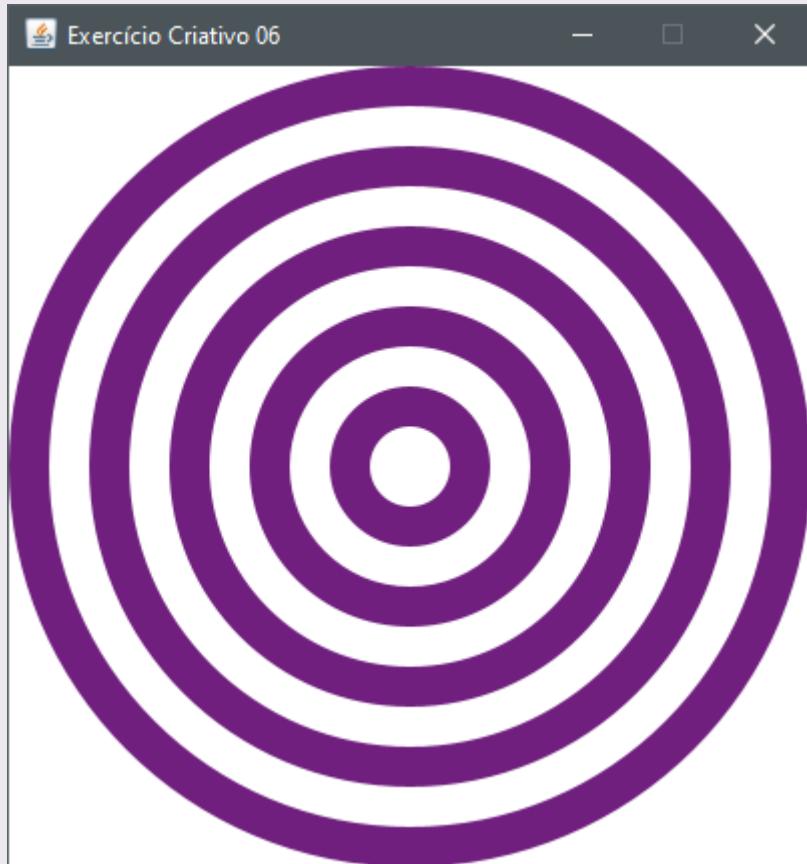
Exercício Criativo 3.5:

Escreva um programa que desenhe o padrão abaixo. Você está livre para usar as cores que quiser. A única restrição é que sua tela deve ter dimensões 400x400 pixels. Se as dimensões da sua tela forem alteradas o padrão continua a ser desenhado? No exemplo, os raios das circunferências adjacentes distam em 20 pixels.

Saída:

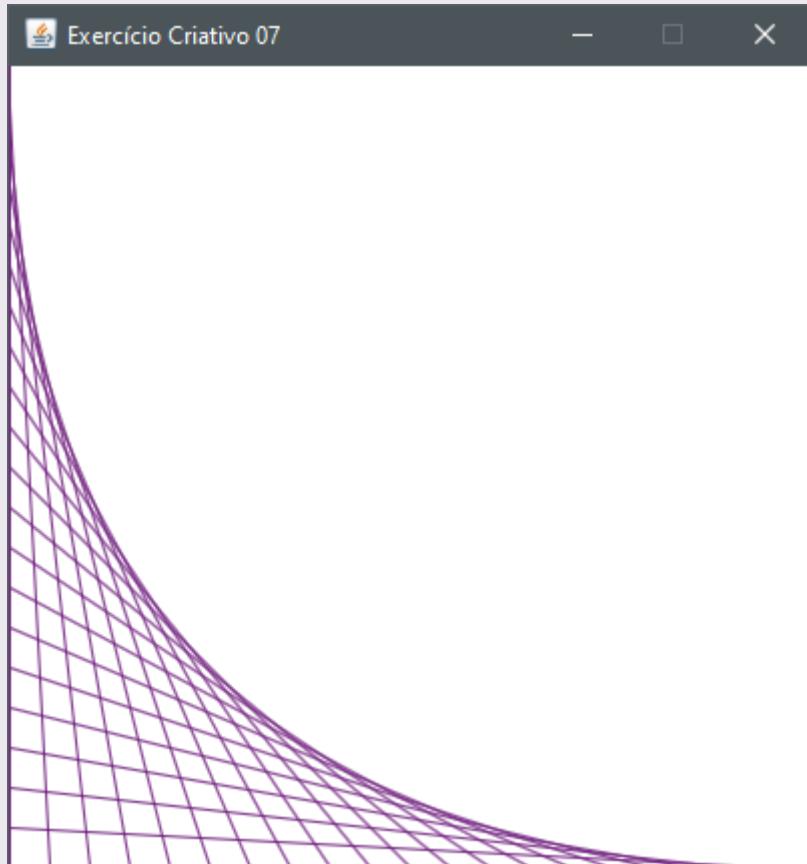
Exercício Criativo 3.6:

Escreva um programa que desenhe o padrão abaixo. Você está livre para usar as cores que quiser. A única restrição é que sua tela deve ter dimensões 400x400 pixels. Se as dimensões da sua tela forem alteradas o padrão continua a ser desenhado? No exemplo, os raios dos círculos adjacentes distam em 20 pixels.

Saída:

Exercício Criativo 3.7 (DEITEL; DEITEL, 2017):

Escreva um programa que desenhe o padrão abaixo. Você está livre para usar as cores que quiser. A única restrição é que sua tela deve ter dimensões 400x400 pixels. Se as dimensões da sua tela forem alteradas o padrão continua a ser desenhado? No exemplo, o espaçamento utilizado entre as linhas é de 20 pixels.

Saída:

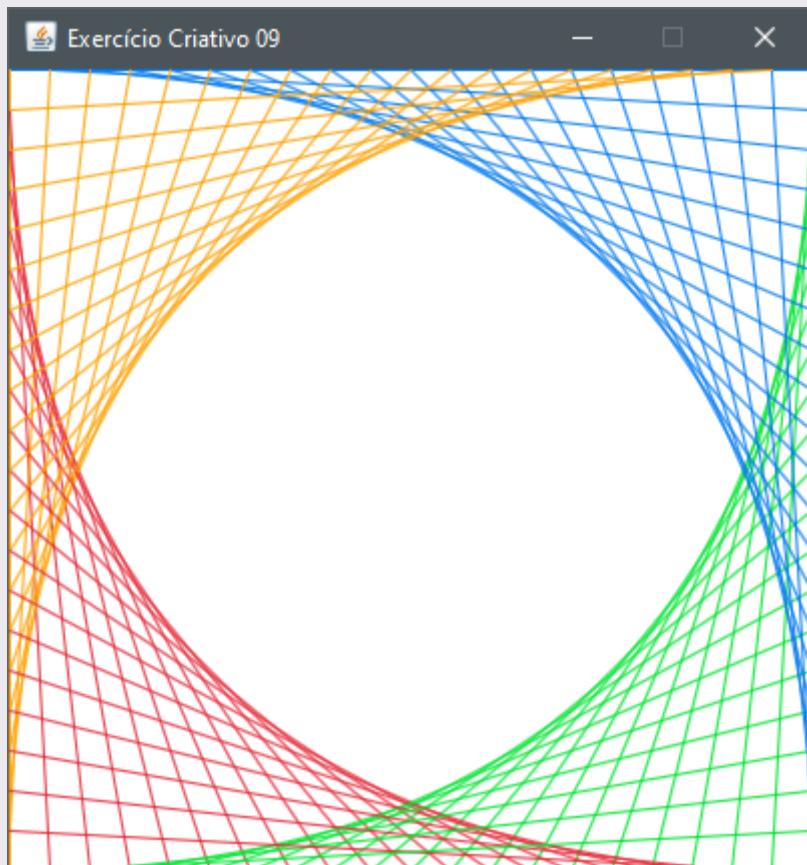
Exercício Criativo 3.8:

Escreva um programa que desenhe o padrão abaixo. Você está livre para usar as cores que quiser. A única restrição é que sua tela deve ter dimensões 400x400 pixels. Se as dimensões da sua tela forem alteradas o padrão continua a ser desenhado? No exemplo, o espaçamento utilizado entre as linhas é de 20 pixels.

Saída:

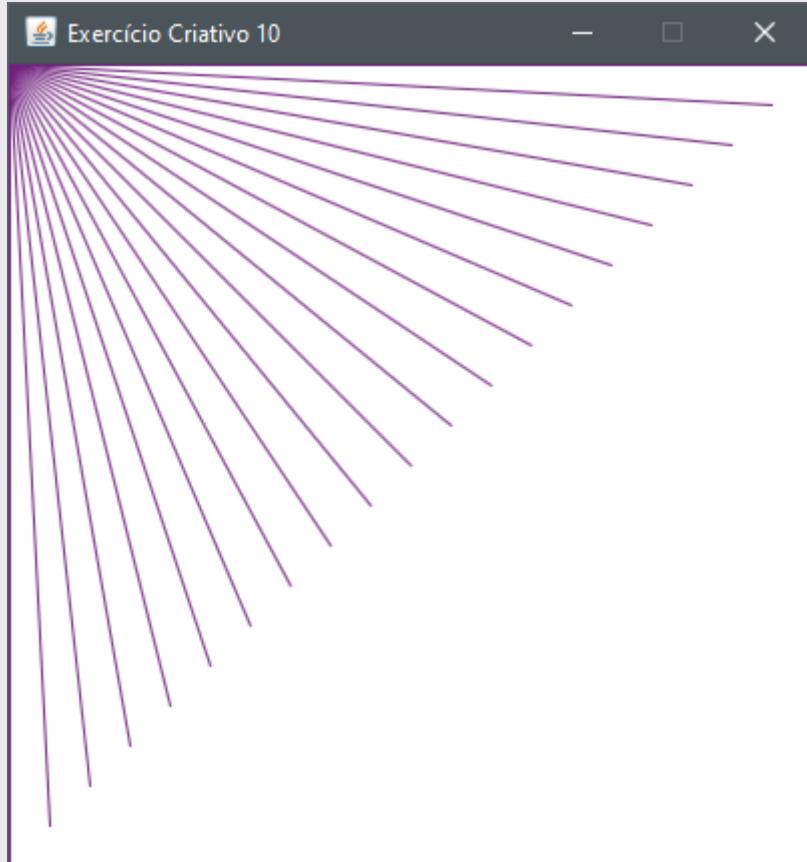
Exercício Criativo 3.9 (DEITEL; DEITEL, 2017):

Escreva um programa que desenhe o padrão abaixo. Você está livre para usar as cores que quiser. A única restrição é que sua tela deve ter dimensões 400x400 pixels. Se as dimensões da sua tela forem alteradas o padrão continua a ser desenhado? No exemplo, o espaçamento utilizado entre as linhas é de 20 pixels.

Saída:

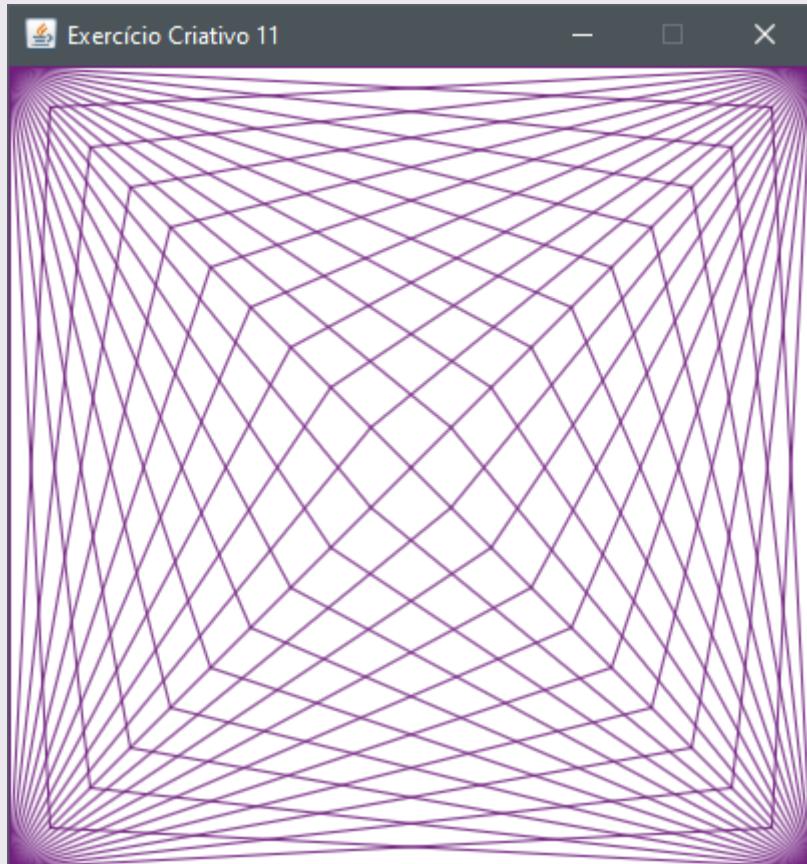
Exercício Criativo 3.10 (DEITEL; DEITEL, 2017):

Escreva um programa que desenhe o padrão abaixo. Você está livre para usar as cores que quiser. A única restrição é que sua tela deve ter dimensões 400x400 pixels. Se as dimensões da sua tela forem alteradas o padrão continua a ser desenhado? No exemplo, o espaçamento utilizado entre as linhas é de 20 pixels.

Saída:

Exercício Criativo 3.11 (DEITEL; DEITEL, 2017):

Escreva um programa que desenhe o padrão abaixo. Você está livre para usar as cores que quiser. A única restrição é que sua tela deve ter dimensões 400x400 pixels. Se as dimensões da sua tela forem alteradas o padrão continua a ser desenhado? No exemplo, o espaçamento utilizado entre as linhas é de 20 pixels.

Saída:

ARRAYS UNIDIMENSIONAIS

*“Algoritmos + Estruturas de Dados
= Programas”.*

Niklaus Wirth



S arrays, ou arranjos, algumas vezes também chamados erroneamente de vetores, são estruturas de dados que armazenam um ou vários elementos de um mesmo tipo, ou seja, são estruturas homogêneas. Neste capítulo serão apresentados os arrays unidimensionais, que são estruturas lineares e indexadas a partir da posição 0.

4.1 Exemplos em Linguagem Java

Declaração e inicialização de arrays

```
1  /*
2   * Arquivo: capitulo04/ArraysUnidimensionaisDeclaracaoInicializacao.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6  public class ArraysUnidimensionaisDeclaracaoInicializacao {
7
8      public static void main( String[] args ) {
9
10         // definição de constante
11         final int N = 2;
12
13         /*
14          * Na linguagem Java, a convenção para nomear
15          * constantes é a de usar apenas letras maiúsculas
16          * e/ou dígitos. Para constantes onde o identificador
17          * precisa de mais de uma palavra, essas palavras são
18          * separadas usando o caractere underscore (_)
19          */
20
21         /*
22          * Declaração de uma variável do tipo array de inteiros
23          * e inicialização com um novo objeto do tipo array de
24          * inteiros de 5 posições.
25          */
26         int[] array1 = new int[5];
27
28         /*
29          * Todo array, quando não especificado, tem os valores
30          * de suas posições inicializados com o valor padrão
31          * do tipo primitivo que ele contém ou um valor nulo
32          * (null) caso o tipo seja de referência.
33          *
34          * Valores padrão:    tipo / valor
35          *                   byte / 0
36          *                   short / 0
37          *                   int / 0
38          *                   long / 0
39          *                   char / '\0'
40          *                   float / 0.0
41          *                   double / 0.0
42          *                   boolean / false
43          *                   referência / null
44          */
45
```

```
46 // declarando um array e inicializando com valores 2
47 // valor inicializado = { 2, 2, 2, 2, 2 }
48 int[] array2 = new int[]{ 2, 2, 2, 2, 2 };
49
50 // declarando um array e inicializando com valores 3
51 // valor inicializado = { 3, 3, 3, 3, 3 }
52 int[] array3 = { 3, 3, 3, 3, 3 };
53
54 // declaração de um array de inteiros de N posições
55 int[] array4 = new int[N];
56
57 /*
58 * Todo array possui uma propriedade chamada length
59 * (comprimento) que é inicializada com o tamanho
60 * do array durante a criação do objeto.
61 */
62 for ( int i = 0; i < array1.length; i++ ) {
63     System.out.print( array1[i] + " " );
64 }
65 System.out.println();
66
67 for ( int i = 0; i < array2.length; i++ ) {
68     System.out.print( array2[i] + " " );
69 }
70 System.out.println();
71
72 for ( int i = 0; i < array3.length; i++ ) {
73     System.out.print( array3[i] + " " );
74 }
75 System.out.println();
76
77 for ( int i = 0; i < array4.length; i++ ) {
78     System.out.print( array4[i] + " " );
79 }
80
81 }
82
83 }
```

Entrada de dados em arrays

```
1  /*
2   * Arquivo: capitulo04/ArraysUnidimensionaisEntradaDados.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6 import java.util.Scanner;
7
8 public class ArraysUnidimensionaisEntradaDados {
9
10    public static void main( String[] args ) {
11
12        // declaração de um array de inteiros de 5 posições
13        int[] array = new int[5];
14        Scanner scan = new Scanner( System.in );
15
16        // lista os dados do array (inicializados com 0)
17        for ( int i = 0; i < array.length; i++ ) {
18            System.out.print( array[i] + " " );
19        }
20        System.out.println();
21
22        // inserção dos dados
23        for ( int i = 0; i < array.length; i++ ) {
24            System.out.printf( "Entre com o valor da posicao %d: ", i );
25            array[i] = Integer.parseInt( scan.nextLine() );
26        }
27
28        System.out.print( "Dados inseridos: " );
29        for ( int i = 0; i < array.length; i++ ) {
30            System.out.print( array[i] + " " );
31        }
32
33        scan.close();
34    }
35}
36
37 }
```

enhanced for (for melhorado) ou for each)

```
1  /*
2   * Arquivo: capitulo04/ArraysUnidimensionaisForMelhorado.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6  public class ArraysUnidimensionaisForMelhorado {
7
8      public static void main( String[] args ) {
9
10         int[] array = { 2, 4, 6, 8, 10 };
11
12         /*
13          * Além da estrutura de repetição for tradicional, existe
14          * também na linguagem Java um outro tipo tipo de for
15          * chamado de "enhanced for" (for melhorado) ou "for each".
16          *
17          * Esse for é usado para iterar sobre coleções de itens iteráveis.
18          * os arrays são estruturas iteráveis em Java, assim como outras
19          * construções que veremos adiante e depois estudaremos
20          * na disciplina de estruturas de dados.
21          *
22          * Veja os trechos de código abaixo, onde ambos serão usados para
23          * apresentar os dados do array criado acima.
24          */
25
26         System.out.println( "Usando o for melhorado: " );
27         for ( int valor : array ) {
28             // cada item de array será atribuído à valor a cada iteração
29             System.out.print( valor + " " );
30         }
31
32         System.out.println();
33
34         /*
35          * For tradicional escrito de forma a mostrar como o for melhorado
36          * funciona
37          */
38         System.out.println( "Usando o for tradicional: " );
39         for ( int i = 0; i < array.length; i++ ) {
40             int valor = array[i];
41             System.out.print( valor + " " );
42         }
43
44         /*
45          * Perceba que o for melhorado é usado primariamente para
46          * processar cada elemento do array, item por item, em ordem,
47          * mas sem se preocupar exatamente com a posição do mesmo.
48          */
```

```
48 * No for tradicional, apesar de ter que escrever mais código,  
49 * você tem acesso à posição, visto que é o objetivo da variável  
50 * i controlar esse processo.  
51 *  
52 * Sendo assim, quando apenas quiser processar os elementos do  
53 * array e não precisar saber em qual posição está no momento,  
54 * o for melhorado basta e é mais enxuto. se precisar saber a  
55 * posição atual do elemento, ai você vai precisar usar o for  
56 * tradicional. note também que, obviamente, você poderia ter  
57 * o controle da posição no for melhorado tendo uma variável  
58 * que inicia em zero e é incrementada a cada iteração, mas isso  
59 * acaba não sendo tão útil, visto que com o for tradicional você  
60 * já obtém esse comportamento.  
61 */  
62  
63    }  
64  
65 }
```

4.2 Representação Gráfica de Arrays

Os dados armazenados nos arrays são organizados de forma contígua na memória, ou seja, cada um dos elementos está situado “lado a lado” em endereços de memória adjacentes, tornando rápido o acesso à cada posição. O padrão de indexação dos elementos do array é algo que gera uma certa confusão para algumas pessoas que estão aprendendo a programar e isso é super normal. Por exemplo, se um array tem capacidade para armazenar 10 elementos, ou seja, seu tamanho é igual a 10, o primeiro elemento estará sempre na posição 0 enquanto o último na posição 9. Isso acontecerá na maioria das linguagens de programação que você terá contato na sua vida profissional, pois a grande parte dessas linguagens são baseadas direta ou indiretamente na linguagem de programação Java. Veja o exemplo gráfico de um array de uma dimensão na Figura 4.1. Usualmente utilizamos uma variável nomeada como `i` para acessar as posições de um array de uma dimensão.

```
int[] a = new int[10];
```

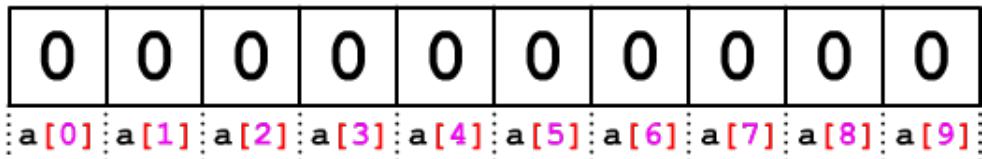


Figura 4.1: Indexação dos Arrays

⚠ | Atenção!

De modo geral, para um array de n elementos, o primeiro elemento está situado na posição 0 e o último na posição $n - 1$.

Vamos aos exercícios!

4.3 Exercícios

Exercício 4.1:

Escreva um programa que preencha um array de números inteiros de 5 posições a partir de números fornecidos pelo usuário. O programa deve armazenar em um segundo array o cubo de cada elemento do primeiro array. Por fim, o programa deve exibir os valores do array que contém o cubo do primeiro array.

Arquivo com a solução: Exercicio4\$1.java

Entrada

```
array[0]: 4  
array[1]: 5  
array[2]: 7  
array[3]: 10  
array[4]: -8
```

Saída

```
arrayCubo[0] = 64  
arrayCubo[1] = 125  
arrayCubo[2] = 343  
arrayCubo[3] = 1000  
arrayCubo[4] = -512
```

Exercício 4.2:

Escreva um programa que preencha um array de números inteiros de 5 posições com valores fornecidos pelo usuário. Logo em seguida, o programa deve pedir o valor de um número inteiro. O programa deve multiplicar cada um dos valores do array inicial pelo valor fornecido e armazenar, em um segundo array, também de 5 posições, o valor da multiplicação de cada posição do array inicial pelo valor fornecido após a leitura do primeiro array. Por fim, o programa deve exibir os valores do array que contém a multiplicação de cada item.

Arquivo com a solução: Exercicio4\$2.java

Entrada

```
array[0] : 4  
array[1] : 5  
array[2] : 7  
array[3] : 10  
array[4] : -8  
Multiplicar por: 5
```

Saída

```
arrayMult[0] = 20  
arrayMult[1] = 25  
arrayMult[2] = 35  
arrayMult[3] = 50  
arrayMult[4] = -40
```

Exercício 4.3:

Escreva um programa que preencha um array de números decimais de 5 posições com valores fornecidos pelo usuário. Após o preenchimento, o programa deve percorrer o array com os dados fornecidos, calculando o somatório e o produtório dos valores contidos no mesmo. Esses resultados devem ser exibidos ao final da execução do programa e devem estar formatados usando duas casas decimais de precisão. Lembrando que:

- $S = \sum_{i=0}^{n-1} a[i]$
- $P = \prod_{i=0}^{n-1} a[i]$
- Onde:
 - S é o somatório dos n elementos do array `a`;
 - P é o produtório dos n elementos do array `a`.

Arquivo com a solução: Exercicio4\$3.java

Entrada

```
array[0] : 4
array[1] : 5.5
array[2] : 7
array[3] : 10.7
array[4] : -8
```

Saída

```
Somatorio: 19.20
Produtorio: -13182.40
```

Exercício 4.4:

Escreva um programa que preencha um array de números inteiros de 5 posições com valores fornecidos pelo usuário. Logo em seguida, o programa deve pedir o valor de um número inteiro. O programa deve apresentar ao usuário a mensagem “ACHEI” caso o valor seja encontrado em um determinado índice (posição) ou “NAO ACHEI” caso contrário.

Arquivo com a solução: Exercicio4\$4.java

Entrada

```
array[0]: 4  
array[1]: 5  
array[2]: 7  
array[3]: 10  
array[4]: -8  
Buscar por: 5
```

Saída

```
Indice 0: NAO ACHEI  
Indice 1: ACHEI  
Indice 2: NAO ACHEI  
Indice 3: NAO ACHEI  
Indice 4: NAO ACHEI
```

Exercício 4.5:

Escreva um programa que preencha um array de números inteiros de 5 posições com valores fornecidos pelo usuário. Logo em seguida, o programa deve pedir o valor de um número inteiro. O programa deve contar quantas ocorrências do número fornecido foram encontradas no array, apresentando, ao final, essa contagem.

Arquivo com a solução: Exercicio4\$5.java

Entrada

```
array[0]: 4  
array[1]: 5  
array[2]: 7  
array[3]: 10  
array[4]: -8  
Buscar por: 5
```

Saída

```
0 array contem 1 ocorrencia do valor 5.
```

Entrada

```
array[0]: 7  
array[1]: 5  
array[2]: 7  
array[3]: 10  
array[4]: 7  
Buscar por: 7
```

Saída

```
0 array contem 3 ocorrencias do valor 7.
```

Entrada

```
array[0]: 4  
array[1]: 5  
array[2]: 7  
array[3]: 10  
array[4]: -8  
Buscar por: 0
```

Saída

```
0 array nao contem o valor 0.
```

Exercício 4.6:

Escreva um programa que preencha um array de números inteiros de 5 posições com valores fornecidos pelo usuário. Logo em seguida, o programa deve pedir o valor de um número inteiro. O programa deve apresentar ao usuário todos os índices (posições) em que o valor fornecido foi encontrado no array.

Arquivo com a solução: Exercicio4\$6.java

Entrada

```
array[0]: 4  
array[1]: 5  
array[2]: 7  
array[3]: 10  
array[4]: -8  
Buscar por: 5
```

Saída

```
O valor 5 foi encontrado no indice 1 do array.
```

Entrada

```
array[0]: 9  
array[1]: 5  
array[2]: 7  
array[3]: 9  
array[4]: 7  
Buscar por: 9
```

Saída

```
O valor 9 foi encontrado nos indices 0 e 3 do array.
```

Entrada

```
array[0]: 7  
array[1]: 5  
array[2]: 7  
array[3]: 10  
array[4]: 7  
Buscar por: 7
```

Saída

```
O valor 7 foi encontrado nos indices 0, 2 e 4 do array.
```

Entrada

```
array[0]: 4  
array[1]: 5  
array[2]: 7  
array[3]: 10  
array[4]: -8  
Buscar por: 0
```

Saída

```
0 array nao contem o valor 0.
```

Exercício 4.7:

Escreva um programa que preencha dois arrays de números inteiros de 5 posições com valores fornecidos pelo usuário. O programa deve preencher um terceiro array com a soma dos dois arrays preenchidos previamente e então exibir o array que contém a soma.

Arquivo com a solução: Exercicio4\$7.java

Entrada

Forneca os valores do primeiro array:

```
array1[0]: 5  
array1[1]: 8  
array1[2]: 7  
array1[3]: 2  
array1[4]: 8
```

Forneca os valores do segundo array:

```
array2[0]: 12  
array2[1]: -10  
array2[2]: -5  
array2[3]: -20  
array2[4]: 9
```

Saída

```
arraySoma[0] = 17  
arraySoma[1] = -2  
arraySoma[2] = 2  
arraySoma[3] = -18  
arraySoma[4] = 17
```

Exercício 4.8:

Escreva um programa que preencha um array de números inteiros de 5 posições com valores fornecidos pelo usuário. O programa deve exibir os números pares desse array e depois os números ímpares, todos na ordem em que aparecem no array.

Arquivo com a solução: Exercicio4\$8.java

Entrada

```
array[0]: 4  
array[1]: 5  
array[2]: 7  
array[3]: 10  
array[4]: -8
```

Saída

```
Numeros pares: 4 10 -8.  
Numeros impares: 5 7.
```

Entrada

```
array[0]: 4  
array[1]: 8  
array[2]: 6  
array[3]: 10  
array[4]: -8
```

Saída

```
Numeros pares: 4 8 6 10 -8.  
Numeros impares: nao ha.
```

Entrada

```
array[0]: 5  
array[1]: 9  
array[2]: 7  
array[3]: 13  
array[4]: -9
```

Saída

```
Numeros pares: nao ha.  
Numeros impares: 5 9 7 13 -9.
```

Exercício 4.9:

Escreva um programa que preencha um array de números inteiros de 5 posições com valores fornecidos pelo usuário. O programa deve copiar os valores desse array para um segundo array, sendo que no segundo array, os valores serão inseridos de forma inversa. Ao final, o programa deve exibir os valores do array invertido.

Arquivo com a solução: Exercicio4\$9.java

Entrada

```
array[0] : 4  
array[1] : 8  
array[2] : 6  
array[3] : 10  
array[4] : -8
```

Saída

```
arrayInv[0] = -8  
arrayInv[1] = 10  
arrayInv[2] = 6  
arrayInv[3] = 8  
arrayInv[4] = 4
```

Exercício 4.10:

Escreva um programa que preencha um array de números inteiros de 5 posições com valores fornecidos pelo usuário. Logo em seguida, o programa deve pedir o valor de um número inteiro. O programa deve copiar para um segundo array, todos os valores do primeiro array que são maiores que o último valor fornecido. Ao final, o programa deve exibir esses valores.

Arquivo com a solução: Exercicio4\$10.java

Entrada

```
array[0]: 4  
array[1]: 5  
array[2]: 7  
array[3]: 10  
array[4]: -8  
Copiar maiores que: 5
```

Saída

```
arrayCopia[0] = 7  
arrayCopia[1] = 10
```

Entrada

```
array[0]: 4  
array[1]: 5  
array[2]: 7  
array[3]: 10  
array[4]: -8  
Copiar maiores que: 100
```

Saída

```
Nao houve copia!
```

Exercício 4.11:

Escreva um programa para ler a quantidade de elementos que serão armazenados em um array de 10 posições de números inteiros. O programa deve aceitar apenas valores entre 1, inclusive, e 9, inclusive. Caso seja fornecido um valor incorreto, ou seja, fora desse intervalo, o programa deve requisitar novamente a entrada da quantidade. Após a leitura de uma quantidade válida, ele deve ler a quantidade de elementos informados, armazenando-os no array a partir da primeira posição. Logo em seguida, o programa deve pedir o valor de um número inteiro. Esse número deve ser inserido na primeira posição do array. Antes da inserção, perceba que há a necessidade de deslocar os elementos existentes para a casa à direita. Por fim, o programa deve imprimir o array após o deslocamento e a inclusão.

Arquivo com a solução: Exercicio4\$11.java

Entrada

```
Quantidade de elementos (1 a 9): 20
Quantidade incorreta, forneca novamente!
Quantidade de elementos (1 a 9): 5
array[0]: 4
array[1]: 5
array[2]: 7
array[3]: 10
array[4]: -8
Valor que sera inserido: 15
```

Saída

```
array[0] = 15
array[1] = 4
array[2] = 5
array[3] = 7
array[4] = 10
array[5] = -8
```

Exercício 4.12:

Escreva um programa que preencha um array de números inteiros de 5 posições com valores fornecidos pelo usuário. O primeiro elemento do array deve ser “excluído”, deslocando para isso todos os elementos a partir da segunda posição para a esquerda. Por fim, o programa deve imprimir o array após a remoção.

Arquivo com a solução: Exercicio4\$12.java

Entrada

```
array[0] : 4  
array[1] : 5  
array[2] : 7  
array[3] : 10  
array[4] : -8
```

Saída

```
array[0] = 5  
array[1] = 7  
array[2] = 10  
array[3] = -8
```

Exercício 4.13:

Escreva um programa que preencha um array de números inteiros de 10 posições com valores fornecidos pelo usuário. Em seguida, o programa deve ler o índice de uma posição do array, ou seja, um valor de 0 a 9. Caso seja informado uma posição inválida, o programa deve informar o usuário e pedir novamente a posição. Após a leitura da posição válida, o programa deve “remover” do array o elemento contido na posição fornecida. Por fim, o programa deve imprimir o array após a remoção.

Arquivo com a solução: Exercicio4\$13.java

Entrada

```
array[0]: 4
array[1]: 5
array[2]: 7
array[3]: 10
array[4]: -8
array[5]: 9
array[6]: 10
array[7]: 14
array[8]: 3
array[9]: 121
Posicao a ser removida (0 a 9): 20
Posicao invalida, forneca novamente!
Posicao a ser removida (0 a 9): 5
```

Saída

```
array[0] = 4
array[1] = 5
array[2] = 7
array[3] = 10
array[4] = -8
array[5] = 10
array[6] = 14
array[7] = 3
array[8] = 121
```

Exercício 4.14:

Escreva um programa que preencha um array de números inteiros de 10 posições com valores fornecidos pelo usuário. O programa deve remover do array todos os elementos que forem pares. Por fim, o programa deve imprimir o array após as remoções.

Arquivo com a solução: Exercicio4\$14.java

Entrada

```
array[0]: 4
array[1]: 5
array[2]: 7
array[3]: 10
array[4]: -8
array[5]: 9
array[6]: 10
array[7]: 14
array[8]: 3
array[9]: 121
```

Saída

```
array[0] = 5
array[1] = 7
array[2] = 9
array[3] = 3
array[4] = 121
```

Exercício 4.15:

Escreva um programa que preencha dois arrays de números inteiros de 5 posições com valores fornecidos pelo usuário. Um terceiro array deve ser preenchido, contendo a intersecção dos elementos contidos nos dois primeiros arrays, ou seja, os valores que são comuns aos dois. Nos dois arrays fornecidos, pode haver repetição de elementos, mas essa repetição não deve ser refletida no array de intersecção. Por fim, o programa deve imprimir o array que contém os valores comuns aos dois arrays fornecidos.

Arquivo com a solução: Exercicio4\$15.java

Entrada

Forneca os valores do primeiro array:

```
array1[0]: 5  
array1[1]: 8  
array1[2]: 7  
array1[3]: 2  
array1[4]: 8
```

Forneca os valores do segundo array:

```
array2[0]: 5  
array2[1]: -10  
array2[2]: -5  
array2[3]: 8  
array2[4]: 2
```

Saída

```
arrayInterseccao[0] = 5  
arrayInterseccao[1] = 8  
arrayInterseccao[2] = 2
```

Entrada

Forneca os valores do primeiro array:

```
array1[0]: 5  
array1[1]: 8  
array1[2]: 7  
array1[3]: 2  
array1[4]: 8
```

Forneca os valores do segundo array:

```
array2[0]: 12  
array2[1]: -10  
array2[2]: -5  
array2[3]: -20  
array2[4]: 9
```

Saída

Nao ha interseccao entre os elementos dos dois arrays fornecidos!

4.4 Exercícios Criativos

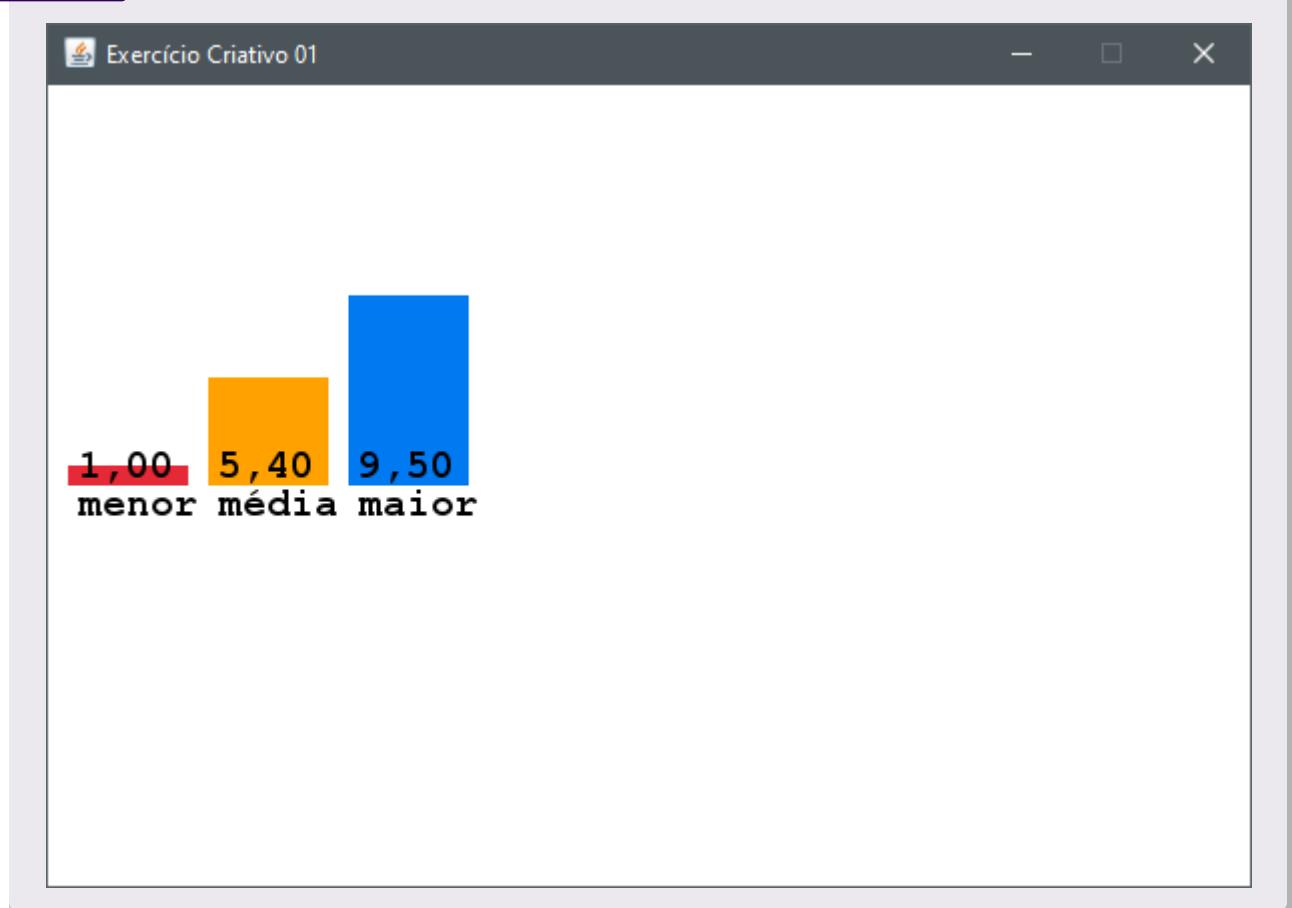
Exercício Criativo 4.1:

Escreva um programa que leia cinco valores decimais que representam a nota final de cinco alunos em uma disciplina. O programa deve apresentar graficamente qual a menor nota, qual a maior nota e qual a média das cinco notas. Obviamente, as notas devem ser armazenadas em um array de cinco posições. Na representação, as notas abaixo de 4.0 devem usar uma cor indicando reprovação, notas no intervalo [4,0..6,0[devem usar uma outra cor, indicando “exame”, e notas maiores ou iguais a 6,0 devem usar uma terceira cor, indicando aprovação. Você está livre para usar as cores que quiser. A única restrição é que sua tela deve ter dimensões 600x400 pixels.

Entrada

```
nota 1: 1  
nota 2: 9.5  
nota 3: 4  
nota 4: 5  
nota 5: 7.5
```

Saída:

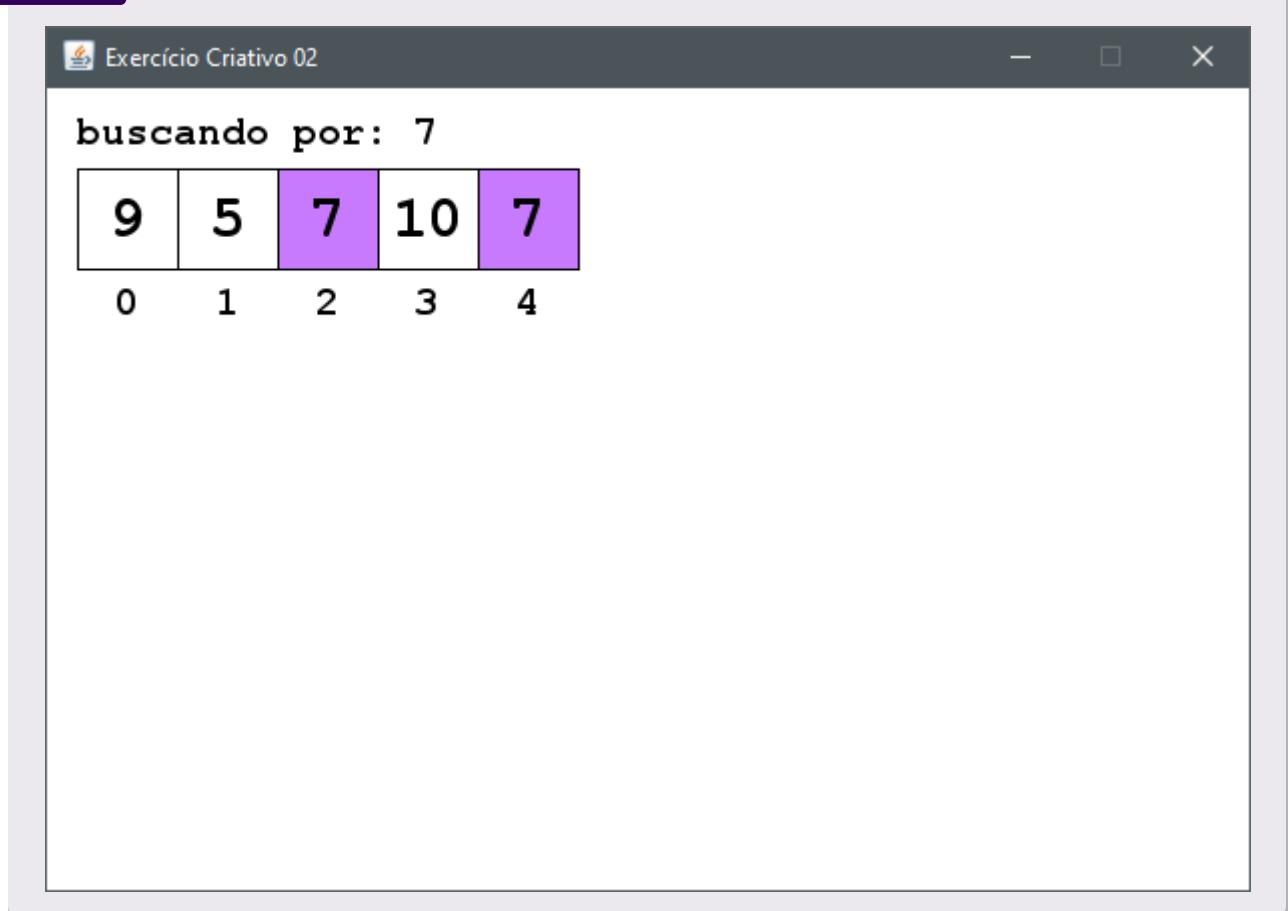


Exercício Criativo 4.2:

Escreva um programa que leia cinco valores inteiros, armazenando-os em um array de cinco posições. Após a leitura dos valores do array, o programa deve ler mais um valor que será buscado. Apresente graficamente o resultado da busca, destacando as posições em que o valor buscado foi encontrado. Você está livre para usar as cores que quiser. A única restrição é que sua tela deve ter dimensões 600x400 pixels.

Entrada

```
array[0] : 9  
array[1] : 5  
array[2] : 7  
array[3] : 10  
array[4] : 7  
buscar por: 7
```

Saída:

4.5 Desafios

Desafio 4.1:

Escreva um programa que preencha um array de números inteiros de 10 posições com valores fornecidos pelo usuário. O programa deve exibir os números pares desse array em ordem crescente e depois os números ímpares em ordem decrescente.

Arquivo com a solução: Desafio4\$1.java

Entrada

```
array[0]: 12
array[1]: 7
array[2]: 5
array[3]: 10
array[4]: -8
array[5]: 121
array[6]: 10
array[7]: 14
array[8]: 3
array[9]: 9
```

Saída

```
Valores pares em ordem crescente: -8 10 10 12 14.
Valores ímpares em ordem decrescente: 121 9 7 5 3.
```

Entrada

```
array[0]: 8
array[1]: 2
array[2]: 14
array[3]: 16
array[4]: 8
array[5]: 12
array[6]: 2
array[7]: 22
array[8]: 6
array[9]: 44
```

Saída

```
Valores pares em ordem crescente: 2 2 6 8 8 12 14 16 22 44.
Valores ímpares em ordem decrescente: nao ha.
```

Entrada

```
array[0]: 9  
array[1]: 7  
array[2]: 3  
array[3]: 1  
array[4]: 5  
array[5]: 75  
array[6]: 9  
array[7]: 15  
array[8]: 13  
array[9]: 27
```

Saída

Valores pares em ordem crescente: nao ha.

Valores impares em ordem decrescente: 75 27 15 13 9 9 7 5 3 1.

ARRAYS MULTIDIMENSIONAIS

“Se as pessoas não acreditam que a Matemática é simples, é só porque não percebem o quanto complicada é a vida”.

John von Neumann



S arrays multidimensionais permitem que os dados sejam armazenados em mais de uma dimensão. Neste capítulo serão apresentados esses arrays, que na memória são armazenados de forma contígua, assim como nos arrays de uma dimensão, mas que podem ter uma interpretação geométrica para que seja facilitada sua visualização.

Na grande maioria das vezes utilizaremos arrays de duas dimensões, algumas vezes chamados erroneamente de matrizes. Essa nomenclatura errada dos arrays se dá, pois muitas vezes os dados armazenados nos mesmos são dados de vetores ou matrizes da álgebra, mas o nome da estrutura é, de fato, array ou arranjo.

5.1 Exemplos em Linguagem Java

Declaração e inicialização de arrays multidimensionais

```
1  /*
2   * Arquivo: capitulo05/Arrays2DDeclaracaoInicializacao.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6 public class Arrays2DDeclaracaoInicializacao {
7
8     public static void main( String[] args ) {
9
10         // definição de constantes
11         final int M = 5;
12         final int N = 2;
13
14         /*
15          * Declaração de um array de duas dimensões de inteiros
16          * e inicialização com um objeto do tipo array de duas
17          * dimensões com 3 linhas e 2 colunas.
18          */
19         int[][] array1 = new int[3][2];
20
21         /*
22          * Em Java, os arrays de mais de uma dimensão são na verdade
23          * arrays de arrays. por exemplo, o array acima é um array de
24          * três posições que contém, em cada uma delas, um array de duas
25          * posições. Essa mesma ideia se aplica a arrays com mais dimensões
26          */
27
28         /*
29          * Declarando um array de dimensões 2x2 e inicializando
30          * com valores 2
31          * valor inicializado = { { 2, 2 },
32          *                         { 2, 2 } }
33          */
34         int[][] array2 = { { 2, 2 }, { 2, 2 } };
35
36         /*
37          * Declarando um array de dimensões 2x3 e inicializando
38          * com valores
39          * valor inicializado = { { 3, 3, 3 },
40          *                         { 0, 0, 0 } }
41          */
42         int[][] array3 = new int[][]{
43             new int[]{ 3, 3, 3 },
44             new int[]{ 0, 0, 0 }
45         };
46 }
```

```
46
47     // declaração de um array de inteiros de dimensões MxN
48     int[][] array4 = new int[M][N];
49
50
51     /*
52      * i será usado normalmente para controlar a linha atual
53      * e j será usado normalmente para controlar a coluna atual
54      * em um array bidimensional.
55     */
56
57     for ( int i = 0; i < array1.length; i++ ) {
58         for ( int j = 0; j < array1[i].length; j++ ) {
59             System.out.print( array1[i][j] + " " );
60         }
61         System.out.println();
62     }
63     System.out.println();
64
65     for ( int i = 0; i < array2.length; i++ ) {
66         for ( int j = 0; j < array2[i].length; j++ ) {
67             System.out.print( array2[i][j] + " " );
68         }
69         System.out.println();
70     }
71     System.out.println();
72
73     for ( int i = 0; i < array3.length; i++ ) {
74         for ( int j = 0; j < array3[i].length; j++ ) {
75             System.out.print( array3[i][j] + " " );
76         }
77         System.out.println();
78     }
79     System.out.println();
80
81     for ( int i = 0; i < array4.length; i++ ) {
82         for ( int j = 0; j < array4[i].length; j++ ) {
83             System.out.print( array4[i][j] + " " );
84         }
85         System.out.println();
86     }
87
88 }
89
90 }
```

Entrada de dados em arrays multidimensionais

```
1  /*
2   * Arquivo: capitulo05/Arrays2DEntradaDados.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6 import java.util.Scanner;
7
8 public class Arrays2DEntradaDados {
9
10    public static void main( String[] args ) {
11
12        // declaração de um array de inteiros de dimensões 2x3
13        int[][] array = new int[2][3];
14        Scanner scan = new Scanner( System.in );
15
16        // lista os dados do array (inicializados com 0)
17        for ( int i = 0; i < array.length; i++ ) {
18            for ( int j = 0; j < array[i].length; j++ ) {
19                System.out.print( array[i][j] + " " );
20            }
21            System.out.println();
22        }
23        System.out.println();
24
25        // inserção dos dados
26        for ( int i = 0; i < array.length; i++ ) {
27            for ( int j = 0; j < array[i].length; j++ ) {
28                System.out.printf(
29                    "Entre com o valor da posicao [%d][%d]: ", i, j );
30                array[i][j] = Integer.parseInt( scan.nextLine() );
31            }
32        }
33
34        System.out.println( "Dados inseridos:" );
35        for ( int i = 0; i < array.length; i++ ) {
36            for ( int j = 0; j < array[i].length; j++ ) {
37                System.out.print( array[i][j] + " " );
38            }
39            System.out.println();
40        }
41
42        scan.close();
43
44    }
45
46 }
```

5.2 Representação Gráfica de Arrays Multidimensionais

Na Figura 5.1 pode-se ver a representação gráfica de um array de duas dimensões como um reticulado. Usualmente utilizamos uma variável nomeada como i para acessar as “linhas” dessa estrutura bidimensional e uma variável nomeada como j para acessar as “colunas”. Além disso, convencionou-se que a primeira dimensão é a que corresponde às linhas enquanto que a segunda é a associada às colunas. Por exemplo, se a é um array bidimensional, $a[2][3]$ refere-se ao elemento situado na quarta coluna da terceira linha. Lembre-se da indexação iniciada em zero!

```
int[][] a = new int[5][5];
```

i	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

Labels below the table:

- Row 0: $a[0][0], a[0][1], a[0][2], a[0][3], a[0][4]$
- Row 1: $a[1][0], a[1][1], a[1][2], a[1][3], a[1][4]$
- Row 2: $a[2][0], a[2][1], a[2][2], a[2][3], a[2][4]$
- Row 3: $a[3][0], a[3][1], a[3][2], a[3][3], a[3][4]$
- Row 4: $a[4][0], a[4][1], a[4][2], a[4][3], a[4][4]$

Figura 5.1: Indexação dos Arrays de Duas Dimensões

Já um array de três dimensões pode ser interpretado como um paralelepípedo reto/retângulo. Veja a Figura 5.2. Agora a primeira dimensão corresponde à profundidade desse paralelepípedo, a segunda às linhas de cada reticulado e a terceira dimensão às colunas, ou seja, se a é um array tridimensional, $a[2][3][1]$ refere-se ao elemento situado na segunda coluna, da quarta linha do terceiro reticulado. Raramente você precisará dessa quantidade de dimensões, muito menos de quantidades maiores, apesar de permitidas. Fica aí um exercício mental. Tente imaginar um array de quatro ou cinco dimensões geometricamente.

```
int[][][] a = new int[3][3][3];
```

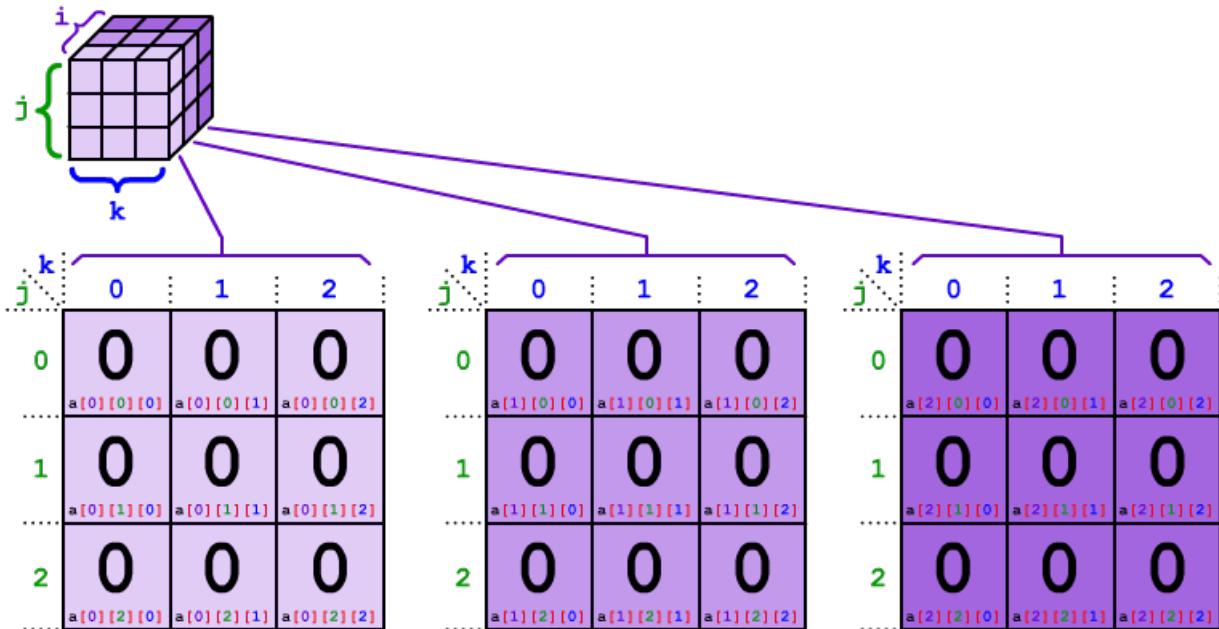


Figura 5.2: Indexação dos Arrays de Três Dimensões

5.3 Exercícios

Exercício 5.1:

Escreva um programa que preencha um array de dimensões 3x2 de inteiros com valores fornecidos pelo usuário e o exiba na forma de uma matriz.

Arquivo com a solução: Exercicio5\$1.java

Entrada

```
array[0][0]: 1
array[0][1]: 2
array[1][0]: 3
array[1][1]: 4
array[2][0]: 5
array[2][1]: 6
```

Saída

```
001 002
003 004
005 006
```

Exercício 5.2:

Escreva um programa que preencha dois arrays de dimensões 3x3 de inteiros com valores fornecidos pelo usuário e armazene a soma desses dois arrays em um terceiro array de dimensões 3x3. No final, o programa deve exibir os três arrays no formato apresentado a seguir.

Arquivo com a solução: Exercicio5\$2.java

Entrada

```
array1[0][0]: 4
array1[0][1]: 7
array1[0][2]: 8
array1[1][0]: 5
array1[1][1]: 1
array1[1][2]: 2
array1[2][0]: 6
array1[2][1]: 5
array1[2][2]: 8
array2[0][0]: 9
array2[0][1]: 5
array2[0][2]: 2
array2[1][0]: 1
array2[1][1]: 4
array2[1][2]: 5
array2[2][0]: 6
array2[2][1]: 3
array2[2][2]: 2
```

Saída

```
array1:*****array2:*****arraySoma:
004 007 008***009 005 002***013 012 010
005 001 002 + 001 004 005 = 006 005 007
006 005 008***006 003 002***012 008 010
```

Exercício 5.3:

Escreva um programa que preencha um array de dimensões 3x4 de inteiros com valores fornecidos pelo usuário. Em seguida, o programa deve ler o valor de um número inteiro. Armazene em um segundo array de dimensões 3x4 a multiplicação do valor fornecido pelas posições do array preenchido inicialmente. No final, o programa deve exibir o array contendo a multiplicação na forma de uma matriz.

Arquivo com a solução: Exercicio5\$3.java

Entrada

```
array[0] [0]: 1
array[0] [1]: 4
array[0] [2]: 5
array[0] [3]: 8
array[1] [0]: 7
array[1] [1]: 4
array[1] [2]: 5
array[1] [3]: 2
array[2] [0]: 3
array[2] [1]: 6
array[2] [2]: 5
array[2] [3]: 4
Multiplicar por: 5
```

Saída

```
arrayMult:
005 020 025 040
035 020 025 010
015 030 025 020
```

Exercício 5.4:

Escreva um programa que preencha um array de dimensões 2x2 de inteiros com valores fornecidos pelo usuário. O programa deve calcular e exibir o determinante da matriz representada por esse array. Lembrando que:

- Para $M_{2x2} = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix}$,
- $D = a_{1,1} \cdot a_{2,2} - (a_{1,2} \cdot a_{2,1})$
- Onde:
 - M_{2x2} é uma matriz de dimensões 2x2;
 - D é o determinante dessa matriz;
 - **Obs:** cuidado com os índices do array, pois iniciam em 0 e não 1.

Arquivo com a solução: Exercicio5\$4.java

Entrada

```
array[0][0]: 4
array[0][1]: 5
array[1][0]: 6
array[1][1]: 1
```

Saída

```
Determinante: -26
```

Exercício 5.5:

Escreva um programa que preencha um array de dimensões 3x3 de inteiros com valores fornecidos pelo usuário. O programa deve calcular e exibir o determinante da matriz representada por esse array. Lembrando que:

- Para $M_{3 \times 3} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}$,
- $D = \frac{a_{1,1} \cdot a_{2,2} \cdot a_{3,3} + a_{1,2} \cdot a_{2,3} \cdot a_{3,1} + a_{1,3} \cdot a_{2,1} \cdot a_{3,2} - (a_{1,3} \cdot a_{2,2} \cdot a_{3,1} + a_{1,1} \cdot a_{2,3} \cdot a_{3,2} + a_{1,2} \cdot a_{2,1} \cdot a_{3,3})}{(a_{1,3} \cdot a_{2,2} \cdot a_{3,1} + a_{1,1} \cdot a_{2,3} \cdot a_{3,2} + a_{1,2} \cdot a_{2,1} \cdot a_{3,3})}$
- Onde:
 - $M_{3 \times 3}$ é uma matriz de dimensões 3x3;
 - D é o determinante dessa matriz;
 - **Obs:** cuidado com os índices do array, pois iniciam em 0 e não 1.

Arquivo com a solução: Exercicio5\$5.java

Entrada

```
array[0][0]: 4
array[0][1]: 5
array[0][2]: 7
array[1][0]: 8
array[1][1]: 2
array[1][2]: 1
array[2][0]: 3
array[2][1]: 6
array[2][2]: 5
```

Saída

```
Determinante: 125
```

Exercício 5.6:

Escreva um programa que preencha um array de dimensões 2x3 de inteiros com valores fornecidos pelo usuário. Esse array será considerado como uma matriz. O programa deve preencher um segundo array de dimensões 3x2 com os valores que representem a matriz transposta da matriz contida no primeiro array. Por fim, o programa deve exibir a matriz original e a matriz transposta. Lembrando que:

- Para $M_{2 \times 3} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \end{bmatrix}$,
- $M^t = \begin{bmatrix} a_{1,1} & a_{2,1} \\ a_{1,2} & a_{2,2} \\ a_{1,3} & a_{2,3} \end{bmatrix}$
- Onde:
 - $M_{2 \times 3}$ é uma matriz de dimensões 2x3;
 - M^t é a matriz transposta de M de dimensões 3x2;
 - **Obs:** cuidado com os índices do array, pois iniciam em 0 e não 1.

Arquivo com a solução: Exercicio5\$6.java

Entrada

```
array[0][0]: 1
array[0][1]: 2
array[0][2]: 3
array[1][0]: 4
array[1][1]: 5
array[1][2]: 6
```

Saída

```
M:
001 002 003
004 005 006
```

```
Mt:
001 004
002 005
003 006
```

Exercício 5.7:

Escreva um programa que preencha dois arrays de inteiros, um de dimensões 3x2 enquanto o outro de dimensões 2x3. As duas matrizes representadas pelos arrays devem ser multiplicadas e o resultado deve ser armazenado em um terceiro array bidimensional de dimensões 3x3. Por fim, o programa deve exibir o array que contém a multiplicação. Lembrando que:

- Para $A = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{bmatrix}$ e
- $B = \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \end{bmatrix}$,
- $A \cdot B = \begin{bmatrix} a_{1,1} \cdot b_{1,1} + a_{1,2} \cdot b_{2,1} & a_{1,1} \cdot b_{1,2} + a_{1,2} \cdot b_{2,2} & a_{1,1} \cdot b_{1,3} + a_{1,2} \cdot b_{2,3} \\ a_{2,1} \cdot b_{1,1} + a_{2,2} \cdot b_{2,1} & a_{2,1} \cdot b_{1,2} + a_{2,2} \cdot b_{2,2} & a_{2,1} \cdot b_{1,3} + a_{2,2} \cdot b_{2,3} \\ a_{3,1} \cdot b_{1,1} + a_{3,2} \cdot b_{2,1} & a_{3,1} \cdot b_{1,2} + a_{3,2} \cdot b_{2,2} & a_{3,1} \cdot b_{1,3} + a_{3,2} \cdot b_{2,3} \end{bmatrix}$
- Onde:
 - A é uma matriz de dimensões 3x2;
 - B é uma matriz de dimensões 2x3;
 - **Obs:** cuidado com os índices do array, pois iniciam em 0 e não 1.

Arquivo com a solução: Exercicio5\$7.java

Entrada

```
array1[0][0]: 2
array1[0][1]: 3
array1[1][0]: 0
array1[1][1]: 1
array1[2][0]: -1
array1[2][1]: 4
array2[0][0]: 1
array2[0][1]: 2
array2[0][2]: 3
array2[1][0]: -2
array2[1][1]: 0
array2[1][2]: 4
```

Saída

```
A x B =
-04 004 018
-02 000 004
-09 -02 013
```

Exercício 5.8 (BEECROWD, 2025):

Escreva um programa que leia um inteiro no intervalo de 1, inclusive, a 100, inclusive. Caso o valor fornecido esteja fora desse intervalo, o programa deve avisar o usuário e terminar. Caso contrário, esse valor deve ser usado para gerar uma saída de acordo com o padrão apresentado em cada exemplo. Os valores inteiros devem ser formatados alinhados à direita, com largura de três caracteres, preenchidos com espaços, usando para isso o especificador de formato `"%3d"` e separados por um espaço. Os asteriscos quase pretos indicam espaços. **Obs.:** Este exercício é uma adaptação do problema número 1435 da plataforma de desafios de programação beecrowd (<<https://www.beecrowd.com.br/judge/en/problems/view/1435>>).

Arquivo com a solução: Exercicio5\$8.java

Entrada

```
Numero entre 1 e 100: 1
```

Saída

```
**1
```

Entrada

```
Numero entre 1 e 100: 2
```

Saída

```
**1***1  
**1***1
```

Entrada

```
Numero entre 1 e 100: 3
```

Saída

```
**1***1***1  
**1***2***1  
**1***1***1
```

Entrada

```
Numero entre 1 e 100: 4
```

Saída

```
**1***1***1***1  
**1***2***2***1  
**1***2***2***1  
**1***1***1***1
```

Entrada

```
Numero entre 1 e 100: 5
```

Saída

```
**1***1***1***1***1  
**1***2***2***2***1  
**1***2***3***2***1  
**1***2***2***2***1  
**1***1***1***1***1
```

Entrada

```
Numero entre 1 e 100: 10
```

Saída

```
**1***1***1***1***1***1***1***1***1***1  
**1***2***2***2***2***2***2***2***1  
**1***2***3***3***3***3***3***3***2***1  
**1***2***3***4***4***4***4***3***2***1  
**1***2***3***4***5***5***4***3***2***1  
**1***2***3***4***5***5***4***3***2***1  
**1***2***3***4***4***4***4***3***2***1  
**1***2***3***3***3***3***3***3***2***1  
**1***2***2***2***2***2***2***2***2***1  
**1***1***1***1***1***1***1***1***1***1
```

Entrada

```
Numero entre 1 e 100: 0
```

Saída

```
Numero incorreto!
```

Entrada

```
Numero entre 1 e 100: 200
```

Saída

```
Numero incorreto!
```

Exercício 5.9 (BEECROWD, 2025):

Escreva um programa que leia um inteiro no intervalo de 1, inclusive, a 100, inclusive. Caso o valor fornecido esteja fora desse intervalo, o programa deve avisar o usuário e terminar. Caso contrário, esse valor deve ser usado para gerar uma saída de acordo com o padrão apresentado em cada exemplo. Os valores inteiros devem ser formatados alinhados à direita, com largura de três caracteres, preenchidos com espaços, usando para isso o especificador de formato `"%3d"` e separados por um espaço. Os asteriscos quase pretos indicam espaços. **Obs.:** Este exercício é uma adaptação do problema número 1478 da plataforma de desafios de programação beecrowd (<<https://www.beecrowd.com.br/judge/en/problems/view/1478>>).

Arquivo com a solução: Exercicio5\$9.java

Entrada

```
Numero entre 1 e 100: 1
```

Saída

```
**1
```

Entrada

```
Numero entre 1 e 100: 2
```

Saída

```
**1***2  
**2***1
```

Entrada

```
Numero entre 1 e 100: 3
```

Saída

```
**1***2***3  
**2***1***2  
**3***2***1
```

Entrada

```
Numero entre 1 e 100: 4
```

Saída

```
**1***2***3***4  
**2***1***2***3  
**3***2***1***2  
**4***3***2***1
```

Entrada

Numero entre 1 e 100: 5

Saída

```
**1***2***3***4***5  
**2***1***2***3***4  
**3***2***1***2***3  
**4***3***2***1***2  
**5***4***3***2***1
```

Entrada

Numero entre 1 e 100: 10

Saída

```
**1***2***3***4***5***6***7***8***9***10  
**2***1***2***3***4***5***6***7***8***9  
**3***2***1***2***3***4***5***6***7***8  
**4***3***2***1***2***3***4***5***6***7  
**5***4***3***2***1***2***3***4***5***6  
**6***5***4***3***2***1***2***3***4***5  
**7***6***5***4***3***2***1***2***3***4  
**8***7***6***5***4***3***2***1***2***3  
**9***8***7***6***5***4***3***2***1***2  
*10***9***8***7***6***5***4***3***2***1
```

Entrada

Numero entre 1 e 100: 0

Saída

Numero incorreto!

Entrada

Numero entre 1 e 100: 200

Saída

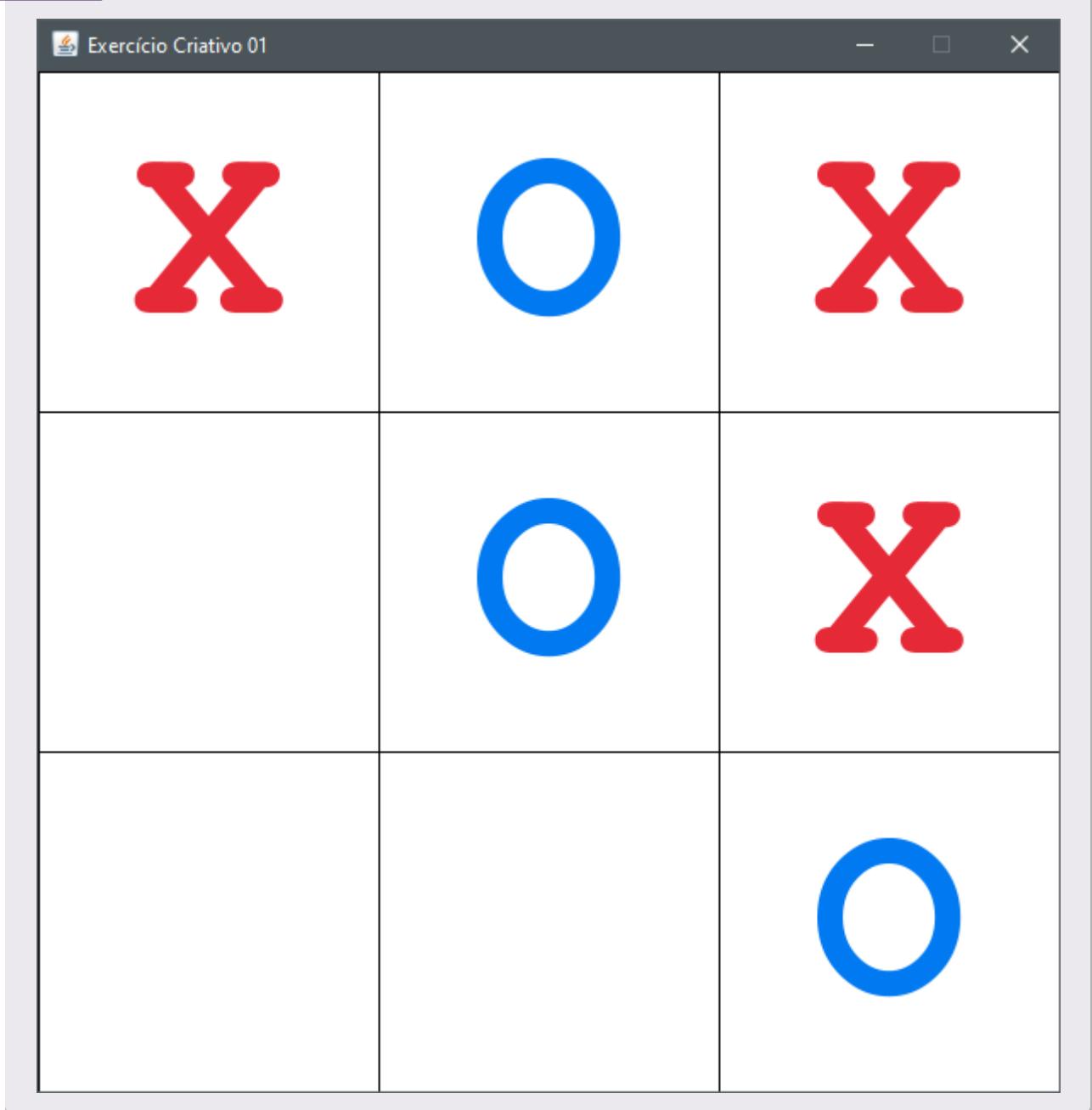
Numero incorreto!

5.4 Exercícios Criativos

Exercício Criativo 5.1:

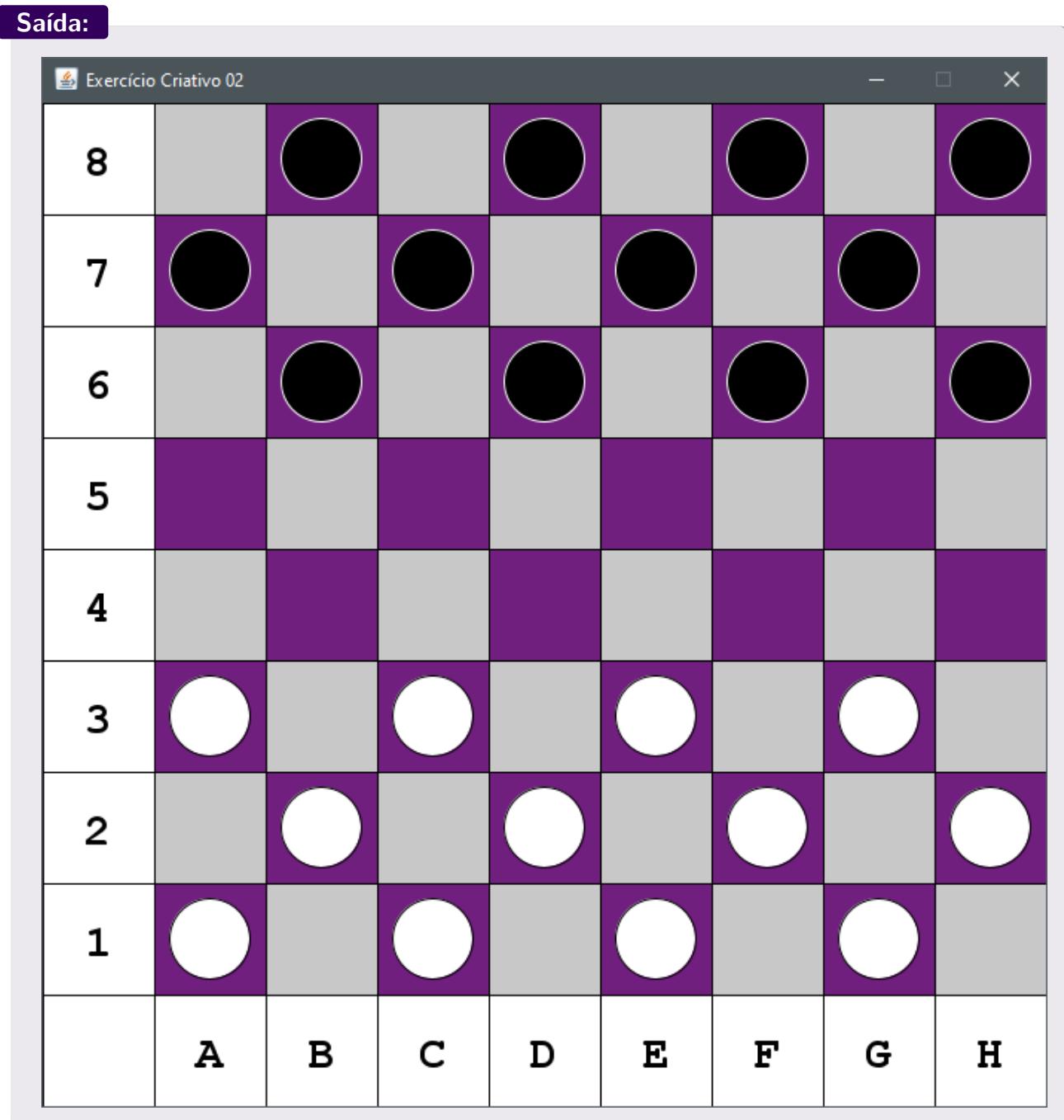
Preparado para desenvolver seu primeiro jogo? Então vamos lá! Desenvolva um “Jogo da Velha” para dois jogadores. Toda a lógica de como o jogo deve funcionar é sua responsabilidade. A única coisa que será apresentada a você é uma ideia de como pode ser a interface gráfica. Projete uma forma em que os jogadores possam fornecer suas jogadas pelo terminal.

Saída:



Exercício Criativo 5.2:

Seu próximo jogo deve ser um “Jogo de Damas” para dois jogadores. Toda a lógica de como o jogo deve funcionar é sua responsabilidade. A única coisa que será apresentada a você é uma ideia de como pode ser a interface gráfica. Projete uma forma em que os jogadores possam fornecer suas jogadas pelo terminal.



5.5 Desafios

Desafio 5.1:

Escreva um programa que preencha dois arrays de inteiros, cada um com dimensões que podem variar de 1x1 a 10x10. As dimensões dos arrays iniciais devem ser fornecidas pelo usuário e devem ser compatíveis para que o programa continue, caso contrário o programa deve terminar. As duas matrizes representadas pelos arrays devem ser multiplicadas e o resultado deve ser armazenado em um terceiro array. Por fim, o programa deve exibir o array que contém a multiplicação.

Arquivo com a solução: Desafio5\$1.java

Desafio 5.2:

Escreva um programa que preencha um array de inteiros, com dimensões que podem variar de 2x2 a 10x10. A dimensão do array inicial deve ser fornecida pelo usuário e deve ser compatível para que o programa continue, caso contrário o programa deve terminar. O programa deve calcular e exibir o determinante da matriz representada por esse array.

Arquivo com a solução: [Desafio5\\$2.java](#)

CAPÍTULO

6

CLASSE MATH E SEUS MÉTODOS MATEMÁTICOS

“A Matemática não mente. Mente quem faz mau uso dela”.

Albert Einstein



maioria das linguagens de programação modernas possuem funcionalidades que permitem a execução de diversas funções matemáticas. Neste Capítulo serão apresentadas os métodos matemáticos mais comuns que são fornecidos na linguagem de programação Java por meio da classe `Math`. Existem outros recursos implementados na plataforma para, por exemplo, a manipulação de números de precisão arbitrária, mas não lidaremos com isso nesse livro.

6.1 Exemplos em Linguagem Java

Principais métodos matemáticos em Java


```

95
96  /*
97   * Método Math.asin (arcsine): calcula o grau em radianos de um
98   * seno.
99   */
100 System.out.printf( "Math.asin(0.5) = %.2f radianos => %.2f graus\n",
101                  Math.asin(0.5), 180/Math.PI * Math.asin(0.5) );
102
103 /*
104  * Método Math.acos (arccosine): calcula o grau em radianos de
105  * um cosseno.
106  */
107 System.out.printf( "Math.acos(0.5) = %.2f radianos => %.2f graus\n",
108                  Math.acos(0.5), 180/Math.PI * Math.acos(0.5) );
109
110 /*
111  * Método Math.atan (arctangent): calcula o grau em radianos
112  * de uma tangente.
113  */
114 System.out.printf(
115     "Math.atan(1)      = %.2f radianos => %.2f graus\n\n",
116     Math.atan(1), 180/Math.PI * Math.atan(1)
117 );
118
119 /*
120  * Método Math.hypot (hypotenuse): calcula o valor da hipotenusa
121  * com base no valor dos dois catetos.
122  */
123 System.out.printf( "Math.hypot(3, 4) = %f\n", Math.hypot(3, 4) );
124
125 /*
126  * Método Math.atan2 (arctangent2): obtém o ângulo de uma
127  * coordenada cartesiana.
128  */
129 System.out.printf( "Math.atan2(4, 3) = (3; 4) cartesiano " );
130 System.out.printf( "corresponde a (%.2f, %.2f) polar\n",
131                   Math.hypot(4, 3), Math.atan2(4, 3) );
132 System.out.printf( "                                         note que %.2f radianos => ",
133                   Math.atan2(4, 3) );
134 System.out.printf(
135     "\n%.2f graus\n\n", 180/Math.PI * Math.atan2(4, 3)
136 );
137
138
139 System.out.println( "**** conversao entre graus e radianos ****" );
140
141 /*
142  * Método Math.toDegrees: converte um valor expresso em radianos
143  * para graus.

```

```
144     */
145     System.out.printf( "Math.toDegrees(pi/3) = %.2f graus\n",
146                         Math.toDegrees(Math.PI/3) );
147
148     /*
149      * Método Math.toRadians: converte um valor expresso em graus
150      * para radianos.
151     */
152     System.out.printf( "Math.toRadians(60)    = %f radianos\n\n",
153                         Math.toRadians(60) );
154
155
156     System.out.println( "**** metodos de arredondamento ****" );
157
158     /*
159      * Método Math.ceil: arredonda um número decimal para o maior
160      * inteiro mais próximo.
161     */
162     System.out.printf( "Math.ceil(+2.4)   = %.2f\n", Math.ceil(2.4) );
163     System.out.printf( "Math.ceil(-2.4)   = %.2f\n\n", Math.ceil(-2.4) );
164
165     /*
166      * Método Math.floor: arredonda um número decimal para o menor
167      * inteiro mais próximo.
168     */
169     System.out.printf( "Math.floor(+2.7) = %.2f\n", Math.floor(2.7) );
170     System.out.printf( "Math.floor(-2.7) = %.2f\n\n", Math.floor(-2.7) );
171
172     // método Math.round: arredonda para o inteiro mais próximo
173     System.out.printf( "Math.round(+2.3) = %d\n", Math.round(2.3) );
174     System.out.printf( "Math.round(+2.5) = %d\n", Math.round(2.5) );
175     System.out.printf( "Math.round(+2.7) = %d\n", Math.round(2.7) );
176     System.out.printf( "Math.round(-2.3) = %d\n", Math.round(-2.3) );
177     System.out.printf( "Math.round(-2.5) = %d\n", Math.round(-2.5) );
178     System.out.printf( "Math.round(-2.7) = %d\n", Math.round(-2.7) );
179
180     /*
181      * Não existe um método para remover a parte de decimal de um
182      * número, mas podemos fazer isso usando uma conversão
183      * explícita de tipos chada de cast ou coerção.
184     */
185     System.out.printf( "+2.7 truncado    = %d\n", (int) 2.7 );
186     System.out.printf( "-2.7 truncado    = %d\n\n", (int) -2.7 );
187
188
189     System.out.println( "**** geracao de numeros aleatorios ****" );
190
191     /*
192      * Método Math.random: retorna um valor decimal no intervalo [0, 1[.
```

```

193     */
194     System.out.printf( "Math.random() = %f\n", Math.random() );
195     System.out.println( "Intervalos:" );
196     System.out.printf( "    [0, 10[ = %d\n",
197             (int) ( Math.random() * 10 ) );
198     System.out.printf( "    [0, 10] = %d\n",
199             (int) ( Math.random() * 11 ) );
200
201 /**
202 * Para gerar valores em intervalos personalizados precisamos
203 * usar um pouco de álgebra.
204 */
205 int inicio = 5;
206 int fim = 10;
207 System.out.printf( "    [%d, %d[ = %d\n", inicio, fim,
208                     inicio + (int) ( Math.random() * ( fim - inicio ) ) );
209 System.out.printf( "    [%d, %d] = %d", inicio, fim,
210                     inicio + (int) ( Math.random() * ( fim - inicio + 1 ) ) );
211
212 }
213
214 }
```

Uma função trigonométrica interessante é a implementada no método `Math.atan2()` que é usado para converter coordenadas cartesianas em coordenadas polares. Na Figura 6.1 pode-se ver um esquema gráfico do uso dessa função. Um exemplo de uso desse método é simular um olho que acompanha/olha o cursor do mouse. Peça para o professor fazer um exemplo em aula!

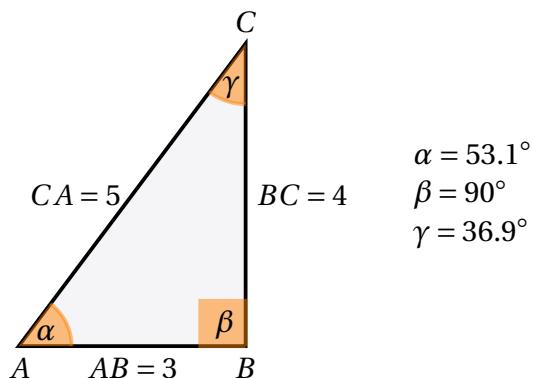


Figura 6.1: Esquema gráfico para entendimento da chamada `Math.atan2(4, 3)` que resulta em 53.1°

| Saiba Mais

A lista completa e a documentação dos métodos matemáticos da classe `Math` da linguagem Java pode ser encontrada aqui: <<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Math.html>>

Vamos aos exercícios!

6.2 Exercícios

Exercício 6.1:

Escreva um programa que peça para o usuário fornecer os coeficientes a , b e c de um polinômio do segundo grau. O programa deve calcular as duas raízes da equação do segundo grau representada por esse polinômio e apresentar o conjunto solução ($S = \{x_1, x_2\}$) ao usuário, sendo que os valores de x devem ser apresentados em ordem crescente. Caso o coeficiente a seja igual a zero, significa que não existe equação do segundo grau, então uma mensagem deve ser exibida ao usuário e o programa deve finalizar. Caso o discriminante da equação (Δ) seja menor que zero, não existem raízes reais, sendo assim, o conjunto solução é vazio. Caso seja igual a zero, as duas raízes têm o mesmo valor e apenas uma deve ser apresentada no conjunto solução. Caso seja maior que zero, existem duas raízes reais distintas que devem ser apresentadas no conjunto solução, em ordem crescente. Apresente também o valor de Δ . Todos os valores são decimais e devem ser apresentados usando duas casas de precisão. Lembrando que, para $ax^2 + bx + c = 0$, tem-se:

$$\bullet \quad x = \frac{-b \pm \sqrt{\Delta}}{2a}$$

$$\bullet \quad \Delta = b^2 - 4ac$$

Arquivo com a solução: Exercicio6\$1.java

Entrada

```
a: 1
b: 5
c: 4
```

Saída

```
Delta: 9.00
S = {-4.00, -1.00}
```

Entrada

```
a: 1
b: 4
c: 4
```

Saída

```
Delta: 0.00
S = {-2.00}
```

Entrada

a: 2
b: 2
c: 1

Saída

Delta: -4.00
S = {}

Entrada

a: 0
b: 3
c: -2

Saída

Nao existe equacao do segundo grau!

Exercício 6.2:

Escreva um programa que peça para o usuário fornecer dois números decimais. Um desses números é a base, enquanto o outro é o expoente. Seu programa deve calcular a base elevada ao expoente e exibir o valor obtido. Exiba o resultado usando duas casas decimais de precisão.

Arquivo com a solução: Exercicio6\$2.java

Entrada

Base: 2
Expoente: 10

Saída

2.00 ^ 10.00 = 1024.00

Exercício 6.3:

Escreva um programa que peça para o usuário fornecer um número decimal. O programa deve calcular e exibir o maior e o menor inteiro mais próximo ao valor fornecido. Exiba os resultados usando duas casas decimais de precisão.

Arquivo com a solução: Exercicio6\$3.java

Entrada

Numero: 3.5

Saída

Maior inteiro mais proximo: 4.00
Menor inteiro mais proximo: 3.00

Entrada

Numero: -3.5

Saída

Maior inteiro mais proximo: -3.00
Menor inteiro mais proximo: -4.00

Exercício 6.4:

Escreva um programa que peça para o usuário fornecer um número decimal. O programa deve calcular e exibir o valor absoluto (módulo) do valor fornecido. Exiba o resultado usando duas casas decimais de precisão.

Arquivo com a solução: Exercicio6\$4.java

Entrada

Numero: 9.5

Saída

Valor absoluto: 9.50

Entrada

Numero: -9.5

Saída

Valor absoluto: 9.50

Exercício 6.5:

Escreva um programa que peça para o usuário fornecer um número decimal. Caso o número seja positivo, o programa deve calcular e exibir sua raiz quadrada, caso contrário, deve calcular e exibir o quadrado do número. Exiba o resultado usando duas casas decimais de precisão.

Arquivo com a solução: Exercicio6\$5.java

Entrada

Numero: 9

Saída

Raiz quadrada de 9.00: 3.00

Entrada

Numero: -5

Saída

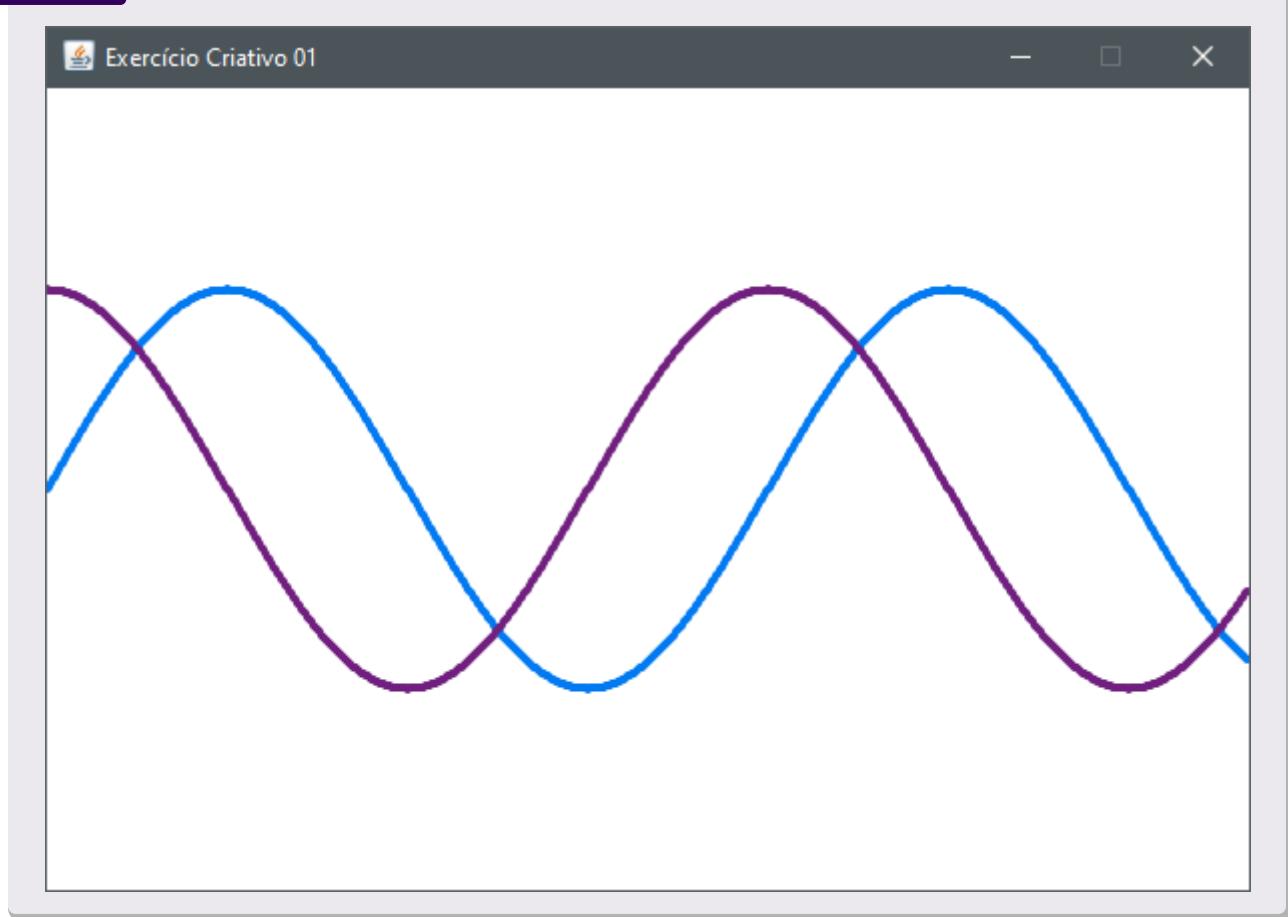
Quadrado de -5.00: 25.00

6.3 Exercícios Criativos

Exercício Criativo 6.1:

Escreva um programa que desenhe a função seno e a função cosseno. Você está livre para usar as cores e as dimensões que quiser. No exemplo, a função seno está representada em azul e a função cosseno em roxo.

Saída:

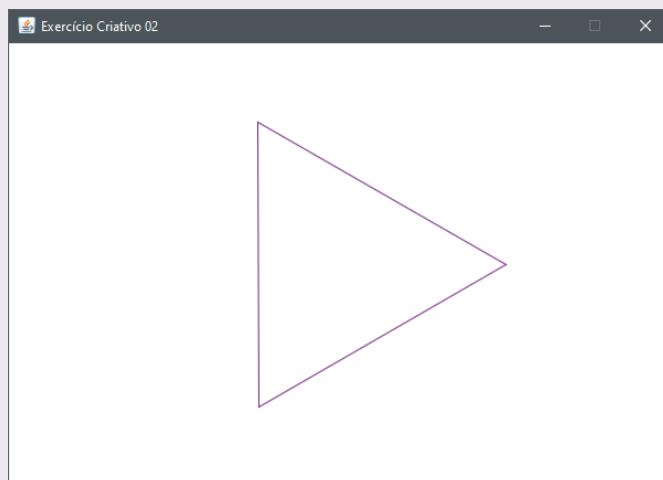


Exercício Criativo 6.2:

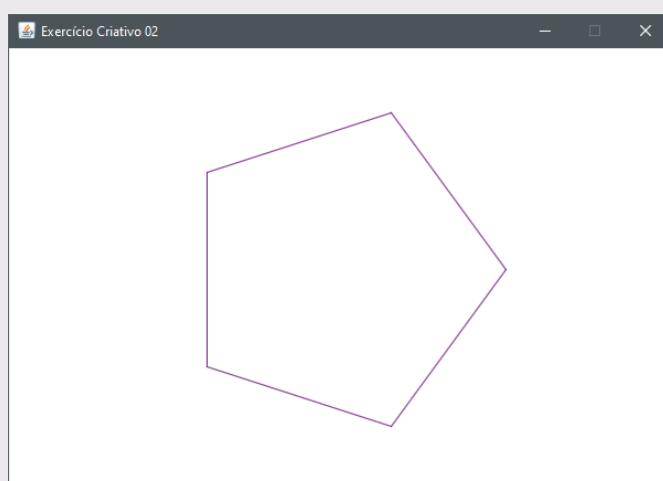
Escreva um programa que dada a quantidade de lados de um polígono regular (maior que 3) e o valor do raio de uma circunferência, desenhe os contornos desse polígono considerando que ele está inscrito nessa circunferência. Você está livre para usar as cores e as dimensões que quiser. Dica: as funções seno e cosseno são um possível caminho para o cálculo dos vértices.

Entrada

```
lados: 3  
raio: 150
```

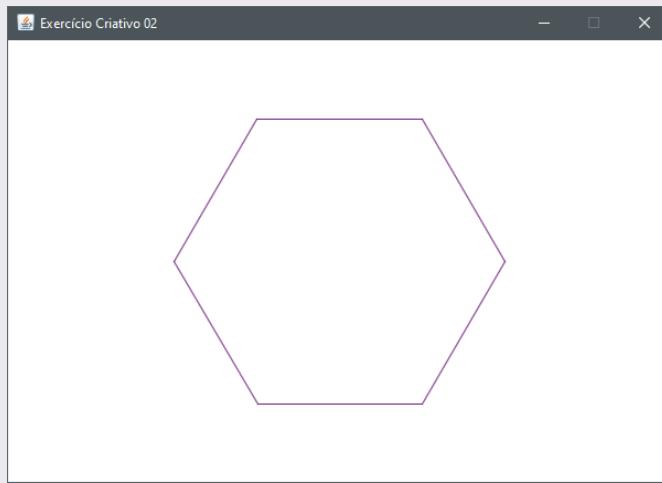
Saída:**Entrada**

```
lados: 5  
raio: 150
```

Saída:

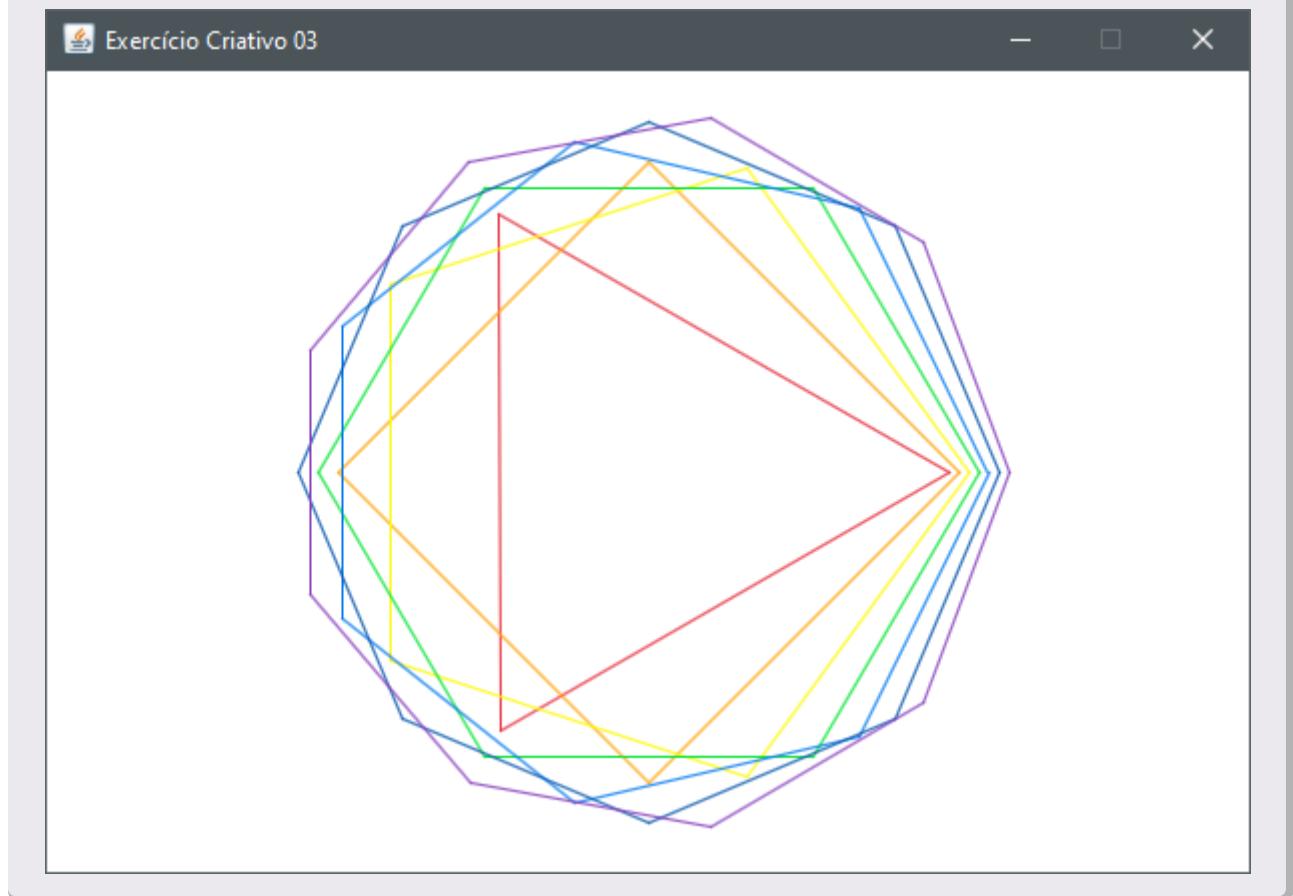
Entrada

lados: 6
raio: 150

Saída:

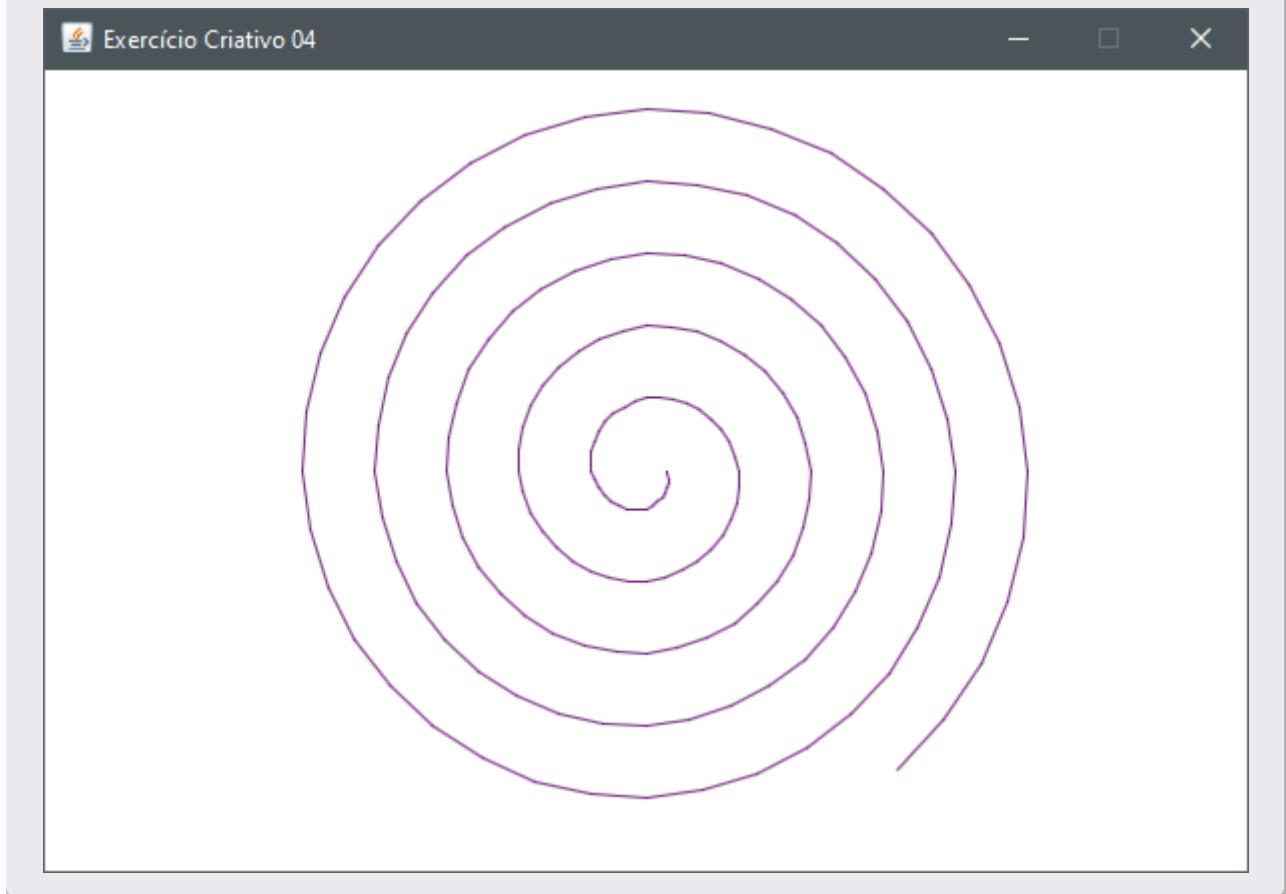
Exercício Criativo 6.3:

Utilizando a sua solução do exercício anterior, desenhe uma série de polígonos regulares concêntricos, de quantidade de lados igual a 3, 4, 5, 6, 7, 8 e 9, cada um de uma cor. No exemplo o polígono mais interno tem raio igual a 150 e cada um dos próximos polígonos possuem 5 pixels a mais de raio em comparação ao anterior. Você está livre para usar as cores e as dimensões que quiser.

Saída:

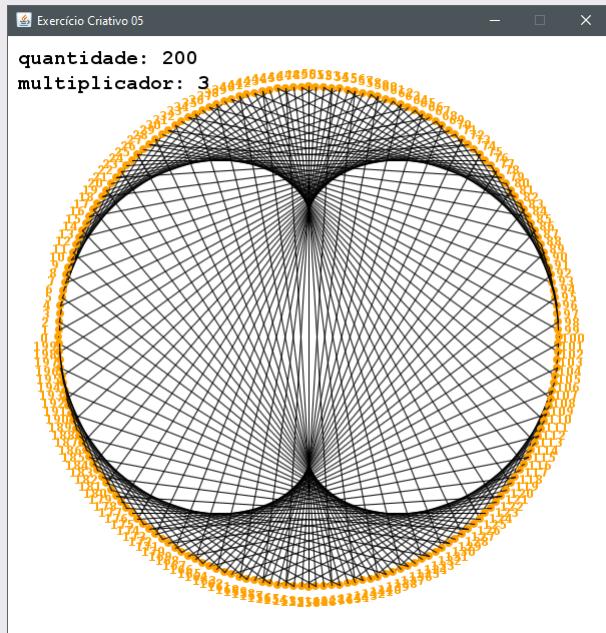
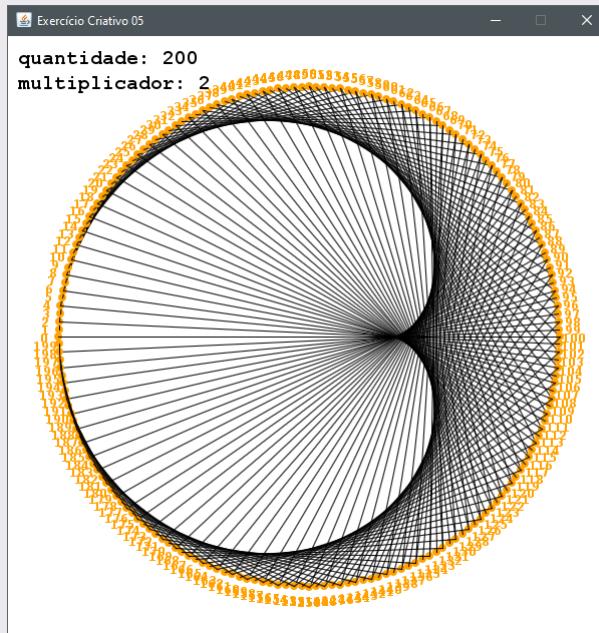
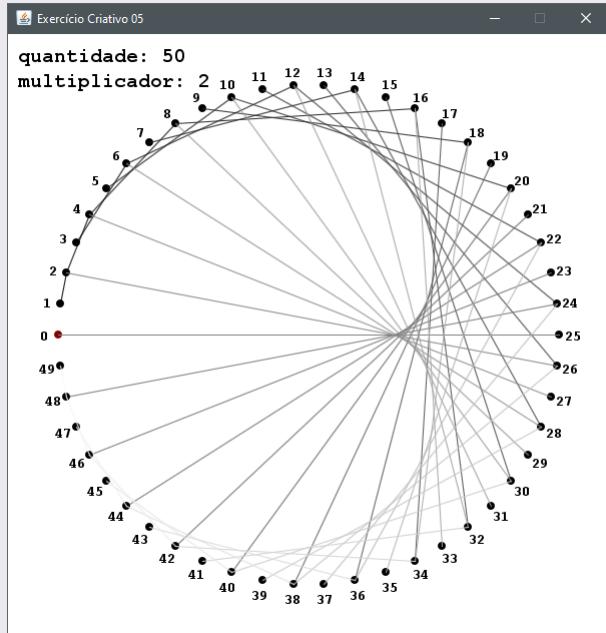
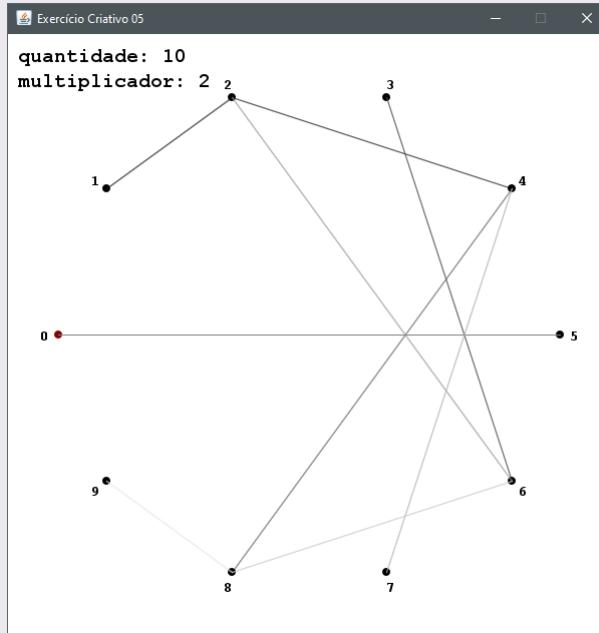
Exercício Criativo 6.4:

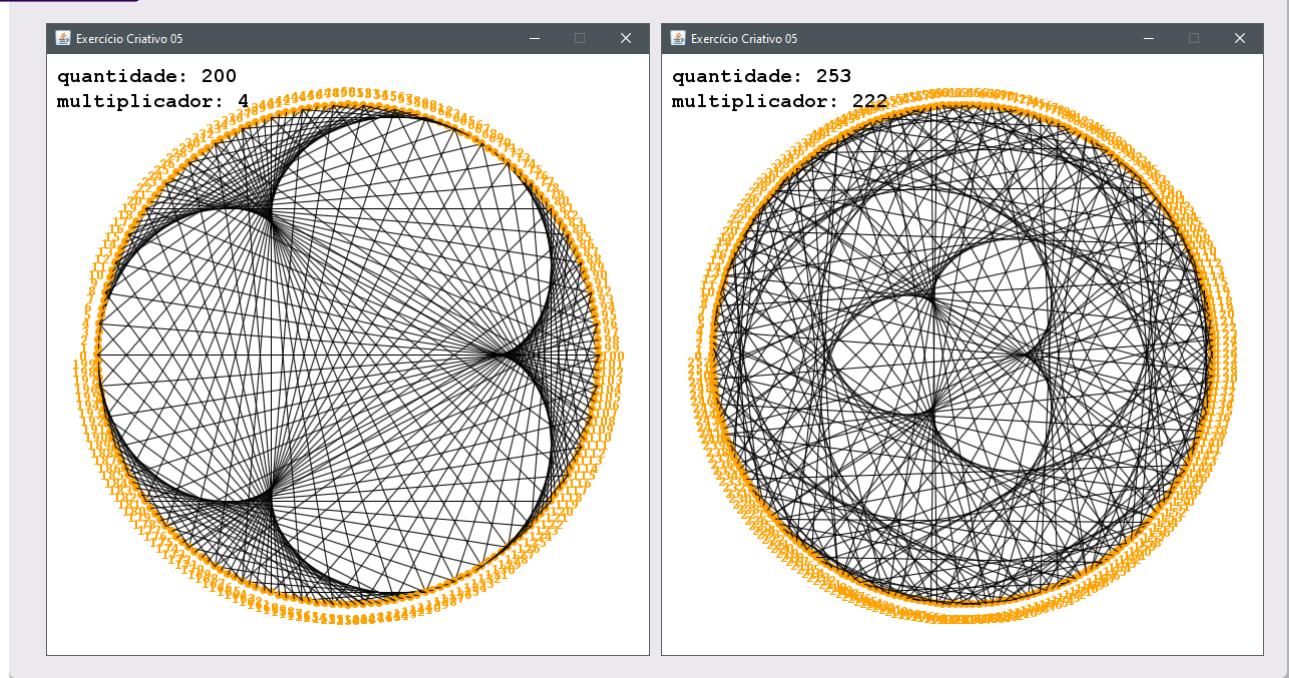
Escreva um programa que desenhe uma espiral. Você está livre para usar as cores e as dimensões que quiser.

Saída:

Exercício Criativo 6.5:

Escreva um programa que desenhe um cardioide. Veja as imagens abaixo e assista ao vídeo <<https://www.youtube.com/watch?v=qhbuKbxJsk8>> para entender qual a ideia geral para a geração de tais desenhos. Esse vai ser bastante divertido!

Saídas:

Saídas:

Que tal um projeto agora? Vamos lá!

6.4 Projetos

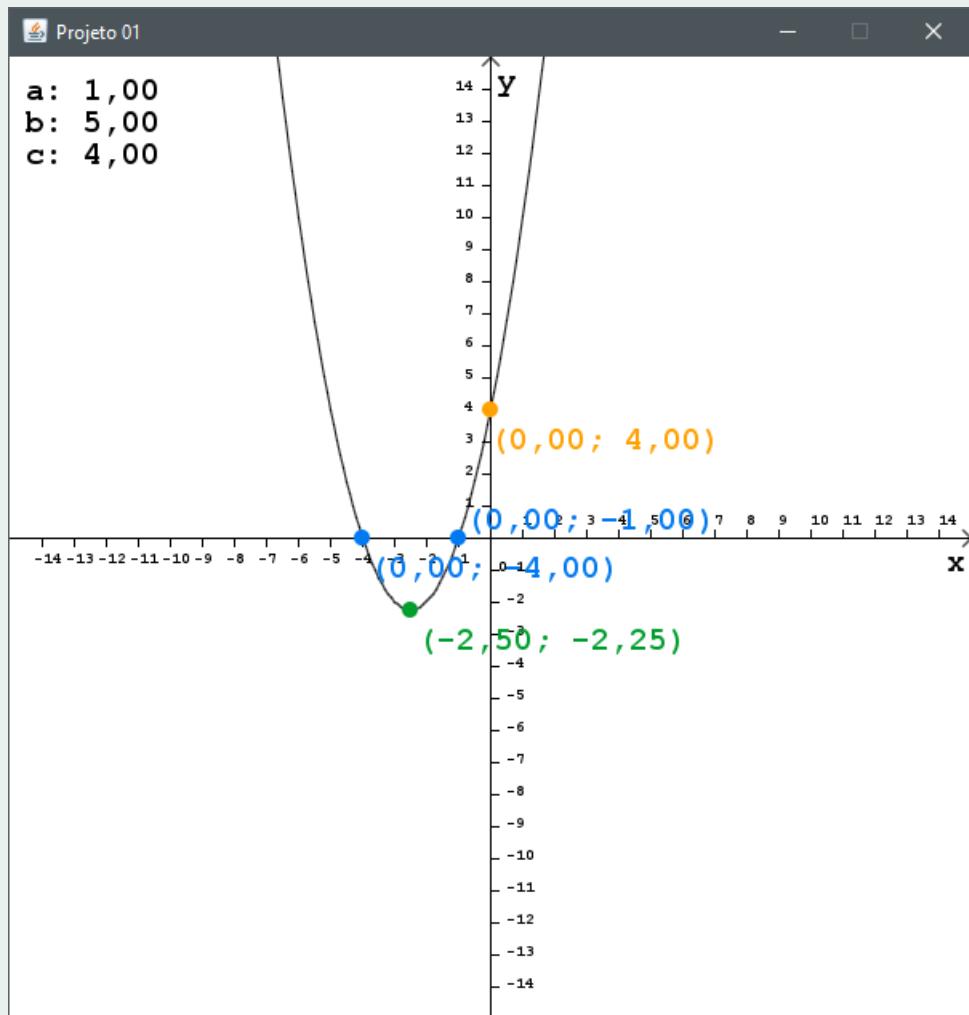
Projeto 6.1:

Desenvolva um programa que o usuário deve fornecer os coeficientes de uma função do segundo grau e, a partir desses dados, deve ser plotado (desenhado) o gráfico da função! A entrada dos dados deve ocorrer via terminal, como você está fazendo até o momento. Além do desenho da função, seu programa deve indicar os zeros da mesma, ou seja, os pontos em que ela cruza o eixo x , o ponto em que ela cruza o eixo y e qual seu ponto mínimo ou máximo. Veja alguns exemplos abaixo.

Entrada

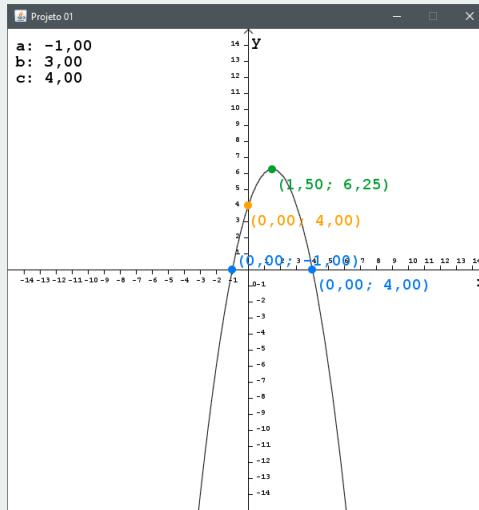
```
a: 1  
b: 5  
c: 4
```

Saída:

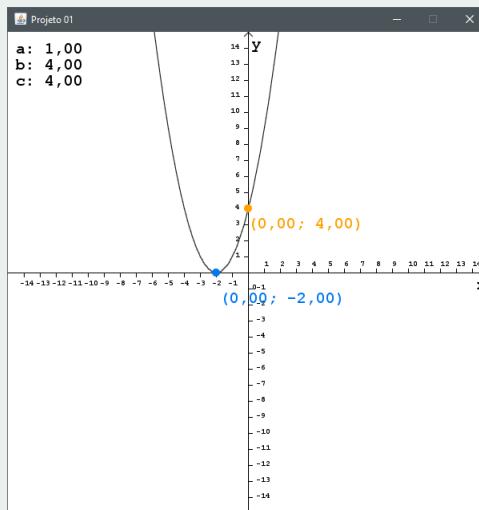


Entrada

a: -1
 b: 3
 c: 4

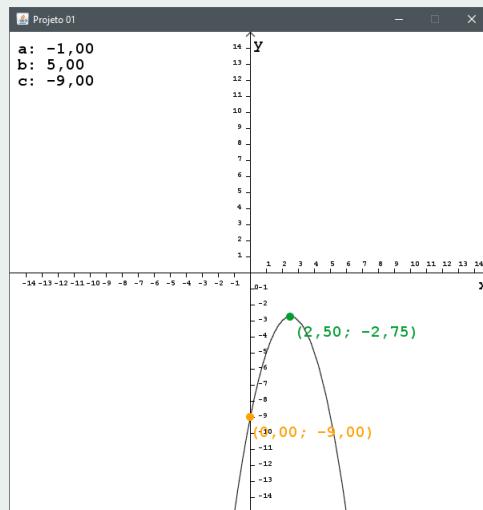
Saída:**Entrada**

a: 1
 b: 4
 c: 4

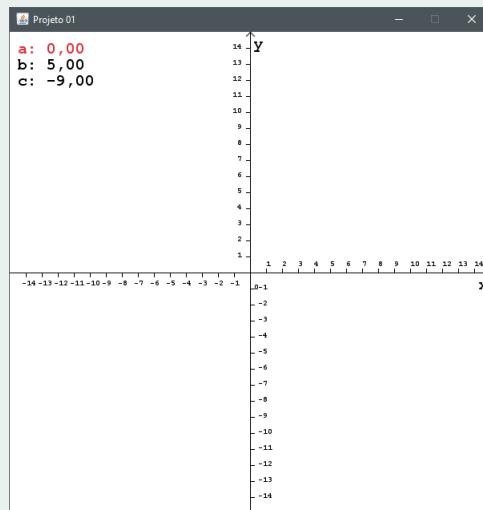
Saída:

Entrada

a: -1
b: 5
c: -9

Saída:**Entrada**

a: 0
b: 5
c: -9

Saída:

MÉTODOS ESTÁTICOS

“Form ever follows function”.

Louis Henri Sullivan



S funções são a unidade de programação básica das linguagens de programação estruturadas como a linguagem C. Na linguagem Java a construção análoga às funções são os métodos, entretanto, existem basicamente dois grandes “tipos de métodos”, ou seja, os métodos de classe e os métodos de instância. Focaremos nesse Capítulo nos métodos de classe, isto é, métodos que dependem de uma classe, mas não dos objetos criados usando-a. Esse tipo de método é o que mais se aproxima das funções da linguagem C. Os métodos de instância e mais detalhes sobre membros de classe e de instância serão tratados posteriormente.

7.1 Exemplos em Linguagem Java

Exemplos de implementação de métodos estáticos

```
1  /*
2   * Arquivo: capitulo07/MetodosEstaticos.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6  public class MetodosEstaticos {
7
8      /*
9       * Métodos Estáticos:
10      *
11      * Os métodos devem ser nomeados, preferencialmente, usando o padrão
12      * camel case com a primeira letra em minúscula.
13      *
14      * Na linguagem Java pode haver mais de um método com o mesmo nome
15      * em um mesmo escopo, mas para que isso seja válido, a assinatura o
16      * método (sua lista de parâmetros) deve ser diferente um do outro.
17      *
18      * Um parâmetro de um método é parte do método e descreve um tipo de
19      * dado que será recebido.
20      *
21      * Um argumento é o valor em si, passado através de um parâmetro, para o
22      * método utilizar. Em Java as passagens de parâmetro são sempre por
23      * valor.
24      */
25
26      /*
27       * Método estático público main
28       * - possui como parâmetro um array de String => ( String[] args );
29       * - não retorna nada => void antes do nome do método.
30       */
31  public static void main( String[] args ) {
32
33      int n1 = 3;
34      int n2 = 4;
35      int[] a = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
36      int resultado;
37
38      System.out.printf( "%d + %d = %d", n1, n2, adicao( n1, n2 ) );
39      pularLinha();
40
41      resultado = subtracao( n1, n2 );
42      System.out.printf( "%d - %d = %d", n1, n2, resultado );
43      pularLinha();
44
45      System.out.print( "zero a dez: " );
```

```
46     imprimirNumeros();
47
48     imprimeTabuada( 5 );
49
50     System.out.println( "Dados do array (fora do metodo):" );
51     for ( int i = 0; i < 10; i++ ) {
52         System.out.printf( "a[%d] = %d\n", i, a[i] );
53     }
54     processarArray( a, 10 );
55     System.out.println( "Dados do array (apos execucao do metodo):" );
56     for ( int i = 0; i < 10; i++ ) {
57         System.out.printf( "a[%d] = %d\n", i, a[i] );
58     }
59
60     System.out.printf( "Soma dos valores 1, 2, 3 e 4 = %d\n",
61                        somarNumeros( 1, 2, 3, 4 ) );
62
63     System.out.printf( "5! = %d\n", fatorial( 5 ) );
64
65 }
66
67 /*
68 * Método estático público adicao:
69 * - possui dois parâmetros inteiros => ( int, int );
70 * - retorna um inteiro => int antes do nome do método;
71 * - obs: todos nossos métodos por enquanto serão públicos
72 *       e estáticos.
73 */
74 public static int adicao( int n1, int n2 ) {
75     return n1 + n2;
76 }
77
78 /*
79 * Método estático público subtracao:
80 * - possui dois parâmetros inteiros => ( int n1, int n2 );
81 * - retorna um inteiro => int antes do nome do método.
82 */
83 public static int subtracao( int n1, int n2 ) {
84
85     /* variável resultado é interna ao método!
86      * ela tem escopo local ao método.
87      */
88     int resultado = n1 - n2;
89
90     return resultado;
91 }
92
93 /*
94 */
```

```
95     * Método estático público pularLinha
96     * - não possui parâmetros => ();
97     * - não retorna nada => void antes do nome do método;
98     * - obs: construções análogas aos métodos estáticos que não
99     *         retornam valores são chamadas também de procedimentos.
100    */
101   public static void pularLinha() {
102     System.out.println();
103   }
104
105  /*
106   * Método estático público imprimirNumeros
107   * - não possui parâmetros => ();
108   * - não retorna nada => void antes do nome do método.
109   */
110  public static void imprimirNumeros() {
111
112    for ( int i = 0; i <= 10; i++ ) {
113      System.out.print( i + " " );
114    }
115
116    pularLinha();
117  }
118
119  /*
120   * Método estático público processarArray
121   * - possui dois parâmetros, um array de inteiros e um
122   *     inteiro => ( int[] a, int n );
123   * - não retorna nada => void antes do nome do método;
124   * - obs: parâmetros que são arrays tem um comportamento "especial".
125   *         Iremos aprender os detalhes disso posteriormente!
126   *         Por enquanto, entenda que um array passado como
127   *         parâmetro poderá ser modificado dentro do método.
128   */
129
130  public static void processarArray( int[] a, int n ) {
131
132    System.out.println( "Dados do array (dentro do metodo):" );
133    for ( int i = 0; i < n; i++ ) {
134      System.out.printf( "a[%d] = %d\n", i, a[i] );
135    }
136
137    System.out.println(
138      "Modificando os dados do array (dentro do metodo)...");
139    for ( int i = 0; i < n; i++ ) {
140      a[i] += 2;
141    }
142
143    System.out.println(
```

```
144         "Dados do array apos modificacao (dentro do metodo):" );
145     for ( int i = 0; i < n; i++ ) {
146         System.out.printf( "a[%d] = %d\n", i, a[i] );
147     }
148 }
149 }
150 /*
151 * Método estatico público imprimeTabuada
152 * - possui um parâmetro inteiro => ( int n );
153 * - não retorna nada => void antes do nome do método.
154 */
155 public static void imprimeTabuada( int n ) {
156
157     for ( int i = 0; i <= 10; i++ ) {
158         System.out.printf( "%d x %d = %d\n", n, i, n*i );
159     }
160 }
161 }
162 /*
163 * Método estatico público somarNumeros
164 * - possui um parâmetro de comprimento
165 * variável (vararg) => ( int... numeros );
166 * - retorna a soma dos valores passados
167 *
168 * Os parâmetros de comprimento variável são
169 * interpretados internamente como se fossem
170 * arrays. Caso nenhum valor seja passado,
171 * existirá o array, entretanto com comprimento
172 * igual a zero. Outro detalhe é que esse tipo
173 * de parâmetro deve ser fornecido, obrigatoriamente,
174 * no final da lista de parâmetros e cada método
175 * só pode ter um parâmetro desse tipo.
176 */
177 public static int somarNumeros( int... numeros ) {
178
179     int soma = 0;
180
181     for ( int i = 0; i < numeros.length; i++ ) {
182         soma += numeros[i];
183     }
184
185     return soma;
186 }
187 }
188 /*
189 * Método estatico público fatorial
190 */
191 /*
192 * Método estatico público fatorial
193 */
```

```
193     * - possui um parâmetro inteiro => ( int n );
194     * - retorna um valor inteiro.
195     */
196 public static int factorial( int n ) {
197
198     /*
199     * A partir da versão 10 da linguagem Java foi adicionado um
200     * nome de tipo reservado chamado "var", que instrui o compilador
201     * a executar a inferência de tipo de variáveis locais, ou seja,
202     * variáveis que são locais à qualquer tipo de método podem ter
203     * seus tipos inferidos pelo compilador a partir do tipo do valor
204     * usado para inicializá-las.
205     *
206     * É obrigatório que variáveis são declaradas usando var sejam
207     * inicializadas e o valor de inicialização não pode ser null.
208     */
209     var fat = 1;
210
211     for ( int i = 2; i <= n; i++ ) {
212         fat *= i;
213     }
214
215     return fat;
216
217 }
218
219 }
```

⚠ | Atenção!

A passagem dos argumentos para os métodos em Java **sempre** é feita por valor, ou seja, **sempre** são copiados os valores das variáveis que forem usadas na chamada do método para os seus respectivos parâmetros. Infelizmente esse assunto gera muita confusão e as vezes é mal ensinado, onde professores dizem que existe passagem por valor e por referência. Isso está errado! Em Java, assim como em C, **a passagem é sempre por valor**. Mesmo que em Java todo e qualquer valor que não represente um valor de um tipo primitivo seja chamado de referência, a passagem de parâmetros ainda assim é por valor. Em C++ existe um tipo diferente de variável que é chamada de referência e, nesse caso, existe a passagem por referência, mas isso foge do escopo desse livro. No exemplo apresentado abaixo é mostrado o uso de um método e na Figura 7.1 pode-se ver uma representação de como isso se dá na memória principal.

Usando um método

```

1 public static void main( String[] args ) {
2     int a = 1;
3     int b = 2;
4     System.out.printf( "%d + %d = %d", a, b, somar(a, b) );
5 }
6
7 public static int somar( int n1, int n2 ) {
8     return n1 + n2;
9 }
```

Memória Principal

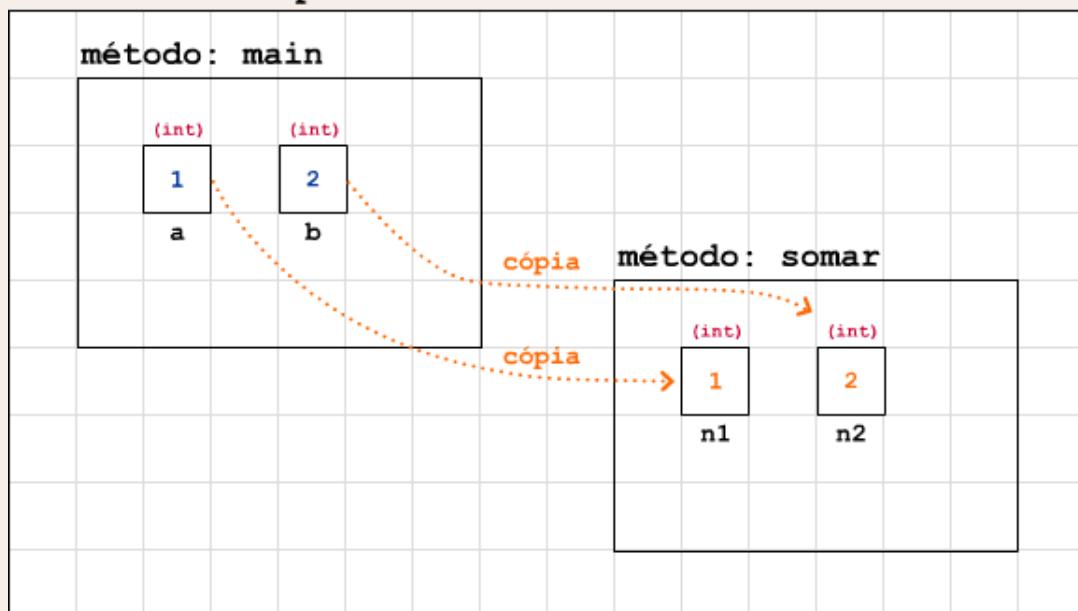


Figura 7.1: Invocação de métodos e passagem por valor

7.2 Exercícios

Exercício 7.1:

Escreva um programa que leia 5 valores inteiros e imprima para cada um o seu valor absoluto. Para obter o valor absoluto do número utilize o método estático público “absoluto”, especificado abaixo:

- **Nome:** `absoluto`
- **Descrição:** Calcula o valor absoluto, ou módulo, do número fornecido.
- **Entrada/Parâmetro(s):** `int n`
- **Saída/Retorno:** O valor absoluto de n (`int`).

Arquivo com a solução: `Exercicio7$1.java`

Entrada

```
n0: 5  
n1: 6  
n2: -7  
n3: 8  
n4: 9
```

Saída

```
absoluto(5) = 5  
absoluto(6) = 6  
absoluto(-7) = 7  
absoluto(8) = 8  
absoluto(9) = 9
```

Exercício 7.2:

Escreva um programa que leia o valor do raio de um círculo. O programa deve calcular e imprimir a área e o perímetro do círculo representado por esse raio. Para obter o valor da área do círculo o programa deverá chamar o método estático público “areaCirculo” e para obter o valor do seu perímetro o programa deverá invocar o método estático público “circunferenciaCirculo”. Para o valor de π , utilize `Math.PI`, assim como já apresentado no Capítulo sobre os métodos matemáticos contidos na classe `Math`.

- **Nome:** `areaCirculo`
- **Descrição:** Calcula a área do círculo representado pelo raio fornecido.
- **Entrada/Parâmetro(s):** `double raio`
- **Saída/Retorno:** A área do círculo (`double`).

- **Nome:** `circunferenciaCirculo`
- **Descrição:** Calcula a circunferência do círculo representado pelo raio fornecido.
- **Entrada/Parâmetro(s):** `double raio`
- **Saída/Retorno:** A circunferência do círculo (`double`).

Arquivo com a solução: `Exercicio7$2.java`

Entrada

```
Raio: 5
```

Saída

```
Area = 78.54  
Circunferencia = 31.42
```

Exercício 7.3:

Escreva um programa que leia 5 pares de valores decimais. Todos os valores lidos devem ser positivos. Caso um valor menor ou igual a zero for fornecido, esse valor deve ser lido novamente. Para cada par lido deve ser impresso o valor do maior elemento do par ou a frase “Eles sao iguais” se os valores do par forem iguais. Para obter o maior elemento do par utilize o método estático público “maiorNúmero”.

- **Nome:** maiorNúmero
- **Descrição:** Calcula o maior valor entre os dois valores.
- **Entrada/Parâmetro(s):** double n1, double n2
- **Saída/Retorno:** Retorna o maior valor os dois fornecidos ou –1 caso sejam iguais (double).
- **Observação:** Considere que os valores de entrada serão sempre positivos.

Arquivo com a solução: Exercicio7\$3.java

Entrada

```
n1[0]: 2
n2[0]: 3
n1[1]: 4
n2[1]: 6
n1[2]: 5
n2[2]: 5
n1[3]: -6
Entre com um valor positivo!
n1[3]: 4
n2[3]: -7
Entre com um valor positivo!
n2[3]: -8
Entre com um valor positivo!
n2[3]: 3
n1[4]: 4
n2[4]: 2
```

Saída

```
2.00, 3.00: O maior valor e 3.00
4.00, 6.00: O maior valor e 6.00
5.00, 5.00: Eles sao iguais
4.00, 3.00: O maior valor e 4.00
4.00, 2.00: O maior valor e 4.00
```

Exercício 7.4:

Escreva um programa que leia 5 números inteiros positivos, ou seja, maiores que zero. Caso o valor lido não seja positivo, o programa deve solicitar a entrada do valor novamente. Para cada valor lido escrever o somatório dos inteiros de 1 ao número informado. O resultado do cálculo desse somatório deve ser obtido através do método estático público “somatorio”.

- **Nome:** `somatorio`
- **Descrição:** Calcula o somatório dos inteiros de 1 ao número fornecido como parâmetro.
- **Entrada/Parâmetro(s):** `int n`
- **Saída/Retorno:** O valor do somatório (`int`).

Arquivo com a solução: `Exercicio7$4.java`

Entrada

```
n[0]: 5
n[1]: 4
n[2]: 9
n[3]: -7
Entre com um valor positivo: 8
n[4]: -8
Entre com um valor positivo: -9
Entre com um valor positivo: -4
Entre com um valor positivo: 3
```

Saída

```
Somatorio de 1 a 5: 15
Somatorio de 1 a 4: 10
Somatorio de 1 a 9: 45
Somatorio de 1 a 8: 36
Somatorio de 1 a 3: 6
```

Exercício 7.5:

Escreva um programa que leia dois números 5 vezes. O programa deve verificar se o primeiro número fornecido é par e se o primeiro número é divisível pelo segundo, ou seja, se o resto da divisão do primeiro pelo segundo é zero. Para fazer tais verificações, utilize os métodos estáticos públicos “ehPar” e “ehDivisivel”.

- **Nome:** ehPar
- **Descrição:** Verifica se o número fornecido é ou não par.
- **Entrada/Parâmetro(s):** int n
- **Saída/Retorno:** true caso o número seja par ou false caso contrário (boolean).

- **Nome:** ehDivisivel
- **Descrição:** Verifica se um número é divisível por outro.
- **Entrada/Parâmetro(s):** int dividendo, int divisor
- **Saída/Retorno:** true caso o dividendo seja divisível pelo divisor ou false caso contrário (boolean).

Arquivo com a solução: Exercicio7\$5.java

Entrada

```
n1[0]: 8
n2[0]: 4
n1[1]: 7
n2[1]: 3
n1[2]: 21
n2[2]: 7
n1[3]: 9
n2[3]: 5
n1[4]: 10
n2[4]: 5
```

Saída

```
8 eh par e 8 eh divisivel por 4
7 eh impar e 7 nao eh divisivel por 3
21 eh impar e 21 eh divisivel por 7
9 eh impar e 9 nao eh divisivel por 5
10 eh par e 10 eh divisivel por 5
```

Exercício 7.6:

Escreva um programa que leia 5 números inteiros positivos, ou seja, maiores que zero. Caso o valor lido não seja positivo, o programa deve solicitar a entrada do valor novamente. Para cada número informado escrever a soma de seus divisores (exceto ele mesmo). Utilize o método estático público “somaDivisores” para obter a soma.

- **Nome:** `somaDivisores`
- **Descrição:** Calcula a soma dos divisores do número informado, exceto ele mesmo.
- **Exemplo:** Para o valor 8, tem-se que $1 + 2 + 4 = 7$
- **Entrada/Parâmetro(s):** `int n`
- **Saída/Retorno:** A soma dos divisores do número fornecido `(int)`

Arquivo com a solução: `Exercicio7$6.java`

Entrada

```
n[0]: 8
n[1]: 10
n[2]: 5
n[3]: -8
Entre com um valor positivo: 9
n[4]: -7
Entre com um valor positivo: -8
Entre com um valor positivo: -7
Entre com um valor positivo: 50
```

Saída

```
Soma dos divisores de 8: 7
Soma dos divisores de 10: 8
Soma dos divisores de 5: 1
Soma dos divisores de 9: 4
Soma dos divisores de 50: 43
```

Exercício 7.7:

Escreva um programa que imprima na tela os números primos existentes entre 1, inclusive, e 20, inclusive. Para verificar se um número é primo utilize o método estático público “ehPrimo”.

- **Nome:** ehPrimo
- **Descrição:** Verifica se um número é ou não primo.
- **Entrada/Parâmetro(s):** int n
- **Saída/Retorno:** true caso o número seja primo ou false caso contrário (boolean).

Arquivo com a solução: Exercicio7\$7.java

Saída

```
1: nao eh primo
2: eh primo
3: eh primo
4: nao eh primo
5: eh primo
6: nao eh primo
7: eh primo
8: nao eh primo
9: nao eh primo
10: nao eh primo
11: eh primo
12: nao eh primo
13: eh primo
14: nao eh primo
15: nao eh primo
16: nao eh primo
17: eh primo
18: nao eh primo
19: eh primo
20: nao eh primo
```

Exercício 7.8:

Escreva um programa que leia 5 pares de inteiros positivos, ou seja, maiores que zero. Caso o valor lido não seja positivo, o programa deve solicitar a entrada do valor novamente. Imprima se os elementos de cada par são números amigos ou não. Dois números “a” e “b” são amigos se a soma dos divisores de “a” excluindo “a” é igual a “b” e a soma dos divisores de “b” excluindo “b” é igual a “a”. Para verificar se dois números são amigos utilize o método estático público “saoAmigos”.

- **Nome:** `saoAmigos`
- **Descrição:** Verifica se dois números são amigos.
- **Observação:** Utilize o método estático público “somaDividores” do exercício anterior.
- **Exemplo:** 220 e 284 são amigos, pois:
 - **220:** $1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284$
 - **284:** $1 + 2 + 4 + 71 + 142 = 220$
- **Entrada/Parâmetro(s):** `int n1, int n2`
- **Saída/Retorno:** `true` caso os números sejam amigos ou `false` caso contrário (`boolean`).

Arquivo com a solução: Exercicio7\$8.java

Entrada

```
n1[0]: 220
n2[0]: 284
n1[1]: 128
n2[1]: 752
n1[2]: 789
n2[2]: 568
n1[3]: 1184
n2[3]: 1210
n1[4]: 874
n2[4]: 138
```

Saída

```
220 e 284 sao amigos
128 e 752 nao sao amigos
789 e 568 nao sao amigos
1184 e 1210 sao amigos
874 e 138 nao sao amigos
```

Exercício 7.9:

Escreva um programa que leia as medidas dos lados de 5 triângulos. Para cada triângulo imprimir a sua classificação (Não é triângulo, Triângulo Equilátero, Isósceles ou Escaleno). O programa deve aceitar apenas valores positivos para as medidas dos lados, ou seja, valores maiores que zero. Caso o valor lido não seja positivo, o programa deve solicitar a entrada do valor novamente. Para verificar se as medidas formam um triângulo chamar o método estático público “ehTriangulo”. Para obter o código da classificação utilize o método estático público “tipoTriangulo”.

- **Nome:** `ehTriangulo`
- **Descrição:** Verifica se as 3 medidas informadas permitem formar um triângulo. Essa condição de existência já foi apresentada em um Capítulo anterior.
- **Entrada/Parâmetro(s):** `int ladoA, int ladoB, int ladoC`
- **Saída/Retorno:** `true` caso os valores representam um triângulo ou `false` caso contrário (`boolean`).

- **Nome:** `tipoTriangulo`
- **Descrição:** A partir das medidas dos lados de um triângulo, verifica o tipo do triângulo.
- **Entrada/Parâmetro(s):** `int ladoA, int ladoB, int ladoC`
- **Saída/Retorno:** Um inteiro (`int`), sendo que:
 - **0:** se não formam um triângulo;
 - **1:** se for um triângulo equilátero;
 - **2:** se for um triângulo isósceles;
 - **3:** se for um triângulo escaleno.

Arquivo com a solução: Exercicio7\$9.java

Entrada

```

ladoA[0] : 2
ladoB[0] : 2
ladoC[0] : 2
ladoA[1] : 2
ladoB[1] : 3
ladoC[1] : -10
Entre com um valor positivo: -5
Entre com um valor positivo: 5
ladoA[2] : 3
ladoB[2] : 4
ladoC[2] : 5
ladoA[3] : 7
ladoB[3] : 7
ladoC[3] : -15
Entre com um valor positivo: -19
Entre com um valor positivo: 8
ladoA[4] : 1
ladoB[4] : 10
ladoC[4] : 20

```

Saída

Valores 2, 2 e 2: triangulo equilatero
Valores 2, 3 e 5: nao formam um triangulo
Valores 3, 4 e 5: triangulo escaleno
Valores 7, 7 e 8: triangulo isosceles
Valores 1, 10 e 20: nao formam um triangulo

Exercício 7.10:

Escreva um programa que leia um valor inteiro de 1 a 9999 e imprima o seu dígito verificador. Para obter o valor do dígito verificador utilize o método estático público “calculaDigito”. Caso seja fornecido um número fora da faixa estabelecida, o programa deve ser encerrado sem apresentar nenhuma mensagem.

- **Nome:** `calculaDigito`
- **Descrição:** Calcula o dígito verificador de um número. Para evitar erros de digitação em números de grande importância, como código de uma conta bancária, geralmente se adiciona ao número um dígito verificador. Por exemplo, o número 1841 é utilizado normalmente como 18414, onde o 4 é o dígito verificador. Ele é calculado da seguinte forma:
 - **a)** Cada algarismo do número é multiplicado por um peso começando em 2, da direita para a esquerda. Para cada algarismo o peso é acrescido de 1. Soma-se então os produtos obtidos. Exemplo: $1 * 5 + 8 * 4 + 4 * 3 + 1 * 2 = 51$
 - **b)** Calcula-se o resto da divisão desta soma por 11: $51 \% 11 = 7$
 - **c)** Subtrai-se de 11 o resto obtido: $11 - 7 = 4$
 - **d)** Se o valor obtido for 10 ou 11, o dígito verificador será o 0, nos outros casos, o dígito verificador é o próprio valor encontrado.
- **Entrada/Parâmetro(s):** `int n`
- **Saída/Retorno:** O dígito verificador do número `(int)`.

Arquivo com a solução: `Exercicio7$10.java`

Entrada

Numero: 1841

Saída

Dígito verificador de 1841: 4

Entrada

Numero: 6857

Saída

Dígito verificador de 6857: 8

Entrada

Numero: 751

Saída

Dígito verificador de 751: 0

Exercício 7.11:

Escreva um programa que leia um valor inteiro de 10 a 99999, onde o último algarismo representa o seu dígito verificador. Imprima uma mensagem indicando se ele foi digitado corretamente ou não. Utilize o método estático público “numeroCorreto” para verificar se o número está correto. Caso seja fornecido um número fora da faixa estabelecida, o programa deve ser encerrado sem apresentar nenhuma mensagem.

- **Nome:** `numeroCorreto`
- **Descrição:** Verifica se um número, em conjunto com seu dígito, está correto.
- **Entrada/Parâmetro(s):** `int n`
- **Saída/Retorno:** `true` se o número está correto ou `false` caso contrário (`boolean`).
- **Observação:** Use os métodos estáticos públicos abaixo: “obtemNúmero”, “obtemDigito” e “calculaDigito”.

- **Nome:** `obtemDigito`
- **Descrição:** Separa o dígito verificador (a unidade) do número.
- **Entrada/Parâmetro(s):** `int n`
- **Saída/Retorno:** O último algarismo do número (`int`).
- **Exemplo:** Para o valor 1823 o dígito é 3.

- **Nome:** `obtemNúmero`
- **Descrição:** Separa o número do dígito verificador.
- **Entrada/Parâmetro(s):** `int n`
- **Saída/Retorno:** O número sem o valor da unidade (`int`).
- **Exemplo:** Para o valor 1823 o número é 182.

Arquivo com a solução: Exercicio7\$11.java

Entrada

```
Numero: 18414
```

Saída

```
Numero completo: 18414
Numero: 1841
Digito: 4
Digito calculado: 4
O numero fornecido esta correto!
```

Entrada

Numero: 68577

Saída

Numero completo: 68577

Numero: 6857

Dígito: 7

Dígito calculado: 8

O número fornecido está incorreto!

Entrada

Numero: 7510

Saída

Numero completo: 7510

Numero: 751

Dígito: 0

Dígito calculado: 0

O número fornecido está correto!

Exercício 7.12:

Escreva um programa que leia 3 duplas de valores inteiros. Exibir cada dupla em ordem crescente. A ordem deve ser impressa através da chamada do método estático público “classificaDupla” especificado abaixo:

- **Nome:** `classificaDupla`
- **Descrição:** Imprime em ordem crescente dois valores inteiros.
- **Entrada/Parâmetro(s):** `int n1, int n2`
- **Saída/Retorno:** nenhum (`void`).

Arquivo com a solução: `Exercicio7$12.java`

Entrada

```
n1[0]: 7  
n2[0]: 9  
n1[1]: 10  
n2[1]: 5  
n1[2]: 2  
n2[2]: 2
```

Saída

```
7 e 9: 7 <= 9  
10 e 5: 5 <= 10  
2 e 2: 2 <= 2
```

Exercício 7.13:

Escreva um programa que leia 3 trincas de valores inteiros. Exibir cada trinca em ordem crescente. A ordem deve ser impressa através da chamada do método estático público “classificaTrinca” especificado abaixo:

- **Nome:** `classificaTrinca`
- **Descrição:** Imprime em ordem crescente três valores inteiros.
- **Entrada/Parâmetro(s):** `int n1, int n2, int n3`
- **Saída/Retorno:** nenhum (`void`).

Arquivo com a solução: `Exercicio7$13.java`

Entrada

```
n1[0]: 9  
n2[0]: 5  
n3[0]: 1  
n1[1]: 8  
n2[1]: 7  
n3[1]: 6  
n1[2]: 3  
n2[2]: 3  
n3[2]: 3
```

Saída

```
9, 5 e 1: 1 <= 5 <= 9  
8, 7 e 6: 6 <= 7 <= 8  
3, 3 e 3: 3 <= 3 <= 3
```

Exercício 7.14:

Escreva um programa que leia 5 duplas de valores inteiros. Após a leitura de todos os elementos, imprimir as duplas que foram armazenadas nas posições pares em ordem crescente e aquelas armazenadas nas posições ímpares em ordem decrescente. Utilize o método estático público “imprimeDuplaClassificada” especificado abaixo para escrever os elementos na ordem desejada.

- **Nome:** `imprimeDuplaClassificada`
- **Descrição:** Imprime os dois inteiros fornecidos na ordem desejada. A ordem é especificada através do parâmetro `emOrdemCrescente`.
- **Entrada/Parâmetro(s):** `int n1, int n2, bool emOrdemCrescente`
- **Saída/Retorno:** nenhuma (`void`).

Arquivo com a solução: Exercicio7\$14.java

Entrada

```
n1[0]: 7  
n2[0]: 9  
n1[1]: 9  
n2[1]: 7  
n1[2]: 8  
n2[2]: 2  
n1[3]: 6  
n2[3]: 4  
n1[4]: 9  
n2[4]: 9
```

Saída

```
7 e 9: 7 <= 9  
9 e 7: 9 >= 7  
8 e 2: 2 <= 8  
6 e 4: 6 >= 4  
9 e 9: 9 <= 9
```


CAPÍTULO

8

CARACTERES E STRINGS

“Adapte os atos às palavras, as palavras aos atos”.

William Shakespeare



linguagem de programação Java é capaz de lidar com dados do tipo caractere, como já vimos em algumas situações nos Capítulos anteriores, mas também tem capacidade para lidar com cadeias de caracteres, as chamadas Strings. Nesse Capítulo você aprenderá como manipular caracteres e como utilizar e manipular Strings na linguagem Java.

8.1 Exemplos em Linguagem Java

Métodos para manipulação de caracteres

```

1  /*
2   * Arquivo: capitulo08/CaracteresStringsManipulacaoCaracteres.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6  public class CaracteresStringsManipulacaoCaracteres {
7
8      public static void main( String[] args ) {
9
10         /*
11          * boolean isLetter( char c )
12          *
13          * Retorna true se c for uma letra ou falso em caso contrário.
14          */
15         System.out.printf ( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
16             "De acordo com Character.isLetter: ",
17             Character.isLetter('A') ?
18                 "A eh uma " : "A nao eh uma ", "letra",
19                 Character.isLetter('b') ?
20                     "b eh uma " : "b nao eh uma ", "letra",
21                     Character.isLetter('&') ?
22                         "& eh uma " : "& nao eh uma ", "letra",
23                         Character.isLetter('4') ?
24                             "4 eh uma " : "4 nao eh uma ", "letra" );
25
26         /*
27          * boolean isLetterOrDigit( char c )
28          *
29          * Retorna true se c for uma letra ou um dígito ou falso em
30          * caso contrário.
31          */
32         System.out.printf( "%s\n%s%s\n%s%s\n%s%s\n\n",
33             "De acordo com Character.isLetterOrDigit: ",
34             Character.isLetterOrDigit('A') ?
35                 "A eh um " : "A nao eh um ", "digito ou uma letra",
36                 Character.isLetterOrDigit('8') ?
37                     "8 eh um " : "8 nao eh um ", "digito ou uma letra",
38                     Character.isLetterOrDigit('#') ?
39                         "# eh um " : "# nao eh um ", "digito ou uma letra" );
40
41         /*
42          * boolean isDigit( char c )
43          *
44          * Retorna true se c for um dígito ou falso em caso contrário.
45          */

```

```
46     System.out.printf( "%s\n%s%s\n%s%s\n\n",
47         "De acordo com Character.isDigit: ",
48         Character.isDigit('8') ?
49             "8 eh um " : "8 nao eh um ", "digito",
50             Character.isDigit('#') ?
51                 "# eh um " : "# nao eh um ", "digito" );
52
53     /*
54      * boolean isLowerCase( char c )
55      *
56      * Retorna true se c for uma letra minúscula ou falso
57      * em caso contrário.
58      */
59     System.out.printf( "%s\n%s%s\n%s%s\n\n",
60         "De acordo com Character.isLowerCase:",
61         "a", Character.isLowerCase('a') ?
62             " eh um " : " nao eh um ",
63             "caractere em caixa baixa (minusculo)",
64             "A", Character.isLowerCase('A') ?
65                 " eh um " : " nao eh um ",
66                 "caractere em caixa baixa (minusculo)" );
67
68     /*
69      * boolean isUpperCase( char c )
70      *
71      * Retorna true se c for uma letra maiúscula ou falso em
72      * caso contrário.
73      */
74     System.out.printf( "%s\n%s%s\n%s%s\n\n",
75         "De acordo com Character.isUpperCase:",
76         "a", Character.isUpperCase('a') ?
77             " eh um " : " nao eh um ",
78             "caractere em caixa alta (maiusculo)",
79             "A", Character.isUpperCase('A') ?
80                 " eh um " : " nao eh um ",
81                 "caractere em caixa alta (maiusculo)" );
82
83     /*
84      * char toLowerCase( int c )
85      *
86      * Se c for uma letra maiúscula, toLowerCase retorna c como uma
87      * letra minúscula. Caso contrário, toLowerCase retorna o argumento
88      * inalterado.
89      */
90     System.out.printf( "%s\n%s%c\n%s%c\n%s%c\n\n",
91         "Usando Character.toLowerCase:",
92         "tolower('a'): ", Character.toLowerCase('a'),
93         "tolower('A'): ", Character.toLowerCase('A'),
94         "tolower('#'): ", Character.toLowerCase('#') );
```

```
95
96  /*
97   * char toUpperCase( int c )
98   *
99   * Se c for uma letra minúscula, toUpperCase retorna c como uma
100  * letra maiúscula. Caso contrário, toUpperCase retorna o argumento
101  * inalterado.
102  */
103 System.out.printf( "%s\n%s%c\n%s%c\n%s%c\n\n",
104                     "Usando Character.toUpperCase:",
105                     "toupper('a'): ", Character.toUpperCase('a'),
106                     "toupper('A'): ", Character.toUpperCase('A'),
107                     "toupper('#'): ", Character.toUpperCase('#') );
108
109 /*
110  * boolean isSpaceChar( int c )
111  *
112  * Retorna um valor verdadeiro se c for um caractere de espaço
113  * ou falso em caso contrário.
114  */
115 System.out.printf( "%s\n%s%s\n%s%s\n%s%s\n\n",
116                     "De acordo com Character.isSpaceChar:",
117                     "'Nova linha'",
118                     Character.isSpaceChar('\n') ? " eh um " : " nao eh um ",
119                     "caractere de espaco",
120                     "'Tab. hor.'",
121                     Character.isSpaceChar('\t') ? " eh um " : " nao eh um ",
122                     "caractere de espaco",
123                     Character.isSpaceChar(' ') ? "' eh um " : "% nao eh um ",
124                     "caractere de espaco",
125                     Character.isSpaceChar('%') ? "% eh um " : "% nao eh um ",
126                     "caractere de espaco" );
127
128 /*
129  * boolean isWhitespace( int c )
130  *
131  * Retorna um valor verdadeiro se c for um caractere de espaço
132  * em branco ou falso em caso contrário.
133  */
134 System.out.printf( "%s\n%s%s\n%s%s\n%s%s\n\n",
135                     "De acordo com Character.isWhitespace:",
136                     "'Nova linha'",
137                     Character.isWhitespace('\n') ? " eh um " : " nao eh um ",
138                     "caractere de espaco em branco",
139                     "'Tab. hor.'",
140                     Character.isWhitespace('\t') ? " eh um " : " nao eh um ",
141                     "caractere de espaco em branco",
142                     Character.isWhitespace(' ') ? "' eh um " : "% nao eh um ",
143                     "caractere de espaco em branco",
```

```
144     Character.isWhitespace('%') ? "% eh um" : "% nao eh um",
145     "caractere de espaco em branco" );
146 }
147 }
148 }
149 }
```

Métodos para conversão de Strings em valores numéricos

```
1  /*
2   * Arquivo: capitulo08/CaracteresStringsConversoes.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6  public class CaracteresStringsConversoes {
7
8      public static void main( String[] args ) {
9
10         /*
11          * A conversão de Strings em tipos primitivos se dá
12          * usando as classes empacotadoras (wrappers) dos tipos primitivos.
13          */
14         byte vByte = Byte.parseByte( "1" );
15         short vShort = Short.parseShort( "2" );
16         int vInt = Integer.parseInt( "3" );
17         long vLong = Long.parseLong( "4" );
18         float vFloat = Float.parseFloat( "12.5" );
19         double vDouble = Double.parseDouble( "29.75" );
20         boolean vBoolean = Boolean.parseBoolean( "true" );
21
22         // imprimindo...
23         System.out.println( "    vByte: " + vByte );
24         System.out.println( "    vShort: " + vShort );
25         System.out.println( "    vInt: " + vInt );
26         System.out.println( "    vLong: " + vLong );
27         System.out.println( "    vFloat: " + vFloat );
28         System.out.println( "    vDouble: " + vDouble );
29         System.out.println( "vBoolean: " + vBoolean );
30
31     }
32
33 }
```

Conceitos, entrada e saída de Strings

```
1  /*
2   * Arquivo: capitulo08/CaracteresStringsConceitosES.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6 import java.util.Scanner;
7
8 public class CaracteresStringsConceitosES {
9
10    public static void main( String[] args ) {
11
12        Scanner scan = new Scanner( System.in );
13
14        // declarando uma variável do tipo String e a inicializando.
15        String nomeCompleto = "David Buzatto";
16
17        // declarando uma String sem inicializar
18        String profissao;
19
20        // um array de 5 Strings
21        String[] conjuntoStrings = new String[5];
22
23        /*
24         * Imprimindo as strings usando o especificador
25         * de formato %s
26         */
27        System.out.printf( "%s\n", nomeCompleto );
28
29        // imprimindo direto
30        System.out.println( nomeCompleto );
31
32        // uma string pode ser "quebrada" para fins de visibilidade.
33        System.out.println(
34            "Essa eh uma string que ficou feia no " +
35            "codigo, pois eh muito comprida e " +
36            "dificulta a leitura, entendeu?" );
37
38        // a leitura de Strings é feita usando também a classe Scanner
39        System.out.print( "Entre com sua profissao: " );
40        profissao = scan.nextLine();
41        System.out.println( "A profissao fornecida foi: " + profissao );
42
43        for ( int i = 0; i < 5; i++ ) {
44            System.out.printf( "conjuntoStrings[%d]: ", i );
45            conjuntoStrings[i] = scan.nextLine();
46        }
47
```

```
48     for ( int i = 0; i < 5; i++ ) {
49         imprimeCaixa( conjuntoStrings[i], 30 );
50     }
51
52     scan.close();
53 }
54
55
56 public static void imprimeCaixa( String str, int largura ) {
57
58     int c = largura - 3 - str.length();
59
60     System.out.print( "+" );
61     for ( int i = 0; i < largura-2; i++ ) {
62         System.out.print( "-" );
63     }
64     System.out.print( "+\n" );
65
66     System.out.print( " | " + str );
67     for ( int i = 0; i < c; i++ ) {
68         System.out.print( " " );
69     }
70     System.out.print( "| \n" );
71
72     System.out.print( "+" );
73     for ( int i = 0; i < largura-2; i++ ) {
74         System.out.print( "-" );
75     }
76     System.out.print( "+\n" );
77 }
78 }
79
80 }
```

Métodos para manipulação de Strings

```
1  /*
2   * Arquivo: capitulo08/CaracteresStringsMetodos.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6  public class CaracteresStringsMetodos {
7
8      public static void main( String[] args ) {
9
10         /*
11          * Em Java, as Strings são imutáveis, ou seja, depois de criadas
12          * não são mais modificadas. Todos os métodos que operam sobre
13          * as Strings gerarão novos valores baseados na String original.
14          */
15         String string1 = "David";
16         String string2 = "Aurora";
17         String string3 = "Buzatto";
18         String testes;
19         int comparacao;
20
21
22         System.out.println( "**** operacoes ****" );
23         System.out.println( "Concatenacao:" );
24
25         // a concatenação de Strings é feita usando o operador +
26         string1 = string1 + " " + string3;
27         System.out.printf( "      %s\n", string1 );
28
29         // pode-se também usar o operador de atribuição composta +=
30         string2 += " " + string3;
31         System.out.printf( "      %s\n", string2 );
32
33
34         System.out.println( "\n" );
35         System.out.println( "**** extracao de dados ****" );
36
37         /*
38          * int length()
39          *
40          * Retorna o tamanho da String (quantidade de
41          * caracteres armazenados).
42          */
43         System.out.println( "length (comprimento):" );
44         System.out.printf( "      A string \'%s\' possui %d caracteres.\n",
45                           string1, string1.length() );
46         System.out.println();
47     }
```

```
48
49  /*
50   * char charAt( int posicao )
51   *
52   * Retorna o caractere em determinada posição.
53   */
54 System.out.println( "charAt (caractere de uma posicao):" );
55 System.out.println( "    Caractere na posicao 2: " +
56                     string1.charAt( 2 ) );
57 System.out.println();
58
59 /*
60  * int indexOf( char caractere )
61  *
62  * Retorna a posição do primeiro caractere encontrado ou -1 caso
63  * o caractere não exista na String.
64  */
65 System.out.println( "indexOf (posicao de caractere):" );
66 System.out.println( "    Posicao do caractere u: " +
67                     string2.indexOf( 'u' ) );
68 System.out.println( "    Posicao do caractere x: " +
69                     string2.indexOf( 'x' ) );
70 System.out.println();
71
72
73 /*
74  * int indexOf( String substring )
75  *
76  * Retorna a posição onde substring passada começa na String
77  * chamadora ou -1 caso a substring não exista na String.
78  */
79 System.out.println( "indexOf (posicao de substring):" );
80 System.out.println( "    Posicao da substring vi: " +
81                     string1.indexOf( "vi" ) );
82 System.out.println( "    Posicao do substring dx: " +
83                     string1.indexOf( "dx" ) );
84 System.out.println();
85
86
87 /*
88  * int lastIndexOf( char caractere )
89  *
90  * Retorna a posição do último caractere encontrado ou -1 caso
91  * o caractere não exista na String.
92  */
93 System.out.println( "lastIndexOf (ultima posicao de caractere):" );
94 System.out.println( "    Ultima posicao do caractere a: " +
95                     string2.lastIndexOf( 'a' ) );
```

```
97     System.out.println( "    Ultima posicao do caractere x: " +
98         string2.lastIndexOf( 'x' ) );
99     System.out.println();
100
101
102 /**
103 * int lastIndexOf( String substring )
104 *
105 * Retorna a posição onde a última substring passada começa
106 * na String chamadora ou -1 caso a substring não exista na String.
107 */
108 System.out.println( "lastIndexOf (ultima posicao de substring):" );
109 System.out.println( "    Ultima posicao da substring id: " +
110     string1.lastIndexOf( "id" ) );
111 System.out.println( "    Ultima posicao do substring dx: " +
112     string1.lastIndexOf( "dx" ) );
113 System.out.println();
114
115
116 /**
117 * char[] toCharArray()
118 *
119 * Retorna um array de caracteres com todos os caracteres da
120 * String chamadora.
121 */
122 System.out.println( "toCharArray (gerar array de caracteres):" );
123 System.out.print( "    Caracteres de: " + string2 + ": " );
124 for ( char c : string2.toCharArray() ) {
125     System.out.print( c + " " );
126 }
127 System.out.println( "\n" );
128
129
130 /**
131 * String[] split( String regex )
132 *
133 * Divide a String chamadora, retornando um array de Strings
134 * baseado em uma expressão regular passada como parâmetro.
135 */
136 System.out.println( "split (dividir em substrings):" );
137 testes = "essa eh uma frase de testes";
138 System.out.print( "    Palavras de: " + testes + ": " );
139 for ( String s : testes.split( " " ) ) {
140     System.out.print( s + " - " );
141 }
142 System.out.println( "\n" );
143
144
145 System.out.println();
```

```
146     System.out.println( "**** verificacoes ****" );
147
148     /*
149      * boolean equals( String outra )
150      *
151      * Retorna verdadeiro caso a String chamadora tenha o mesmo
152      * conteúdo da String outra ou falso em caso contrário.
153      */
154     System.out.println( "equals (sao iguais?):" );
155     testes = "aurora buzatto";
156     if ( string2.equals( testes ) ) {
157         System.out.println(
158             "    " + string2 + " e " + testes +
159             " sao iguais (mesmo conteudo)" );
160     } else {
161         System.out.println(
162             "    " + string2 + " e " + testes +
163             " sao diferentes (conteudo diferente)" );
164     }
165     System.out.println();
166
167
168     /*
169      * boolean equalsIgnoreCase( String outra )
170      *
171      * Retorna verdadeiro caso a String chamadora tenha o mesmo
172      * conteúdo, ignorando a caixa das letras, da String outra ou
173      * falso em caso contrário.
174      */
175     System.out.println( "equalsIgnoreCase (sao iguais?):" );
176     if ( string2.equalsIgnoreCase( testes ) ) {
177         System.out.println(
178             "    " + string2 + " e " + testes +
179             " sao iguais (mesmo conteudo)" );
180     } else {
181         System.out.println(
182             "    " + string2 + " e " + testes +
183             " sao diferentes (conteudo diferente)" );
184     }
185     System.out.println();
186
187
188     /*
189      * boolean contains( String substring )
190      *
191      * Retorna verdadeiro a substring exista na String chamadora
192      * ou falso em caso contrário.
193      */
194     System.out.println( "contains (contem?):" );
```

```
195     System.out.printf( "      \"%s\" contem '\\w+'? %b\n", testes,
196                         testes.matches( "\\w+" ) );
197     testes = "abcdef";
198     System.out.printf( "      \"%s\" contem '\\w+'? %b\n", testes,
199                         testes.matches( "\\w+" ) );
200     System.out.println();
201
202
203     /*
204      * boolean isBlank
205      *
206      * Retorna verdadeiro caso a String seja composta de apenas
207      * caracteres brancos (espaço, pulo de linha etc.) ou falso
208      * em caso contrário.
209      */
210    System.out.println( "isBlank (eh branca?):" );
211    System.out.println( "      Em branco? " + "".isBlank() );
212    System.out.println( "      Em branco? " + " ".isBlank() );
213    System.out.println( "      Em branco? " + "\n \t \f ".isBlank() );
214    System.out.println( "      Em branco? " + " a ".isBlank() );
215    System.out.println();
216
217
218     /*
219      * boolean isEmpty
220      *
221      * Retorna verdadeiro caso a String seja vazia (comprimento 0)
222      * ou falso em caso contrário.
223      */
224    System.out.println( "isEmpty (esta vazia?):" );
225    System.out.println( "      Vazia? " + "".isEmpty() );
226    System.out.println( "      Vazia? " + " ".isEmpty() );
227    System.out.println( "      Vazia? " + "\n \t \f ".isEmpty() );
228    System.out.println( "      Vazia? " + " a ".isEmpty() );
229    System.out.println();
230
231
232     /*
233      * int compareTo( String outraString )
234      * Compara a String que invocou o método com outraString e retorna:
235      *      um valor negativo, caso quem invocou venha antes de
236      *      outraString;
237      *      zero, caso quem invocou seja igual a outraString;
238      *      um valor positivo, caso quem invocou venha após de
239      *      outraString;
240      */
241    System.out.println( "compareTo (comparar):" );
242    comparacao = string1.compareTo( string2 );
243    if ( comparacao < 0 ) {
```

```
244         System.out.printf( "      %s vem antes de %s\n",
245                         string1, string2 );
246     } else if ( comparacao > 0 ) {
247         System.out.printf( "      %s vem antes de %s\n",
248                         string2, string1 );
249     } else {
250         System.out.printf( "      %s e %s tem o mesmo conteudo!\n",
251                         string1, string2 );
252     }
253     System.out.println();
254
255
256     /*
257      * boolean matches( String regex )
258      *
259      * Retorna verdadeiro caso haja o casamento (match) da expressão
260      * regular ou falso em caso contrário.
261      */
262     System.out.println( "matches (ha casamento?):" );
263     System.out.printf( "      \"%s\" => %b\n", testes,
264                         testes.matches( "\\w+" ) );
265     testes = "abcdef";
266     System.out.printf( "      \"%s\" => %b\n", testes,
267                         testes.matches( "\\w+" ) );
268     System.out.println();
269
270
271     System.out.println();
272     System.out.println( "**** transformacoes ****" );
273
274     /*
275      * String substring( int inicio )
276      *
277      * Retorna a substring iniciada na posição inicio até o fim da
278      * String original. A posição inicio é inclusiva.
279      */
280     System.out.println( "substring (subcadeia):" );
281     System.out.println( "      substring (inicio): " +
282                         string1.substring( 7 ) );
283
284
285     /*
286      * String substring( int inicio, int fim )
287      *
288      * Retorna a substring iniciada na posição inicio até a posição
289      * fim da String original. A posição inicio é inclusiva e a posição
290      * fim é exclusiva.
291      */
292     System.out.println( "      substring (inicio, fim): " +
```

```
293         string2.substring( 2, 9 ) );
294     System.out.println();
295
296
297     /*
298      * String toLowerCase()
299      *
300      * Retorna uma nova String com todos os caracteres em minúsculo.
301      */
302     System.out.println( "toLowerCase (para minusculas):" );
303     System.out.printf( "    \"%s\" => \"%s\"\n\n",
304                        string2, string2.toLowerCase() );
305
306
307     /*
308      * String toUpperCase
309      *
310      * Retorna uma nova String com todos os caracteres em maiúsculo.
311      */
312     System.out.println( "toUpperCase (para maiusculas):" );
313     System.out.printf( "    \"%s\" => \"%s\"\n\n",
314                        string2, string2.toUpperCase() );
315
316
317     /*
318      * String trim()
319      *
320      * Retorna uma nova String removendo todos os espaços do início
321      * e do fim da mesma.
322      */
323     System.out.println( "trim (apurar):" );
324     testes = "    " + string1 + "    ";
325     System.out.printf( "    \"%s\" => \"%s\"\n\n",
326                        testes, testes.trim() );
327
328
329     /*
330      * String replace( char antigo, char novo )
331      *
332      * Retorna uma nova String substituindo todas as ocorrências do
333      * caractere antigo pelo novo.
334      */
335     System.out.println( "replace (substituir caractere):" );
336     System.out.printf( "    \"%s\" => \"%s\"\n", string1,
337                        string1.replace( 'd', 'x' ) );
338     System.out.println();
339
340
341     */
```

```
342     * String replace( String antigo, String novo )
343     *
344     * Retorna uma nova String substituindo todas as ocorrências da
345     * substring antigo pelo novo.
346     */
347 System.out.println( "replace (substituir strings):");
348 System.out.printf( "    \"%s\" => \"%s\"\n\n", string2,
349                     string2.replace( "ro", "xy" ) );
350
351 /*
352     * String replaceAll( String regex, String novo )
353     *
354     * Retorna uma nova String substituindo todas os casamentos
355     * (matches) da expressão regular por novo.
356     */
357 System.out.println( "replaceAll (substituir todos):");
358 testes = "essa eh uma frase de testes";
359 System.out.printf( "    \"%s\" => \"%s\"\n\n", testes,
360                     testes.replaceAll( "\\w{2}", "+" ) );
361
362
363 /*
364     * String format( String formato, Object... valores );
365     *
366     * O método público estático format da classe String é
367     * utilizado para Strings. Ele funciona de forma análoga
368     * ao método System.out.printf, entretanto, ao invés de
369     * direcionar os dados para a saída, ele gera uma String
370     * com os valores formatados e a retorna.
371     *
372     * Todas as regras de formatação aplicadas no printf
373     * são aplicadas ao format.
374     */
375 System.out.println( "format (formatacao):" );
376 String str = String.format( "%02d/%02d/%04d", 25, 2, 1985 );
377 System.out.println( str );
378
379 }
380 }
381 }
```

Blocos de Texto

```
1  /*
2   * Arquivo: capitulo08/CaracteresStringsBlocosTexto.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6  public class CaracteresStringsBlocosTexto {
7
8      public static void main( String[] args ) {
9
10         /*
11          * A partir da versão 15 da linguagem Java foi adicionado
12          * um novo artifício sintático que permite a criação de
13          * blocos de texto, facilitando a definição de Strings que
14          * contém muitos pulos de linha. Imagine uma String que
15          * contém como texto esse comentário. Antes da versão 15,
16          * teríamos que fazer da seguinte maneira:
17          */
18
19         String textoString =
20             "A partir da versão 15 da linguagem Java foi adicionado\n" +
21             "um novo artifício sintático que permite a criação de\n" +
22             "blocos de texto, facilitando a definição de Strings que\n" +
23             "contém muitos pulos de linha. Imagine uma String que\n" +
24             "contém como texto esse comentário. Antes da versão 15,\n" +
25             "teríamos que fazer da seguinte maneira:";
26
27
28         /*
29          * Obviamente poderíamos inicializar essa String com um conteúdo
30          * sem quebras (tudo em uma linha), mas isso dificultaria a leitura
31          * do código. Esse é apenas um exemplo básico, pois não quero
32          * misturar vários assuntos aqui, mas um exemplo mais útil seria
33          * uma String que contém um trecho de um documento HTML. Veja agora,
34          * a mesma String anterior inicializada com a sintaxe de bloco de
35          * texto, que consiste em uma String delimitada entre um par de três
36          * aspas duplas ("\"").
37          */
38
39         String textoUsandoBloco =
40             """
41                 A partir da versão 15 da linguagem Java foi adicionado
42                 um novo artifício sintático que permite a criação de
43                 blocos de texto, facilitando a definição de Strings que
44                 contém muitos pulos de linha. Imagine uma String que
45                 contém como texto esse comentário. Antes da versão 15,
46                 teríamos que fazer da seguinte maneira:
47                 """;
```

```
48     System.out.println( textoUsandoBloco );
49
50     /*
51      * Note que ao inicializar a String dessa forma, ganhamos em
52      * legibilidade. A margem esquerda da String sempre será dada
53      * a partir do caractere válido mais à esquerda do bloco. Caso
54      * → queira
55      * que caracteres de espaço sejam adicionados ao final de uma linha,
56      * há a necessidade de inserir o caractere de escape \s.
57      */
58     String blocoUsandoEspaco =
59     """
60         Bloco de texto com três espaços no fim desta linha.\s\s\s
61         Dois espaços no início desta linha.
62         Nenhum nesta.
63         """;
64
65     // verifique a saída do for abaixo ao executar essa classe!
66     for ( String s : blocoUsandoEspaco.split( "\n" ) ) {
67         System.out.println( "***" + s + "***" );
68     }
69 }
70 }
71 }
```

Processamento de Parâmetros Via Linha de Comando

```
/*
 * Arquivo: capitulo08/CaracteresStringsArgumentosLinhaComando.java
 * Autor: Prof. Dr. David Buzatto
 */

public class CaracteresStringsArgumentosLinhaComando {

    /*
     * Através do parâmetro args o método main consegue processar
     * argumentos enviados na linha de comando ao se executar a classe
     * compilada.
     */
    public static void main( String[] args ) {

        int n1 = 0;
        int n2 = 0;

        if ( args.length > 0 ) {

            switch ( args[0] ) {

                case "/?":
                    System.out.println(
                        "Programa desenvolvido por David Buzatto." );
                    break;

                case "/calc":
                    if ( args.length == 4 ) {

                        try {
                            n1 = Integer.parseInt( args[1] );
                        } catch ( NumberFormatException exc ) {
                            System.out.println(
                                "Voce precisa fornecer numeros inteiros!" );
                        }

                        try {
                            n2 = Integer.parseInt( args[3] );
                        } catch ( NumberFormatException exc ) {
                            System.out.println(
                                "Voce precisa fornecer numeros inteiros!" );
                        }

                        switch ( args[2] ) {
                            case "+":
                                System.out.printf(
                                    "%d + %d = %d", n1, n2, n1 + n2 ) ;
                        }
                    }
            }
        }
    }
}
```

```
48         break;
49     case "-":
50         System.out.printf(
51             "%d - %d = %d", n1, n2, n1 - n2 ) ;
52         break;
53     case "*":
54         System.out.printf(
55             "%d * %d = %d", n1, n2, n1 * n2 ) ;
56         break;
57     case "/":
58         System.out.printf(
59             "%d / %d = %d", n1, n2, n1 / n2 ) ;
60         break;
61     case "pow":
62         System.out.printf(
63             "Math.pow( %d, %d ) = %d", n1, n2,
64             (int) Math.pow( n1, n2 ) );
65         break;
66     default:
67         System.out.println( "Operador invalido!" );
68     }
69
70     }
71     break;
72 }
73 }
74 }
75 }
76 }
77 }
78 }
```

| Saiba Mais

A documentação da classe `Character` e de seus métodos pode ser encontrada aqui:
[<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Character.html>](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Character.html)

| Saiba Mais

A documentação da classe `String` e de seus métodos pode ser encontrada aqui:
[<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html>](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html)

8.2 Exercícios

Exercício 8.1:

Escreva um programa para ler uma string e apresentar seus quatro primeiros caracteres.

Arquivo com a solução: Exercicio8\$1.java

Entrada

```
String: essa eh uma string
```

Saída

```
e, s, s, a.
```

Exercício 8.2:

Escreva um programa para ler uma sentença e apresentar:

- O primeiro caractere da sentença;
- O último caractere da sentença;
- O número de caracteres existente na sentença.

Arquivo com a solução: Exercicio8\$2.java

Entrada

Sentenca: ola, como vai, tudo bem?

Saída

Primeiro caractere: o
Ultimo caractere: ?
Número de caracteres: 24

Exercício 8.3:

Escreva um programa para ler uma sentença e imprimir todos os seus caracteres das posições pares. Se algum caractere for um espaço, imprima-o cercado de aspas simples.

Arquivo com a solução: Exercicio8\$3.java

Entrada

Sentenca: um dois tres

Saída

u, ' ', o, s, t, e

Exercício 8.4:

Escreva um programa para ler uma sentença e imprimir todos os seus caracteres das posições ímpares.

Arquivo com a solução: Exercicio8\$4.java

Entrada

```
Sentenca: um dois tres
```

Saída

```
m, d, i, ' ', r, s
```

Exercício 8.5:

Escreva um programa para ler um nome e imprimi-lo 5 vezes, um por linha.

Arquivo com a solução: Exercicio8\$5.java

Entrada

Nome: Fernanda

Saída

Fernanda
Fernanda
Fernanda
Fernanda
Fernanda

Exercício 8.6:

Escreva um programa que receba um nome e que o imprima tantas vezes quanto forem seus caracteres.

Arquivo com a solução: Exercicio8\$6.java

Entrada

Nome: Aurora

Saída

Aurora
Aurora
Aurora
Aurora
Aurora
Aurora

Exercício 8.7:

Escreva um programa que leia 5 pares de strings e que imprima:

- IGUAIS, se as strings do par forem iguais;
- ORDEM CRESCENTE, se as strings do par foram fornecidas em ordem crescente;
- ORDEM DECRESCENTE, se as strings do par foram fornecidas em ordem decrescente.

Arquivo com a solução: Exercicio8\$7.java

Entrada

```
Par 1, palavra 1: Joao
Par 1, palavra 2: Maria
Par 2, palavra 1: Fernanda
Par 2, palavra 2: David
Par 3, palavra 1: Rafaela
Par 3, palavra 2: Rafaela
Par 4, palavra 1: Renata
Par 4, palavra 2: Cecilia
Par 5, palavra 1: Joana
Par 5, palavra 2: Zelia
```

Saída

```
Joao - Maria: ORDEM CRESCENTE
Fernanda - David: ORDEM DECRESCENTE
Rafaela - Rafaela: IGUAIS
Renata - Cecilia: ORDEM DECRESCENTE
Joana - Zelia: ORDEM CRESCENTE
```

Exercício 8.8:

Escreva um programa que leia três strings. A seguir imprimir as 3 strings em ordem alfabética.

Arquivo com a solução: Exercicio8\$8.java

Entrada

```
String 1: Luiz  
String 2: Everton  
String 3: Breno
```

Saída

```
Breno, Everton e Luiz
```

Exercício 8.9:

Escreva um programa para ler uma string. A seguir gere uma outra string que é o inverso da string original e imprima-a. Para isso, implemente e utilize o método `inverter`, apresentado abaixo:

- `public static String inverter(String origem)`

Arquivo com a solução: Exercicio8\$9.java

Entrada

String: abacate verde

Saída

Invertida: edrev etacaba

Exercício 8.10:

Escreva um programa para ler um caractere e logo após uma frase. Para cada frase informada, imprimir o número de ocorrências do caractere na frase. O programa deve ser encerrado quando a frase digitada for a palavra “fim”, que por sua vez não deve ter as ocorrências do caractere informado contadas. A contagem de ocorrências deve ser feita pelo método `contarOcorrencias` implementado por você e apresentado abaixo. Note que para cada frase informada, o programa deve gerar a saída correspondente imediatamente. Essa característica está denotada pelo uso de cores diferentes na entrada e na saída. Seu programa não deve alterar cor alguma!

- `public static int contarOcorrencias(String str, char c)`

Arquivo com a solução: Exercicio8\$10.java

Entrada

```
Caractere: a
Frase: camarao assado
Frase: mas que cabelo sujo!
Frase: fim
```

Saída

```
"camarao assado" tem 5 ocorrencia(s) do caractere 'a'
"mas que cabelo sujo!" tem 2 ocorrencia(s) do caractere 'a'
```

Exercício 8.11:

Escreva um programa para ler uma frase e contar o número de ocorrências de cada uma das 5 primeiras letras do alfabeto (tanto maiúsculas quanto minúsculas) e imprimir essas contagens. Você pode usar o método `contarOcorrencias` implementado anteriormente.

Arquivo com a solução: Exercicio8\$11.java

Entrada

Frase: UI, QUE medo DO white walker!

Saída

A/a: 1
B/b: 0
C/c: 0
D/d: 2
E/e: 4

Exercício 8.12:

Escreva um programa para ler uma frase e contar o número de ocorrências das letras A, E, I, O e U (tanto maiúsculas quanto minúsculas) e imprimir essas contagens. Você pode usar o método `contarOcorrencias` implementado anteriormente.

Arquivo com a solução: Exercicio8\$12.java

Entrada

Frase: UI, QUE medo DO white walker!

Saída

A/a: 1
E/e: 4
I/i: 2
O/o: 2
U/u: 2

Exercício 8.13:

Escreva um programa para ler uma frase. A seguir, apresente a mesma frase com todas as letras em maiúsculas.

Arquivo com a solução: Exercicio8\$13.java

Entrada

Frase: Fui comprar um COMPUTADOR.

Saída

FUI COMPRAR UM COMPUTADOR.

Exercício 8.14:

Escreva um programa para ler uma frase. A seguir, apresente a mesma frase com todas as letras em minúsculas.

Arquivo com a solução: Exercicio8\$14.java

Entrada

Frase: Fui comprar um COMPUTADOR.

Saída

fui comprar um computador.

Exercício 8.15:

Escreva um programa para ler uma frase e um caractere. A seguir gerar uma nova string com o conteúdo da frase fornecida, mas sem todas as letras iguais (tanto as ocorrências maiúsculas quanto minúsculas) à informada e imprimir essa nova string.

Arquivo com a solução: Exercicio8\$15.java

Entrada

Frase: ja acabou, JESSICA?

Caractere: a

Saída

j cbou, JESSIC?

Exercício 8.16:

Escreva um programa para ler uma frase e contar o número de palavras existentes na frase. Considere palavra um conjunto qualquer de caracteres separados por um conjunto qualquer de espaços em branco. A contagem deve ser feita por meio do método `contarPalavras`, apresentado abaixo:

- `public static int contarPalavras(String str)`

Arquivo com a solução: Exercicio8\$16.java

Entrada

Frase: A aranha arranha a ra.

Saída

Quantidade de palavras: 5

Exercício 8.17:

Escreva um programa que leia uma string e verifique se a mesma é um palíndromo. Um palíndromo é toda sentença que pode ser lida da mesma forma da esquerda para a direita e vice-versa. Letras maiúsculas e minúsculas devem ser diferenciadas, ou seja, o caractere 'a' é diferente do caractere 'A'. Implemente para isso o método `ehPalindromo`, apresentado abaixo:

- `public static boolean ehPalindromo(String str)`

Arquivo com a solução: Exercicio8\$17.java

Entrada

String: arara

Saída

"arara" eh um palindromo!

Entrada

String: Arara

Saída

"Arara" nao eh um palindromo!

Entrada

String: ArarA

Saída

"ArarA" eh um palindromo!

Entrada

String: Macaco

Saída

"Macaco" nao eh um palindromo!

Exercício 8.18:

Escreva um programa que recorte uma string com base em uma posição inicial e uma posição final. O índice inicial é inclusivo e o final é exclusivo. Caso algum índice inválido seja fornecido, a string não deve ser recortada.

Arquivo com a solução: Exercicio8\$18.java

Entrada

```
String: Girafa  
Inicio: 0  
Fim: 3
```

Saída

```
Recorte: Gir
```

Entrada

```
String: Girafa  
Inicio: 5  
Fim: 3
```

Saída

```
Recorte: Girafa
```

Entrada

```
String: Girafa  
Inicio: 5  
Fim: 10
```

Saída

```
Recorte: Girafa
```

Entrada

```
String: Girafa  
Inicio: 10  
Fim: 12
```

Saída

```
Recorte: Girafa
```

Exercício 8.19:

Escreva um programa que leia duas strings e que verifique se a segunda está contida na primeira.

Arquivo com a solução: Exercicio8\$19.java

Entrada

```
String fonte: rabanete  
String a pesquisar: bane
```

Saída

```
"bane" esta contida em "rabanete"
```

Entrada

```
String: Hamburguer  
String a pesquisar: Hambo
```

Saída

```
"Hambo" nao esta contida em "Hamburguer"
```

Exercício 8.20:

Escreva um programa que leia uma string e que a imprima centralizada no terminal, considerando que o terminal tem 80 colunas. Para isso, implemente o método `imprimirCentralizado`, apresentado abaixo:

- `public static void imprimirCentralizado(String str)`

Arquivo com a solução: Exercicio8\$20.java

Entrada

String: um texto qualquer

Saída

um texto qualquer

Exercício 8.21:

Escreva um programa que leia uma string e que a imprima alinhada à direita no terminal, considerando que o terminal tem 80 colunas. Para isso, implemente o método `imprimirDireita`, apresentado abaixo:

- `public static void imprimirDireita(String str)`

Arquivo com a solução: Exercicio8\$21.java

Entrada

String: um texto qualquer

Saída

um texto qualquer

Exercício 8.22:

Escreva um programa que leia uma string e que a imprima dentro de uma caixa desenhada usando os símbolos =, + e |. Para isso, implemente o método `imprimirCaixa`, apresentado abaixo:

- `public static void imprimirCaixa(String str)`

Arquivo com a solução: Exercicio8\$22.java

Entrada

```
String: um texto qualquer
```

Saída

```
+=====+  
|| um texto qualquer ||  
+=====+
```

8.3 Exercícios Criativos

Exercício Criativo 8.1:

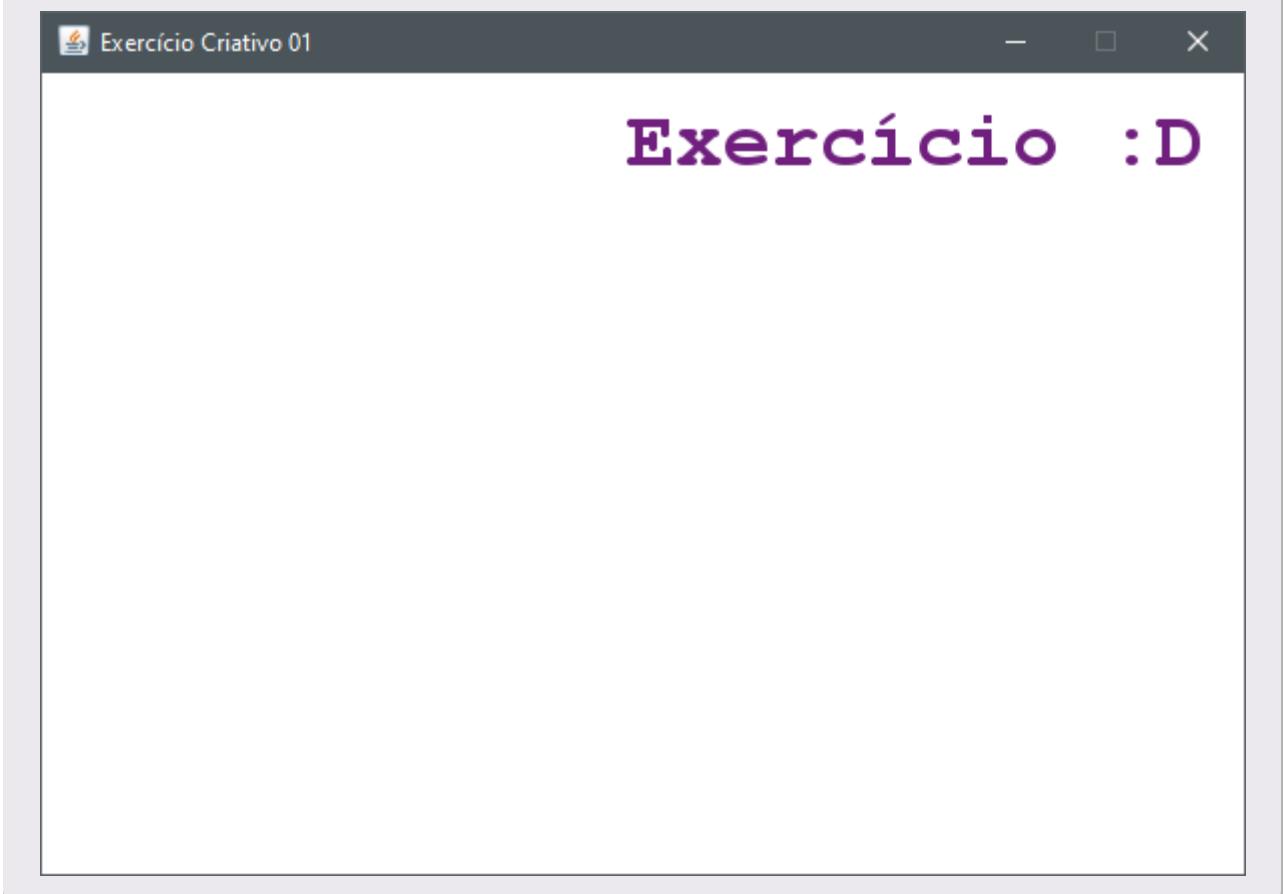
Escreva um programa que, dada uma string qualquer, à desenhe alinhada à direita da janela. Você está livre para usar as cores e as dimensões que quiser. Utilize o método `measureText`, apresentado abaixo, para calcular o tamanho (largura) da string em questão.

Método:

```
1 int measureText( String text, int fontSize );
```

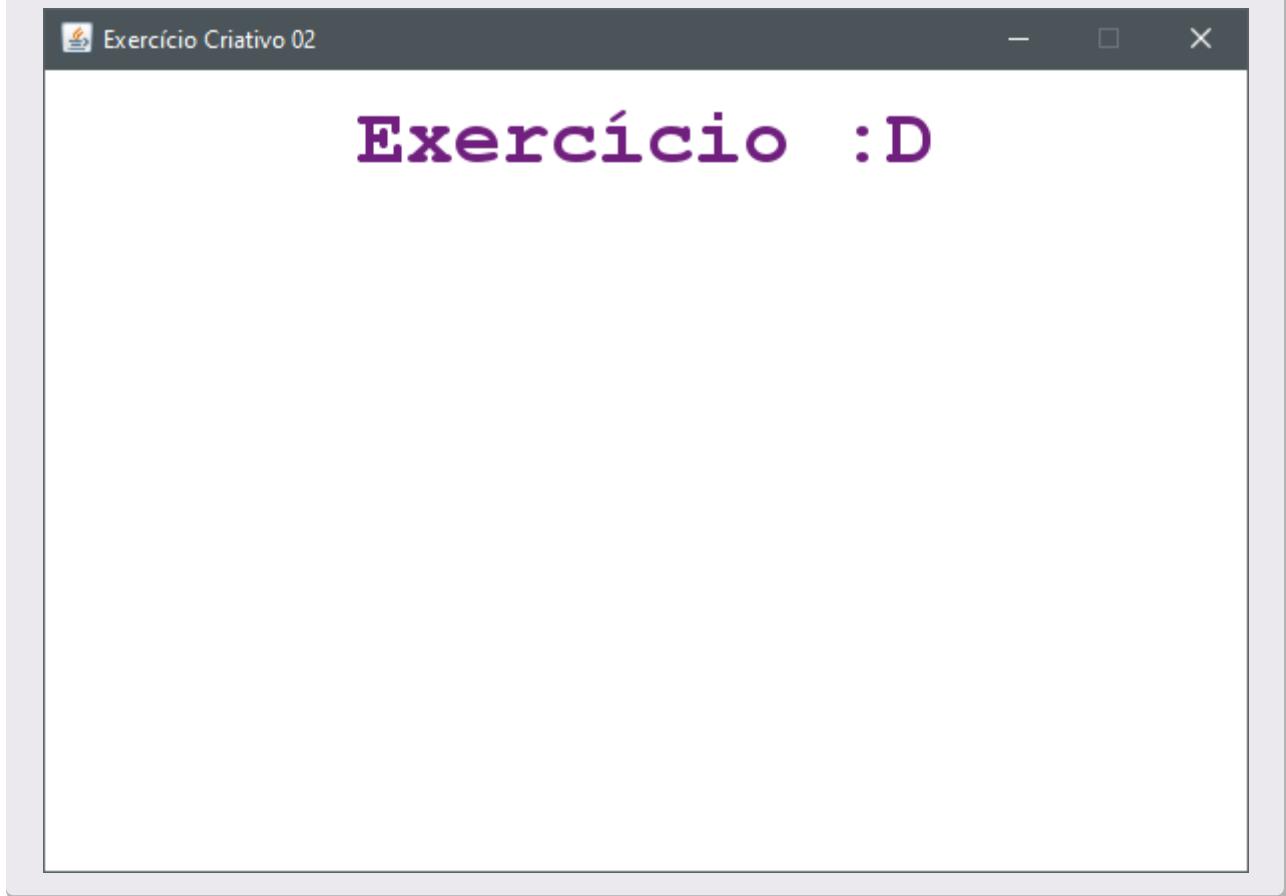
- **Nome:** `measureText`
- **Descrição:** Calcula a largura de uma string, baseada no tamanho da fonte passado.
- **Entrada/Parâmetro(s):**
 1. `text`: a string a ser medida;
 2. `fontSize`: o tamanho da fonte que está sendo utilizada.
- **Saída/Retorno:** a largura da string (`int`), medida com base na fonte atual e no tamanho fornecido.

Saída:



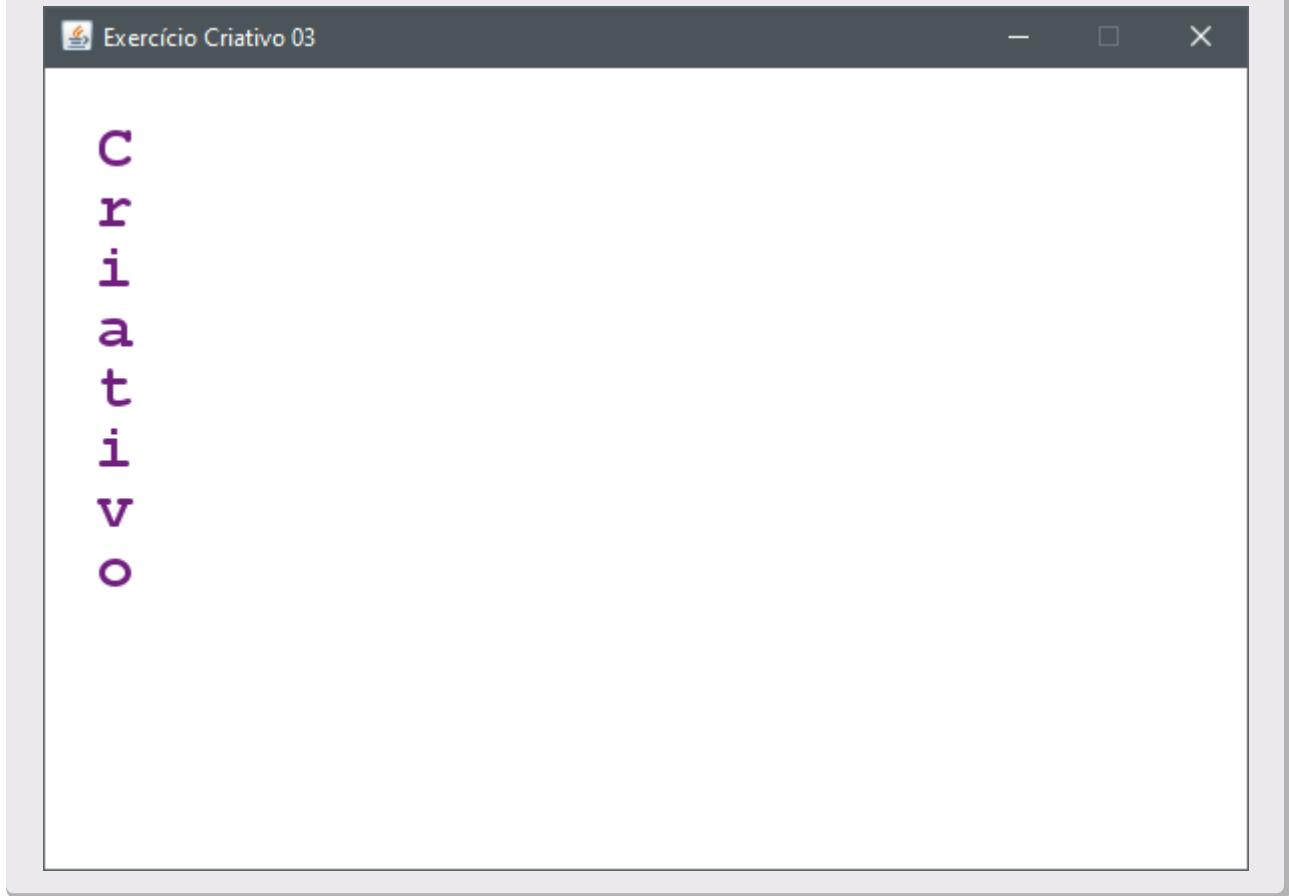
Exercício Criativo 8.2:

Escreva um programa que, dada uma string qualquer, à desenhe alinhada ao centro (horizontal) da janela. Você está livre para usar as cores e as dimensões que quiser.

Saída:

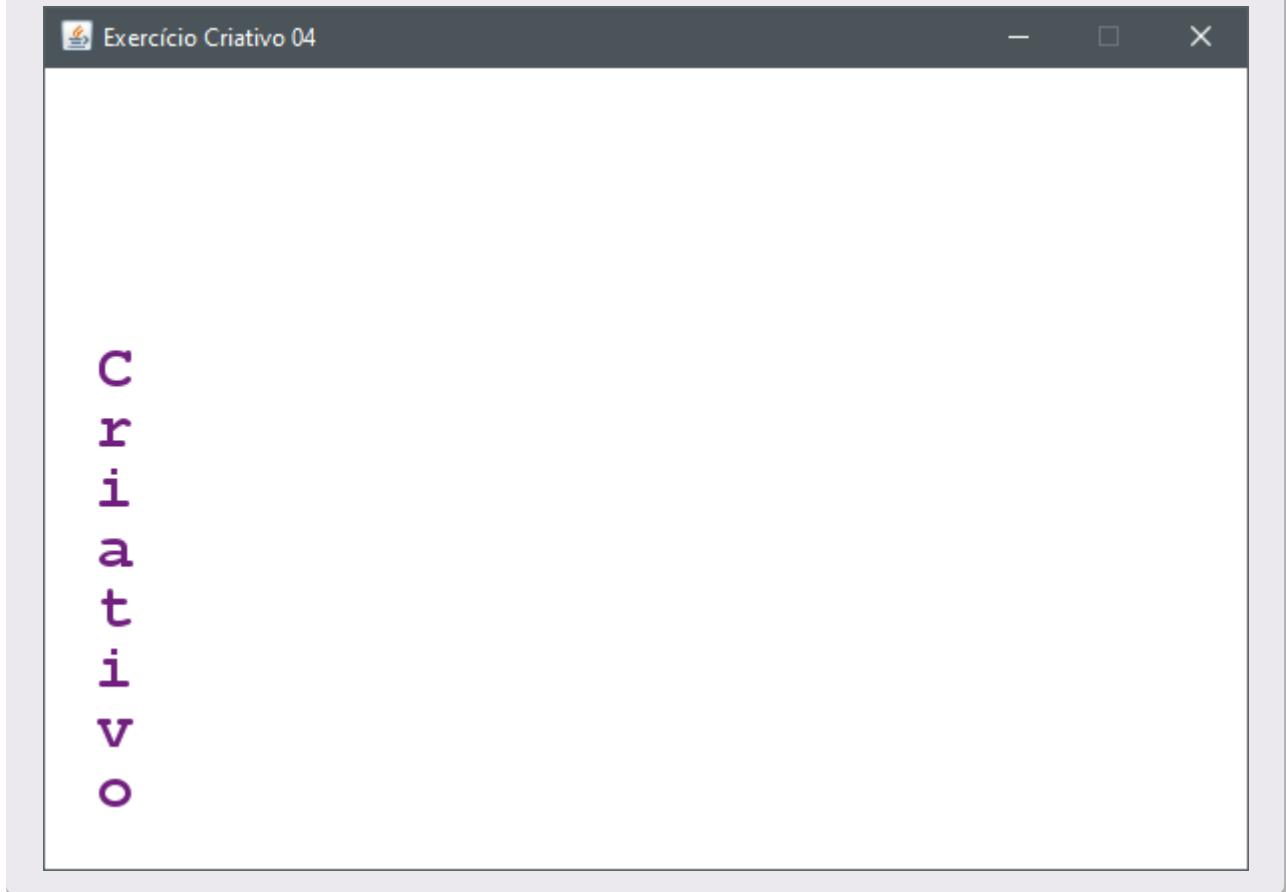
Exercício Criativo 8.3:

Escreva um programa que, dada uma string qualquer, à desenhe verticalmente, alinhada ao topo da janela. Você está livre para usar as cores e as dimensões que quiser.

Saída:

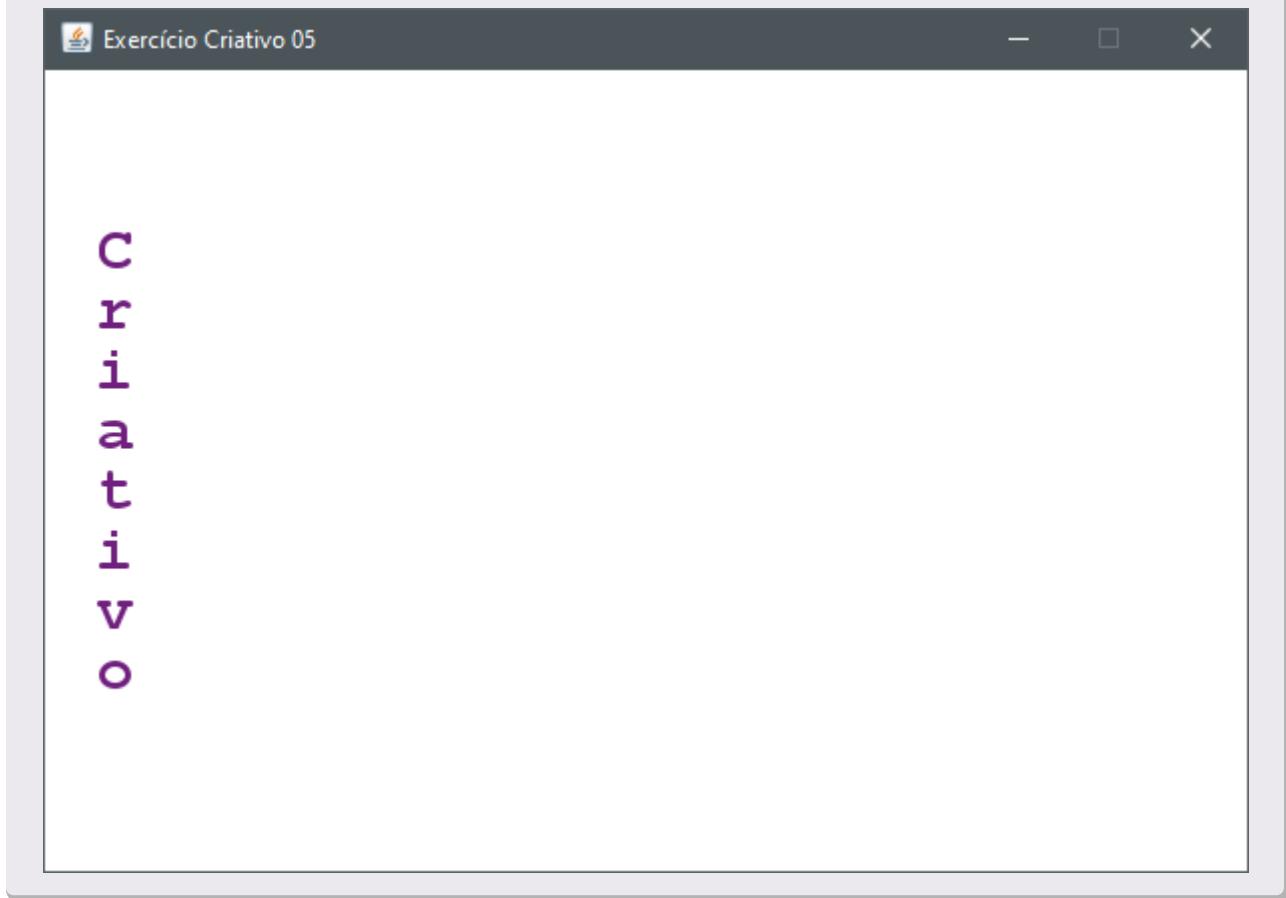
Exercício Criativo 8.4:

Escreva um programa que, dada uma string qualquer, à desenhe verticalmente, alinhada ao “chão” da janela. Você está livre para usar as cores e as dimensões que quiser.

Saída:

Exercício Criativo 8.5:

Escreva um programa que, dada uma string qualquer, à desenhe verticalmente, alinhada ao centro (vertical) da janela. Você está livre para usar as cores e as dimensões que quiser.

Saída:

ARQUIVOS

“A memória dos velhos é menos pronta, porque o seu arquivo é muito extenso”.

Marquês de Maricá



A linguagem Java, assim como na maioria das outras linguagens de programação, quando um programa entra em execução, passando a ser um processo, são ligados à ele pelo menos três fluxos (*streams*) de dados: entrada, saída e erro. Até agora nossos programas lidaram com um fluxo de entrada, associado ao teclado do computador, e um fluxo de saída, associado ao que vemos na tela do terminal. Em alguns casos, os programas que escrevemos precisam lidar com mais de um tipo de *stream* de entrada e/ou saída. Para acessarmos um *stream* na linguagem Java nós utilizamos alguns objetos estáticos contidos na classe `System`. Para que possamos armazenar dados em arquivos, precisamos criar novos fluxos de dados e é isso que veremos brevemente e de forma bastante simples e direta neste Capítulo. Outro detalhe pertinente é que trabalharemos com a API de nível mais baixo, sendo que existem alternativas de nível mais alto como as classes contidas no pacote `java.nio.file`.

Como dito, na linguagem Java existem três *streams* padrão, descritos na Tabela 9.1.

Referência	Stream	Mapeamento padrão
<code>System.in</code>	Entrada padrão	Teclado
<code>System.out</code>	Saída padrão	Tela
<code>System.err</code>	Erro padrão	Tela

Tabela 9.1: Streams padrão

Perceba que são justamente os objetos `System.out` e `System.in` que estamos utilizando até o momento para acessar, respectivamente, os fluxos de saída e de entrada!

9.1 Exemplos em Linguagem Java

Abertura de arquivo, leitura de conteúdo e impressão na tela

```
1  /*
2   * Arquivo: capitulo09/ArquivosAberturaLeitura.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6 import java.io.File;
7 import java.io.FileNotFoundException;
8 import java.util.Scanner;
9
10 public class ArquivosAberturaLeitura {
11
12     public static void main( String[] args ) {
13
14         String dadosLinha;
15         int inteiro;
16         double decimal;
17
18         /*
19          * Uma referência para o arquivo contido no mesmo diretório
20          * em que a JVM está executando.
21          */
22         File arquivo = new File( "arquivoDados.txt" );
23
24         // tentaremos processar o arquivo.
25         try {
26
27             /*
28              * Usaremos um Scanner para processar o arquivo.
29              * o construtor utilizado, ou seja, o que recebe um arquivo
30              * como parâmetro lança uma exceção checada caso o arquivo não
31              * exista. Essa exceção (FileNotFoundException) precisa ser
32              * tratada no nosso caso.
33              */
34             Scanner scan = new Scanner( arquivo );
35
36             // lê uma linha do arquivo três vezes e a imprime
37             for ( int i = 0; i < 3; i++ ) {
38                 dadosLinha = scan.nextLine();
39                 System.out.println( "Linha lida: " + dadosLinha );
40             }
41
42             // lê um inteiro do arquivo
43             inteiro = Integer.parseInt( scan.nextLine() );
44             System.out.printf( "Inteiro lido: %d\n", inteiro );
45         }
```

```
46      // mais uma linha!
47      dadosLinha = scan.nextLine();
48      System.out.println( "Linha lida: " + dadosLinha );
49
50      // lê um double do arquivo
51      decimal = Double.parseDouble( scan.nextLine() );
52      System.out.printf( "Decimal lido: %f\n", decimal );
53
54      // verifica se chegou no fim do arquivo
55      System.out.printf( "Fim do arquivo? %s\n",
56                         scan.hasNextLine() ? "sim" : "nao" );
57
58      // mais uma linha!
59      dadosLinha = scan.nextLine();
60      System.out.println( "Linha lida: " + dadosLinha );
61
62      /*
63       * Nesse ponto, devido aos dados do arquivo, não há
64       * mais dados para serem lidos!
65       */
66      System.out.printf( "Fim do arquivo? %s\n",
67                         scan.hasNextLine() ? "sim" : "nao" );
68
69      // fechando o Scanner que está lendo o arquivo, liberando-o
70      scan.close();
71
72  } catch ( FileNotFoundException exc ) {
73      System.out.printf( "O arquivo nao existe!" );
74  }
75 }
76 }
```

Leitura de um arquivo linha por linha

```
1  /*
2   * Arquivo: capitulo09/ArquivosLeituraTodasLinhas.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6 import java.io.File;
7 import java.io.FileNotFoundException;
8 import java.util.Scanner;
9
10 public class ArquivosLeituraTodasLinhas {
11
12     public static void main( String[] args ) {
13
14         String dadosLinha;
15         File arquivo = new File( "arquivoDados.txt" );
16
17         try {
18
19             Scanner scan = new Scanner( arquivo );
20
21             // lê o arquivo, linha por linha, e imprime na tela
22             while ( scan.hasNextLine() ) {
23                 dadosLinha = scan.nextLine();
24                 System.out.println( dadosLinha );
25             }
26
27             scan.close();
28
29         } catch ( FileNotFoundException exc ) {
30             System.out.printf( "O arquivo nao existe!" );
31         }
32     }
33 }
```

Escrita e leitura de um arquivo

```
1  /*
2   * Arquivo: capitulo09/ArquivosEscrita.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6 import java.io.File;
7 import java.io.FileNotFoundException;
8 import java.io.PrintWriter;
9 import java.util.Scanner;
10
11 public class ArquivosEscrita {
12
13     public static void main( String[] args ) {
14
15         String dados;
16         int inteiro;
17         double decimal;
18         File arquivo = new File( "arquivoEscrita.txt" );
19
20         try {
21
22             Scanner scan = new Scanner( System.in );
23             PrintWriter pw = new PrintWriter( arquivo );
24
25             System.out.print( "Entre um uma frase: " );
26             dados = scan.nextLine();
27
28             System.out.println( "Escrevendo no arquivo..." );
29
30             // escrevendo no arquivo o que foi lido da entrada
31             pw.print( dados );
32
33             // escreve no arquivo um caractere
34             pw.print( '\n' );
35
36             // escreve no arquivo uma string pulando linha
37             pw.println( "mais uma linha para o arquivo!" );
38
39
40             System.out.print( "Entre com um inteiro: " );
41             inteiro = Integer.parseInt( scan.nextLine() );
42
43             System.out.print( "Entre com um decimal: " );
44             decimal = Double.parseDouble( scan.nextLine() );
45
46             // escreve o inteiro e o decimal no arquivo
47             pw.print( inteiro );
```

```
48     pw.print( '\n' );
49     pw.print( decimal );
50
51     pw.close();
52     scan.close();
53
54 } catch ( FileNotFoundException exc ) {
55     System.out.printf( "O arquivo nao existe!" );
56 }
57
58 // imprimindo dados
59 imprimeConteudo( arquivo );
60
61 }
62
63 public static void imprimeConteudo( File arquivo ) {
64
65     try {
66
67         Scanner scan = new Scanner( arquivo );
68
69         while ( scan.hasNextLine() ) {
70             System.out.println( scan.nextLine() );
71         }
72
73         scan.close();
74
75     } catch ( FileNotFoundException exc ) {
76         System.out.printf( "O arquivo nao existe!" );
77     }
78 }
79 }
```

Vamos aos exercícios!

9.2 Exercícios

Os exercícios a seguir lidam com a leitura e escrita de arquivos de texto. Lembre-se de sempre inserir no pacote de código fonte da entrega os arquivos de dados.

Exercício 9.1:

Escreva um programa que leia o arquivo de texto “notas.txt” e que calcule e exiba a média das notas armazenadas nesse arquivo.

Conteúdo do arquivo “notas.txt”

```
6  
8  
6  
9  
10  
9.5  
5  
4  
2  
3.5  
9.75  
8
```

Arquivo com a solução: Exercicio9\$1.java

Saída

```
Media: 6.73
```

Exercício 9.2:

Escreva um programa que leia o arquivo de texto “barras.txt” e exiba uma representação em barras, usando asteriscos, dos dados lidos.

Conteúdo do arquivo “barras.txt”

```
5  
6  
7  
9  
4  
3  
10  
5
```

Arquivo com a solução: Exercicio9\$2.java

Saída

```
***** (5)  
***** (6)  
***** (7)  
***** (9)  
*** (4)  
*** (3)  
***** (10)  
***** (5)
```

Exercício 9.3:

Escreva um programa que:

1. Leia o arquivo de texto chamado “`numeros.txt`”;
2. Para cada um dos cinco números lidos, o programa deve calcular o valor absoluto da diferença de cada um por 10;
3. Os valores da diferença que foram gerados, devem ser usados para desenhar triângulos de asteriscos;
4. Os dados desses triângulos devem ser armazenados em cinco arquivos diferentes (`tri1.txt`, `tri2.txt`, `tri3.txt`, `tri4.txt` e `tri5.txt`);
5. Por fim, o programa deve ler cada um desses arquivos e exibir os dados na tela.

Conteúdo do arquivo “`numeros.txt`”

```
7  
6  
5  
8  
4
```

Arquivo com a solução: `Exercicio9$3.java`

Saída

```
Numero: 7
Absoluto da diferenca: 3
Gerando arquivo 'tri1.txt'...

Numero: 6
Absoluto da diferenca: 4
Gerando arquivo 'tri2.txt'...

Numero: 5
Absoluto da diferenca: 5
Gerando arquivo 'tri3.txt'...

Numero: 8
Absoluto da diferenca: 2
Gerando arquivo 'tri4.txt'...

Numero: 4
Absoluto da diferenca: 6
Gerando arquivo 'tri5.txt'...

Conteudo do arquivo 'tri1.txt':
*
**
***
Conteudo do arquivo 'tri2.txt':
*
**
***
****
Conteudo do arquivo 'tri3.txt':
*
**
***
*****
Conteudo do arquivo 'tri4.txt':
*
**
Conteudo do arquivo 'tri5.txt':
*
**
***
****
*****
*****
```

RECURSIVIDADE

“Para entender recursão, é preciso entender recursão”.



recursividade é uma ferramenta essencial para a solução de diversos tipos de problemas computacionais. Algo que é definido em termos de si mesmo é chamado de recursivo. A seguir diversos exemplos serão apresentados.

Fatorial

Notação 1

$$n! = \begin{cases} 1, & \text{se } n \leq 1 \\ n(n-1)!, & \text{caso contrário} \end{cases} \quad n \in \mathbb{N}$$

Notação 2

$$fat(n) = \begin{cases} 1, & \text{se } n \leq 1 \\ n * fat(n-1), & \text{caso contrário} \end{cases} \quad n \in \mathbb{N}$$

Exemplo em Linguagem Java

```
1 public static int fat( int n ) {  
2     if ( n <= 1 ) {  
3         return 1;  
4     } else {  
5         return n * fat( n - 1 );  
6     }  
7 }
```

```

6     }
7 }
```

Somatório

$$sum(n) = \begin{cases} 0, & \text{se } n = 0 \\ n + sum(n - 1), & \text{caso contrário} \end{cases} \quad n \in \mathbb{N}$$

Exemplo em Linguagem Java

```

1 public static int sum( int n ) {
2     if ( n == 0 ) {
3         return 0;
4     } else {
5         return n + sum( n - 1 );
6     }
7 }
```

Fibonacci

$$fib(n) = \begin{cases} 1, & \text{se } n \leq 1 \\ fib(n - 2) + fib(n - 1), & \text{caso contrário} \end{cases} \quad n \in \mathbb{N}$$

Exemplo em Linguagem Java

```

1 public static int fib( int n ) {
2     if ( n <= 1 ) {
3         return 1;
4     } else {
5         return fib( n - 2 ) + fib( n - 1 );
6     }
7 }
```

Adição

Restrição: somar apenas de uma em uma unidade.

Notação 1

$$a + b = \begin{cases} a, & \text{se } b = 0 \\ a + (b - 1) + 1, & \text{caso contrário} \end{cases} \quad a, b \in \mathbb{N}$$

Notação 2

$$add(a, b) = \begin{cases} a, & \text{se } b = 0 \\ add(a, b - 1) + 1, & \text{caso contrário} \end{cases} \quad a, b \in \mathbb{N}$$

Exemplo em Linguagem Java

```

1 public static int add( int a, int b ) {
2     if ( b == 0 ) {
3         return a;
4     } else {
5         return add( a, b - 1 ) + 1;
6     }
7 }
```

Subtração

Restrição: Subtrair apenas de uma em uma unidade.

Notação 1

$$a - b = \begin{cases} a, & \text{se } b = 0 \\ a - (b - 1) - 1, & \text{caso contrário} \end{cases} \quad a, b \in \mathbb{N}$$

Notação 2

$$sub(a, b) = \begin{cases} a, & \text{se } b = 0 \\ sub(a, b - 1) - 1, & \text{caso contrário} \end{cases} \quad a, b \in \mathbb{N}$$

Exemplo em Linguagem Java

```

1 public static int sub( int a, int b ) {
2     if ( b == 0 ) {
3         return a;
4     } else {
5         return sub( a, b - 1 ) - 1;
6     }
7 }
```

Multiplicação

Restrição: Somando qualquer quantidade.

Notação 1

$$a \cdot b = \begin{cases} 0, & \text{se } a = 0 \text{ ou } b = 0 \\ (a - 1) \cdot b + b, & \text{caso contrário} \end{cases} \quad a, b \in \mathbb{N}$$

Notação 2

$$\text{mult}(a, b) = \begin{cases} 0, & \text{se } a = 0 \text{ ou } b = 0 \\ \text{mult}(a - 1, b) + b, & \text{caso contrário} \end{cases} \quad a, b \in \mathbb{N}$$

Exemplo em Linguagem Java

```

1 public static int mult( int a, int b ) {
2     if ( a == 0 || b == 0 ) {
3         return 0;
4     } else {
5         return mult( a - 1, b ) + b;
6     }
7 }
```

Divisão

Restrição: Subtraindo qualquer quantidade e somando apenas de uma em uma unidade.

Notação 1

$$\frac{a}{b} = \begin{cases} 0, & \text{se } a < b \\ \frac{a-b}{b} + 1, & \text{caso contrário} \end{cases} \quad a, b \in \mathbb{N}$$

Notação 2

$$\text{div}(a, b) = \begin{cases} 0, & \text{se } a < b \\ \text{div}(a - b, b) + 1, & \text{caso contrário} \end{cases} \quad a, b \in \mathbb{N}$$

Exemplo em Linguagem Java

```

1 public static int div( int a, int b ) {
2     if ( a < b ) {
3         return 0;
4     } else {
5         return div( a - b, b ) + 1;
6     }
7 }
```

Resto

Restrição: Subtraindo qualquer quantidade.

Notação 1

$$a \% b = \begin{cases} a, & \text{se } a < b \\ (a - b) \% b, & \text{caso contrário} \end{cases} \quad a, b \in \mathbb{N}$$

Notação 2

$$\text{mod}(a, b) = \begin{cases} a, & \text{se } a < b \\ \text{mod}(a - b, b), & \text{caso contrário} \end{cases} \quad a, b \in \mathbb{N}$$

Exemplo em Linguagem Java

```

1 public static int mod( int a, int b ) {
2     if ( a < b ) {
3         return a;
4     } else {
5         return mod( a - b, b );
6     }
7 }
```

Exponenciação

Restrição: Multiplicando qualquer quantidade.

Notação 1

$$x^n = \begin{cases} 1, & \text{se } n = 0 \\ x \cdot x^{n-1}, & \text{caso contrário} \end{cases} \quad x, n \in \mathbb{N}$$

Notação 2

$$\text{pow}(x, n) = \begin{cases} 1, & \text{se } n = 0 \\ x * \text{pow}(x, n - 1), & \text{caso contrário} \end{cases} \quad x, n \in \mathbb{N}$$

Exemplo em Linguagem Java

```

1 public static int pow( int x, int n ) {
2     if ( n == 0 ) {
3         return 1;
4     } else {
```

```

5         return x * pow( x, n - 1 );
6     }
7 }
```

Usando *squaring*

$$2^4 = 2^{(4/2)2} = (2^{4/2})^2 = (2^2)^2 = 4^2 = 16$$

$$2^5 = 2^{1+(4/2)2} = 2(2^{4/2})^2 = 2(2^2)^2 = 2(4^2) = 32$$

$$2^6 = 2^{(6/2)2} = (2^{6/2})^2 = (2^3)^2 = 8^2 = 64$$

$$2^7 = 2^{1+(6/2)2} = 2(2^{6/2})^2 = 2(2^3)^2 = 2(8^2) = 128$$

$$pows(x, n) = \begin{cases} 1, & \text{se } n = 0 \\ pows(x, n/2)^2, & \text{se } n > 0 \text{ é par} \\ x * pows(x, (n-1)/2)^2, & \text{se } n > 0 \text{ é ímpar} \end{cases} \quad x, n \in \mathbb{N}$$

Exemplo em Linguagem Java

```

1 public static int pows( int x, int n ) {
2     if ( n == 0 ) {
3         return 1;
4     } else if ( n % 2 == 0 ) {    // par
5         int y = pows( x, n / 2 );
6         return y * y;
7     } else {                      // ímpar
8         int y = pows( x, ( n - 1 ) / 2 );
9         return x * y * y;
10    }
11 }
```

10.1 Exercícios

Exercício 10.1:

Escreva um programa que leia dois inteiros e que calcule o resultado da função `ackermann` implementada através de um método estático público de mesmo nome. Sua assinatura é:

- `public static int ackermann(int m, int n)`

Essa função é definida como:

$$\text{ackermann}(m, n) = \begin{cases} n + 1, & \text{se } m = 0 \\ \text{ackermann}(m - 1, 1), & \text{se } m > 0 \text{ e } n = 0 \\ \text{ackermann}(m - 1, \text{ackermann}(m, n - 1)), & \text{se } m > 0 \text{ e } n > 0 \end{cases}$$

| Saiba Mais

Quer saber mais sobre a função Ackermann? Siga o link <<http://dan-scientia.blogspot.com/2009/08/funcao-ackermann.html>>.

Arquivo com a solução: Exercicio10\$1.java

Entrada

```
Entre com o valor de m: 2
Entre com o valor de n: 1
```

Saída

```
ackermann( 2, 1 ) = 5
```

Entrada

```
Entre com o valor de m: 0
Entre com o valor de n: 1
```

Saída

```
ackermann( 0, 1 ) = 2
```

Entrada

```
Entre com o valor de m: 2
Entre com o valor de n: 3
```

Saída

```
ackermann( 2, 3 ) = 9
```

Exercício 10.2:

Escreva um programa que leia dois inteiros e que calcule o máximo divisor comum entre dois números. O cálculo deve ser feito utilizando o método estático público `mdc`. Sua assinatura é:

- `public static int mdc(int a, int b)`

Essa função é definida como:

$$mdc(a, b) = \begin{cases} a, & \text{se } b = 0 \\ mdc(b, a \% b), & \text{se } a \geq b \text{ e } b > 0 \end{cases}$$

Arquivo com a solução: Exercicio10\$2.java

Entrada

```
Entre com o valor de a: 5
Entre com o valor de b: 20
```

Saída

```
mdc( 5, 20 ) = 5
```

Entrada

```
Entre com o valor de a: 15
Entre com o valor de b: 225
```

Saída

```
mdc( 15, 225 ) = 15
```

Entrada

```
Entre com o valor de a: 149
Entre com o valor de b: 7
```

Saída

```
mdc( 149, 7 ) = 1
```

Exercício 10.3:

Escreva um programa que leia uma String e que a inverta, retornando o resultado como String. Para realizar a inversão, deve-se usar o método estático público `inverter`, que por sua vez, deve fazer a inversão de forma recursiva. Sua assinatura é:

- `public static String inverter(String origem)`

Arquivo com a solução: Exercicio10\$3.java

Entrada

String: abracadabra

Saída

Invertida: arbadacarba

Parte II

Introdução à Programação Orientada a Objetos em Java

CLASSES, ATRIBUTOS E MÉTODOS

*“Nunca entendi o que significam
esses malditos pontos”.*

Winston Churchill



S classes tem papel primordial em todas as linguagens de programação do paradigma orientado a objetos. Neste Capítulo iremos começar a aprender como definir, implementar e usar novas classes, permitindo que nós possamos criar novos Tipos Abstratos de Dados (TADs), de modo a modelar objetos do mundo real e/ou que sejam necessários para o desenvolvimento do algoritmo desejado.

11.1 Exemplos em Linguagem Java

Definição e implementação da classe Data

```
1  /*
2  * Arquivo: capitulo11/Data.java
3  * Autor: Prof. Dr. David Buzatto
4  *
5  * Em Java, assim como nas linguagens Orientadas a Objetos, a unidade
6  * básica de programação é chamada de Classe.
7  *
8  * Uma classe, fazendo um paralelo com o mundo real, seria como
9  * uma planta de uma casa. A planta contém as informações básicas
10 * de como será uma casa construída usando aquela planta, ou seja,
11 * a medida das paredes, a altura do piso em relação ao terreno, onde
12 * passarão as redes de água, esgoto, elétrica, lógica, gás etc.
13 *
14 * Sendo assim, ao se usar a planta como um modelo, constroi-se uma ou
15 * mais casas. Claro que aqui estamos tratando do assunto de forma um
16 * pouco mais abstrata, pois na realidade precisaríamos de mais de um
17 * projeto para registrar na prefeitura da cidade etc., mas acredito que
18 * você tenha entendido até aqui.
19 *
20 * Voltando às Classes e à nossa analogia, uma classe contém as "instruções"
21 * de como objetos criados ou instanciados a partir dela se comportarão, ou
22 * seja, quais suas operações (ou métodos) e quais serão suas caracte-
23 * rísticas (ou atributos). Existem diversos outros conceitos atrelados ao
24 * que é uma classe nas linguagens de programação Orientadas a Objetos, mas
25 * por enquanto, isso nos basta. Agora vamos partirmos para detalhes inerentes
26 * à linguagem Java.
27 *
28 * Cada arquivo de código fonte precisa conter pelo menos uma classe.
29 * Usualmente definimos uma classe pública por arquivo e essa classe,
30 * por ser pública, precisa, obrigatoriamente, ter o mesmo nome do
31 * arquivo ou vice-versa.
32 *
33 * Toda classe pública poderá ser usada dentro dos arquivos de código
34 * do projeto em que ela foi criada para criarmos variáveis do tipo
35 * daquela classe. Esses tipos que conseguimos criar a partir da definição
36 * de novas classes são chamados de Tipos Abstratos de Dados (TAD).
37 *
38 * Nomeamos nossas classes sempre usando CamelCase e iniciamos o nome
39 * das mesmas sempre usando letra maiúscula. Usualmente, uma classe
40 * representa uma "coisa" do mundo real, sendo assim, seu nome normalmente
41 * será um substantivo.
42 *
43 * No arquivo Data.java temos a classe pública Data que representa
44 * uma data do mundo real. Podemos representar uma data de algumas formas,
45 * mas talvez a mais fácil de entender seja armazenarmos três valores
```

```
46 * inteiros: um para o dia, um para o mês e um para o ano.
47 */
48 public class Data {
49
50 /*
51 * Em Java, normalmente, no início do código da classe, declaramos
52 * seus atributos. Como dito anteriormente, a classe Data tem os
53 * atributos dia, mês e ano. Os atributos, quando declarados SEM usar
54 * a palavra chava static, são chamados de atributos de instância
55 * e fazem parte de um tipo maior de elementos chamados de membros de
56 * instância. Também podemos chamar esses atributos de variáveis
57 * de instância, visto que esses atributos só existirão a partir do
58 * momento que novos objetos dessa classe forem criados, por isso seus
59 * nomes são "de instância" (de objeto).
60 *
61 * Usualmente os atributos são marcados como privados (private), ou
62 * seja, só são acessíveis (ou enxergados) dentro da própria classe.
63 * De forma contrária, quando marcamos membros como públicos (public),
64 * os membros da classe poderão ser acessados por qualquer parte de
65 * qualquer código que use a nossa classe. A ideia de marcar atributos
66 * como privados e criar formas de acessá-los tem a ver com o conceito
67 * de Encapsulamento, que iremos abordar das próximas aulas com mais
68 * detalhes.
69 */
70
71 // declaração do atributo inteiro privado "dia"
72 private int dia;
73 private int mes;
74 private int ano;
75
76 /*
77 * Atributos de instância recebem valores padrão quando objetos das
78 * classes que os contém são criados, da mesma forma que elementos
79 * de arrays são inicializados na criação de novos arrays, ou seja:
80 *
81 * Valores padrão: tipo / valor
82 *                  byte / 0
83 *                  short / 0
84 *                  int / 0
85 *                  long / 0
86 *                  char / '\0'
87 *                  float / 0.0
88 *                  double / 0.0
89 *                  boolean / false
90 *                  referência / null
91 */
92
93 /*
94 * Toda classe conterá pelo menos um construtor. Os construtores
```

```
95 * são responsáveis em preparar os atributos de instância da classe
96 * de modo que estejam em um estado válido, trazendo também, por
97 * consequência o validado do estado do objeto que foi construído.
98 *
99 * Os construtores se assemelham aos métodos, mas não possuem tipo de
100 * retorno. Outro detalhe é que construtores NÃO SÃO MÉTODOS!
101 *
102 * O construtor que não tem parâmetros é chamado de construtor padrão
103 * e, caso não o forneçamos nenhum construtor para a classe, o
104 * compilador da linguagem Java se encarregará de criá-lo para nós.
105 * Se por algum motivo definirmos algum construtor diferente do
106 * padrão e precisarmos do construtor padrão além do outro ou outros
107 * que criamos, o compilador não fornecerá o construtor padrão, sendo
108 * nossa responsabilidade o definir.
109 *
110 * Usualmente construtores são públicos.
111 */
112 public Data() {
113
114     /*
115      * Todo construtor, implicitamente, invocará outros construtores
116      * até que se chegue à classe mãe de todas as classes em Java
117      * chamada de Object. Esse não é um detalhe essencial nesse momento,
118      * mas é importante sabermos disso.
119      *
120      * Para a classe Data, sua superclasse, também implícita, já é a
121      * classe Object. Sendo assim, implicitamente, a primeira instrução
122      * do construtor padrão da classe Data é:
123      */
124     super();
125
126     /*
127      * A linha acima, como já dito, será inserida implicitamente pelo
128      * compilador. Caso você remova a linha acima, você verá que sua
129      * classe continua a ser compilada sem problema.
130      *
131      * Os conceitos de superclasse (classe mãe), subclasse (classe
132      * filha) etc. serão tratados no Capítulo de Herança e Polimorfismo.
133      */
134
135     /*
136      * Agora temos a chance de inicializarmos os atributos da classe
137      * com valores que são válidos de acordo com os requisitos de
138      * implementação da mesma classe. Caso não o façamos, tanto dia,
139      * quanto mes, quanto ano receberão o valor 0. Podemos dizer que
140      * uma data válida é o dia primeiro do mês de janeiro do ano de
141      * 1970.
142      */
143     dia = 1;
```

```
144     mes = 1;
145     ano = 1970;
146
147 }
148
149 /*
150 * Como dito, podemos ter mais de um construtor. Agora definiremos um
151 * construtor com três parâmetros que permitirá que, quando criarmos
152 * objetos da classe Data, possamos fornecer valores iniciais para seus
153 * três atributos.
154 */
155 public Data( int dia, int mes, int ano ) {
156
157 /*
158 * Note que agora temos no construtor três parâmetros com os mesmos
159 * nomes dos atributos. Em Java isso é permitido, pois há como
160 * resolvemos esse conflito de nomes, chamado tecnicamente de
161 * "shadowing" (sombreamento), pois ainda conseguimos diferenciar
162 * atributos de instância de parâmetros de construtores e/ou
163 * métodos.
164 *
165 * Fazemos isso precedendo os atributos de instância com a palavra-
166 * chave "this" (esse), dizendo explicitamente que queremos lidar
167 * com determinado atributo desse objeto.
168 */
169 this.dia = dia;
170
171 /*
172 * Na linha acima, é atribuído do valor do parâmetro dia ao atributo
173 * dia do objeto (this.dia). Obviamente, poderíamos diferenciar o
174 * nome do parâmetro para talvez "d", então o código poderia ter
175 * ficado assim:
176 *
177 * this.dia = d;
178 *
179 * ou simplesmente
180 *
181 * dia = d;
182 *
183 * Note que como não haveria mais a colisão de nomes entre atributo
184 * e parâmetro, não haveria mais a necessidade de diferenciar quem
185 * é da instância de quem não é. Note que isso foi feito no cons-
186 * trutor padrão, visto que lá não houve conflito. Alguns progra-
187 * madores optam por sempre preceder atributos de instância com
188 * a palavra-chave this, mesmo em lugares onde não há necessidade.
189 *
190 * Vamos agora inicializar cada um dos atributos faltantes.
191 */
192 this.mes = mes;
```

```
193     this.ano = ano;
194
195     /*
196      * Nesse ponto, os atributos do objeto criado terá recebido os
197      * valores fornecidos nos parâmetros do construtor. Note ainda que
198      * nada impede que quem usar a classe Data forneça valores que não
199      * fazem sentido, como por exemplo, valores negativos ou valores
200      * fora do intervalo de valores aceitáveis para os dias (1 a 31) e
201      * meses (1 a 12). Ainda, os valores dos dias dependem do mês em
202      * questão, pois podemos ter meses de 28, 29, 30 e 31 dias. Isso é
203      * assunto para depois, por enquanto vamos assumir que vivemos num
204      * mundo ideal e ninguém usaria valores errados para inicializar o
205      * objeto.
206     */
207
208 }
209
210 /*
211  * Por fim, vamos fornecer à classe Data um método chamado toString.
212  * Esse método, herdado da classe Object, é utilizado para criarmos
213  * uma representação textual do objeto. No nosso caso, faremos ele
214  * retornar os dados da data formatados da forma que usamos no Brasil,
215  * ou seja, o dia, uma barra, o mês, outra barra e o ano.
216  *
217  * Note que foi usada a anotação @Override nesse método, mas isso também
218  * é assunto para depois. Por enquanto vamos assumir que precisamos
219  * usá-la.
220  */
221 @Override
222 public String toString() {
223     return String.format( "%02d/%02d/%04d", dia, mes, ano );
224 }
225
226 }
```

Teste da classe Data

```
1  /*
2   * Arquivo: capitulo11/TesteData.java
3   * Autor: Prof. Dr. David Buzatto
4   *
5   * Agora nós vamos testar a classe Data. Tendo as duas classes no mesmo
6   * diretório, ao compilarmos esta classe, a classe Data também será
7   * compilada automaticamente. Não estamos preocupados, por enquanto,
8   * em como as classes podem ser organizadas em pacotes. Basta que você as
9   * defina no mesmo diretório que, para o que precisamos no momento, bastará.
10  */
11
12 public class TesteData {
13
14     public static void main( String[] args ) {
15
16         /*
17          * Declaração de uma variável local do tipo Data chamada d1
18          * inicializando-a com um novo objeto do tipo Data, instanciado
19          * usando o operador new e o construtor padrão.
20          */
21         Data d1 = new Data();
22
23         /*
24          * Declaração de uma variável local do tipo Data chamada d2
25          * inicializando-a com um novo objeto do tipo Data, instanciado
26          * usando o operador new e o construtor padrão que recebe como
27          * parâmetro o dia, o mês e o ano.
28          */
29         Data d2 = new Data( 25, 2, 1985 );
30
31         /*
32          * Declaração de uma variável local do tipo Data chamada d3
33          * inicializando-a com um novo objeto do tipo Data, instanciado
34          * usando o operador new e o construtor padrão que recebe como
35          * parâmetro o dia, o mês e o ano, mas usando valores incorretos.
36          */
37         Data d3 = new Data( 40, 14, 2022 );
38
39         /*
40          * Usando o método toString para obter uma representação textual
41          * dos objetos criados.
42          *
43          * Note que, ao concatenarmos uma String com um objeto, o método
44          * toString será invocado implicitamente, não havendo a necessidade
45          * de chamá-lo explicitamente.
46          */
47         System.out.println( "d1: " + d1.toString() );
```

```
48     System.out.println( "d2: " + d2 );
49     System.out.println( "d3: " + d3 );
50
51     /*
52      * Para terminar esse exemplo, vamos pensar no seguinte: e se
53      * quiséssemos acessar cada atributo da classe Data separadamente?
54      * Ou seja, queremos usar o dia para realizar algum cálculo, ou
55      * simplesmente para exibi-lo. Como fariamos?
56      *
57      * No estado que nossa implementação está ainda não conseguimos
58      * fazer isso, o que acaba tornando nossa Data algo quase inútil,
59      * mas iremos começar a sanar isso logo logo.
60      *
61      * Por enquanto estamos preocupados em criar novas classes com
62      * seus atributos e inicializar tais valores. Uma coisa de cada vez.
63      */
64
65 }
66
67 }
```

Definição e implementação da classe Pessoa

```
1  /*
2   * Arquivo: capitulo11/Pessoa.java
3   * Autor: Prof. Dr. David Buzatto
4   *
5   * Classe que representa uma Pessoa.
6   */
7  public class Pessoa {
8
9      private String nome;
10     private String sobrenome;
11     private double peso;
12
13     public Pessoa() {
14         nome = "";
15         sobrenome = "";
16         peso = 50;
17     }
18
19     public Pessoa( String nome, String sobrenome, double peso ) {
20         this.nome = nome;
21         this.sobrenome = sobrenome;
22         this.peso = peso;
23     }
24
25     @Override
26     public String toString() {
27
28         String n = nome;
29         String s = sobrenome;
30
31         if ( nome == null || nome.isEmpty() ) {
32             n = "sem nome";
33         }
34
35         if ( sobrenome == null || sobrenome.isEmpty() ) {
36             s = "sem sobrenome";
37         }
38
39         return String.format( "%s, %s: (%.2f)", s, n, peso );
40     }
41 }
42
43 }
```

Teste da classe Pessoa

```
1  /*
2   * Arquivo: capitulo11/TestePessoa.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5  public class TestePessoa {
6
7      public static void main( String[] args ) {
8
9          Pessoa p1 = new Pessoa();
10         Pessoa p2 = new Pessoa( "David", "Buzatto", 104.5 );
11
12         System.out.println( p1 );
13         System.out.println( p2 );
14     }
15
16 }
17 }
```

Definição e implementação da classe Conta

```
1  /*
2   * Arquivo: capitulo11/Conta.java
3   * Autor: Prof. Dr. David Buzatto
4   *
5   * Classe que representa uma Conta.
6   */
7  public class Conta {
8
9      private String numero;
10
11     /*
12      * Para lidarmos com valores monetários, que exigem precisão total,
13      * double não é o tipo mais apropriado, mas o usaremos por enquanto
14      * por questão de facilidade.
15     */
16     private double saldo;
17     private double limite;
18
19     public Conta() {
20         numero = "";
21         saldo = 0;
22         limite = 500;
23     }
24
25     public Conta( String numero, double saldo, double limite ) {
26
27         this.numero = numero;
28
29         // atualiza o saldo caso o valor inicial seja maior que zero
30         if ( saldo > 0 ) {
31             this.saldo = saldo;
32         }
33
34         // atualiza o limite caso o valor inicial seja maior que zero
35         if ( limite > 0 ) {
36             this.limite = limite;
37         }
38     }
39 }
40
41 /**
42  * Saca um valor da conta respeitando o limite.
43  *
44  * @param valor Valor a ser sacado.
45  * @return Verdadeiro caso o saque seja bem-sucedido, falso em caso
46  * contrário.
47 */
```

```
48 public boolean sacar( double valor ) {  
49  
50     // só tenta sacar se o valor fornecido for maior que zero  
51     if ( valor > 0 ) {  
52  
53         // verifica se há limite para sacar  
54         if ( saldo + limite - valor >= 0 ) {  
55  
56             saldo -= valor;  
57             return true;  
58  
59             // se não houver, não saca  
60         } else {  
61             return false;  
62         }  
63     }  
64  
65     return false;  
66 }  
67  
68 /**  
69 * Deposita um valor na conta.  
70 *  
71 * @param valor Valor a ser depositado. Valores negativos são ignorados.  
72 */  
73 public void depositar( double valor ) {  
74  
75     // só deposita se o valor fornecido for maior que zero  
76     if ( valor > 0 ) {  
77         saldo += valor;  
78     }  
79 }  
80  
81 }  
82  
83  
84 @Override  
85 public String toString() {  
86  
87     String situacao = saldo < 0 ? "devedora" : "credora";  
88     String dados = String.format( "Conta: %s\n", numero );  
89     dados += String.format( "Saldo: %sR$%.2f\n",  
90         saldo < 0 ? "-" : "",  
91         saldo < 0 ? -saldo : saldo );  
92     dados += String.format( "Limite: R$%.2f\n", limite );  
93     dados += "Situacao: " + situacao;  
94  
95     return dados;  
96 }
```

```
97      }  
98  
99 }
```

Teste da classe Conta

```
1  /*
2   * Arquivo: capitulo11/TesteConta.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5  public class TesteConta {
6
7      public static void main( String[] args ) {
8
9          Conta conta = new Conta( "12345", 2000, 1000 );
10         System.out.println( conta + "\n" );
11
12         System.out.println( "Depositando R$500,00" );
13         conta.depositar( 500 );
14         System.out.println( conta + "\n" );
15
16         System.out.print( "Sacando R$1000,00: " );
17         if ( conta.sacar( 1000 ) ) {
18             System.out.println( "Saque realizado!" );
19         } else {
20             System.out.println( "Limite insuficiente!" );
21         }
22         System.out.println( conta + "\n" );
23
24         System.out.print( "Sacando R$2000,00: " );
25         if ( conta.sacar( 2000 ) ) {
26             System.out.println( "Saque realizado!" );
27         } else {
28             System.out.println( "Limite insuficiente!" );
29         }
30         System.out.println( conta + "\n" );
31
32         System.out.print( "Sacando R$200,00: " );
33         if ( conta.sacar( 200 ) ) {
34             System.out.println( "Saque realizado!" );
35         } else {
36             System.out.println( "Limite insuficiente!" );
37         }
38         System.out.println( conta + "\n" );
39
40         System.out.print( "Sacando R$400,00: " );
41         if ( conta.sacar( 400 ) ) {
42             System.out.println( "Saque realizado!" );
43         } else {
44             System.out.println( "Limite insuficiente!" );
45         }
46         System.out.println( conta + "\n" );
47     }
```

```
48     System.out.print( "Sacando R$300,00: " );
49     if ( conta.sacar( 300 ) ) {
50         System.out.println( "Saque realizado!" );
51     } else {
52         System.out.println( "Limite insuficiente!" );
53     }
54     System.out.println( conta + "\n" );
55
56     System.out.println( "Depositando R$1800,00" );
57     conta.depositar( 1800 );
58     System.out.println( conta );
59
60 }
61
62 }
```

11.2 Representação em UML

A *Unified Modeling Language*¹ (UML) é uma linguagem de modelagem gráfica para o projeto de aplicações que usam linguagens orientadas a objetos como linguagens de implementação. Você aprenderá mais sobre UML no decorrer do seu curso técnico ou de graduação. Por enquanto veremos superficialmente como representar classes e seus relacionamentos nos diagramas de classe dessa linguagem. Nossa objetivo, novamente, não é esgotar o assunto, mas somente mostrar como são representadas as classes e seus relacionamentos na UML. Na Figura 11.1, na Figura 11.2 e na Figura 11.3 são representadas, respectivamente, as classes `Data`, `Pessoa` e `Conta` que vimos a implementação anteriormente.

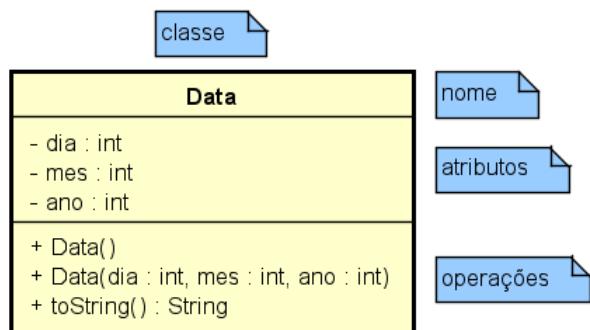


Figura 11.1: A classe Data em UML

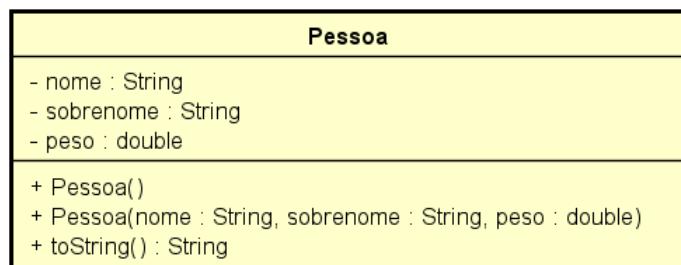


Figura 11.2: A classe Pessoa em UML

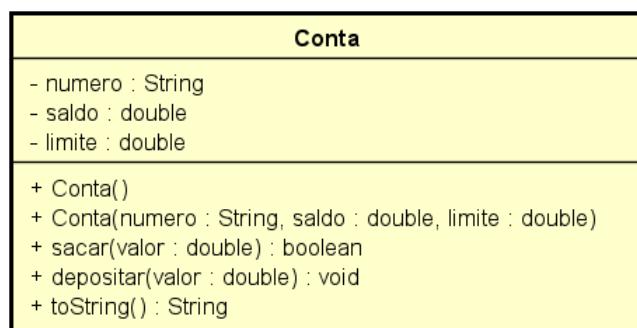


Figura 11.3: A classe Conta em UML

¹Linguagem de Modelagem Unificada, em tradução livre.

11.3 Exercícios

Exercício 11.1:

Crie uma classe chamada `Recibo` que uma loja de ferramentas poderia usar para representar um recibo de um item vendido na loja. Um `Recibo` possui quatro atributos: o identificador do item vendido (tipo `String`), a descrição do item (tipo `String`), a quantidade do item que está sendo comprado (tipo `int`) e o preço de venda de cada item (`double`). Sua classe deve possuir um construtor que inicializa as quatro variáveis de instância. Se a quantidade fornecida for menor ou igual a zero, ela deve ser inicializada com 1 e, se o preço for negativo, ele deve ser inicializado com 0.0. Forneça o método `public double getValorTotal()`, que calcula e retorna o valor total do recibo, ou seja, ele deve retornar a quantidade do item comprado multiplicado pelo preço unitário. Sobrescreva o método `public String toString()` para retornar uma representação textual de cada um dos recibos. Escreva uma segunda classe, chamada `TesteRecibo`, onde a classe `Recibo` será testada de modo a mostrar suas funcionalidades. O formato da `String` retornada pelo método `toString()`, usando os dados do primeiro exemplo, é o seguinte:

| Exemplo

Formato `toString()`

```

1 Recibo:
2   id = p123
3   descricao = parafuso
4   quantidade = 10
5   preco = R$0,50
6   valor total = R$5,00

```

Diretório base: ex11\$1/

Arquivos com a solução:

- `Recibo.java`
- `TesteRecibo.java` `main`

Entrada

```

Id: p123
Descricao: parafuso
Quantidade: 10
Preco unitario: 0.5

```

Saída

```
Recibo:  
    id = p123  
    descricao = parafuso  
    quantidade = 10  
    preco = R$0,50  
    valor total = R$5,00
```

Entrada

```
Id: m456  
Descricao: mangueira  
Quantidade: -4  
Preco unitario: 5.5
```

Saída

```
Recibo:  
    id = m456  
    descricao = mangueira  
    quantidade = 1  
    preco = R$5,50  
    valor total = R$5,50
```

Entrada

```
Id: f389  
Descricao: fio de cobre  
Quantidade: 100  
Preco unitario: -1
```

Saída

```
Recibo:  
    id = f389  
    descricao = fio de cobre  
    quantidade = 100  
    preco = R$0,00  
    valor total = R$0,00
```

Exercício 11.2:

Crie uma classe chamada `Empregado` que possui três atributos: o nome (`String`), o sobrenome (`String`) e seu salário mensal (`double`). Forneça um construtor que initialize cada um dos atributos. Se o salário fornecido não for positivo, ele não deve ser inicializado, deixando o valor padrão. Forneça o método `public void concederAumento()`, que aumentará o salário atual do empregado em 10%. Sobrescreva o método `public String toString()` para retornar uma representação textual de cada um dos empregados. Escreva uma segunda classe, chamada `TesteEmpregado`, onde a classe `Empregado` será testada de modo a mostrar suas funcionalidades. O formato da `String` retornada pelo método `toString()`, usando os dados do primeiro exemplo, é o seguinte:

| Exemplo**Formato `toString()`**

```
1 Empregado:  
2     nome = joao  
3     sobrenome = da silva  
4     salario = R$2000,00
```

Diretório base: ex11\$/

Arquivos com a solução:

- `Empregado.java`
- `TesteEmpregado.java` `main`

Entrada

```
Nome: joao  
Sobrenome: da silva  
Salario: 2000
```

Saída

```
Empregado:  
    nome = joao  
    sobrenome = da silva  
    salario = R$2000,00  
Aumentando o salario...  
Empregado:  
    nome = joao  
    sobrenome = da silva  
    salario = R$2200,00
```

Exercício 11.3:

Um estacionamento cobra uma taxa mínima de R\$2,00 para alguém estacionar por até três horas e, a partir de três horas, mais R\$0,50 por hora extra utilizada. A taxa máxima para um período de 24 horas é de R\$10,00. Assuma que nenhum carro fica estacionado por mais de 24 horas. Crie uma classe chamada `Vaga` que possui dois atributos: a placa do veículo estacionado (`String`) e a quantidade de horas (`int`). Forneça um construtor que inicialize cada um dos atributos. Se a quantidade de horas fornecida for menor ou igual a zero, atribua o valor 1 a ela. Forneça o método `public double getTaxa()`, que será responsável em calcular e retornar o valor da taxa que um determinado usuário do estacionamento deverá pagar com base na quantidade de horas que seu carro ficou estacionado. Sobrescreva o método `public String toString()` para retornar uma representação textual de cada uma das vagas. Escreva uma segunda classe, chamada `TesteVaga`, onde a classe `Vaga` será testada de modo a mostrar suas funcionalidades. O formato da `String` retornada pelo método `toString()`, usando os dados do primeiro exemplo, é o seguinte:

| Exemplo

Formato `toString()`

```

1 Vaga:
2     placa = ERT1234
3     quantidade = 2
4     taxa = R$2,00

```

Diretório base: ex11\$3/

Arquivos com a solução:

- `Vaga.java`
- `TesteVaga.java` `main`

Entrada

```

Placa: ERT1234
Quantidade: 2

```

Saída

```

Vaga:
    placa = ERT1234
    quantidade = 2
    taxa = R$2,00

```

Entrada

Placa: ERT1234

Quantidade: 6

Saída

Vaga:

placa = ERT1234

quantidade = 6

taxa = R\$3,50

Entrada

Placa: ERT1234

Quantidade: 22

Saída

Vaga:

placa = ERT1234

quantidade = 22

taxa = R\$10,00

Exercício 11.4:

Projete um classe chamada `Cor` que representará uma cor no modelo *Red, Green and Blue* (RGB), ou seja, ela deve ter três atributos inteiros: vermelho, verde e azul. Forneça um construtor que inicializa cada um desses atributos. O intervalo de valores permitidos para essa inicialização varia de 0, inclusive a 255, inclusive. Para cada um dos três componentes, caso um valor menor que 0 seja fornecido, o valor padrão deve ser mantido, enquanto caso um valor maior que 255 seja fornecido, o valor 255 deve ser atribuído àquele componente. Sobrescreva o método `public String toString()` para retornar uma representação textual de cada uma das cores. Escreva uma segunda classe, chamada `TesteCorEx4`, onde a classe `Cor` será testada de modo a mostrar suas funcionalidades. O formato da `String` retornada pelo método `toString()`, usando os dados do primeiro exemplo, é o seguinte:

| Exemplo

Formato `toString()`

```
1  rgb( 50, 60, 70 )
```

Diretório base: ex11\$4/

Arquivos com a solução:

- `Cor.java`
- `TesteCorEx4.java` `main`

Entrada

```
Vermelho: 50
Verde: 60
Azul: 70
```

Saída

```
Cor: rgb( 50, 60, 70 )
```

Entrada

```
Vermelho: -10
Verde: -10
Azul: -10
```

Saída

```
Cor: rgb( 0, 0, 0 )
```

Entrada

Vermelho: 260

Verde: 180

Azul: 280

Saída

Cor: `rgb(255, 180, 255)`

Exercício 11.5:

Utilizando a solução do exercício anterior, incremente a implementação da classe `Cor` fornecendo o método `public Cor escurecer()`, que será responsável em criar uma nova cor usando como base a cor que o invocar, sendo essa nova cor mais escura que a original. Esse método deve multiplicar cada componente da cor por 0,7, truncando a parte decimal no resultado para a nova cor. Escreva uma segunda classe, chamada `TesteCorEx5`, onde a classe `Cor` será testada de modo a mostrar suas funcionalidades.

Diretório base: ex11\$5/

Arquivos com a solução:

- `Cor.java`
- `TesteCorEx5.java` **main**

Entrada

Vermelho: 255

Verde: 180

Azul: 90

Saída

Cor base: `rgb(255, 180, 90)`

Cor escurecida: `rgb(178, 125, 62)`

Exercício 11.6:

Utilizando a solução do exercício anterior, incremente a implementação da classe `Cor` fornecendo o método `public Cor clarear()`, que será responsável em criar uma nova cor usando como base a cor que o invocar, sendo essa nova cor mais clara que a original. Esse método deve dividir cada componente por 0,7, truncando a parte decimal no resultado para a nova cor. Entretanto, existem alguns casos especiais que devem ser observados: 1) Se o valor de todos os componentes da cor original forem iguais a zero, a nova cor deve ser gerada com todos os componentes valendo 3; 2) Caso algum componente da cor original for maior que zero, mas menor que 3, deve-se configurá-lo como 3 na nova cor antes da divisão por 0,7; 3) Se a divisão por 0,7 de um componente da cor original resultar em algum valor maior que 255, deve-se configurar, na nova cor, esse componente com o valor 255. Escreva uma segunda classe, chamada `TesteCorEx6`, onde a classe `Cor` será testada de modo a mostrar suas funcionalidades.

Diretório base: ex11\$6/

Arquivos com a solução:

- `Cor.java`
- `TesteCorEx6.java` `main`

Entrada

Vermelho: 10

Verde: 30

Azul: 40

Saída

Cor base: `rgb(10, 30, 40)`

Cor clareada: `rgb(14, 42, 57)`

Entrada

Vermelho: 0

Verde: 0

Azul: 0

Saída

Cor base: `rgb(0, 0, 0)`

Cor clareada: `rgb(3, 3, 3)`

Entrada

```
Vermelho: 1  
Verde: 2  
Azul: 1
```

Saída

```
Cor base: rgb( 1, 2, 1 )  
Cor clareada: rgb( 4, 4, 4 )
```

Entrada

```
Vermelho: 100  
Verde: 150  
Azul: 230
```

Saída

```
Cor base: rgb( 100, 150, 230 )  
Cor clareada: rgb( 142, 214, 255 )
```

Exercício 11.7:

Projete uma classe chamada `Ponto` que possui os atributos x e y, ambos decimais. Forneça um construtor que inicialize cada um desses atributos. Forneça também o método `getDistancia`, apresentado abaixo:

- `public double getDistancia(Ponto outroPonto)`

Esse método deve receber um ponto e retornar a distância entre o ponto em que o método foi invocado até o ponto que foi passado como parâmetro. Sobrescreva o método `toString()` para retornar uma representação textual de cada um dos pontos. Escreva uma segunda classe, chamada `TestePonto`, onde a classe `Ponto` será testada de modo a mostrar suas funcionalidades. O formato da `String` retornada pelo método `toString()`, usando os dados do primeiro exemplo, é o seguinte:

| Exemplo**Formato `toString()`**

```
1 [x=0,00;y=0,00]
```

Diretório base: ex11\$7/

Arquivos com a solução:

- `Ponto.java`
- `TestePonto.java` `main`

Entrada

```
Ponto 1:  
  x: 0  
  y: 0  
Ponto 2:  
  x: 10  
  y: 10
```

Saída

```
Ponto 1: [x=0,00;y=0,00]  
Ponto 2: [x=10,00;y=10,00]  
Distancia entre os pontos: 14,14
```

Exercício 11.8:

Utilizando a classe `Data` do exemplo deste Capítulo, incremente sua implementação criando um método chamado `public int getDiaDoAno()` que deve retornar o dia do ano correspondente à data chamadora. Escreva uma segunda classe, chamada `TesteData`, onde a classe `Data` será testada de modo a mostrar suas funcionalidades.

Diretório base: ex11\$8/

Arquivos com a solução:

- `Data.java`
- `TesteData.java` `main`

Entrada

```
Dia: 30  
Mes: 3  
Ano: 1999
```

Saída

```
Data: 30/03/1999  
Dia do ano: 89
```

Entrada

```
Dia: 30  
Mes: 3  
Ano: 2000
```

Saída

```
Data: 30/03/2000  
Dia do ano: 90
```

Exercício 11.9:

Projete um classe chamada `Hora` que representará um horário contendo três atributos inteiros: a quantidade de horas, a quantidade de minutos e a quantidade de segundos. Forneça um construtor que inicializa cada um desses atributos, não permitindo que nenhum seja negativo e que os minutos e os segundos não ultrapassem 59, sendo que, se ultrapassarem o valor 59, eles devem ser inicializados com 59 e caso sejam negativos devem ser inicializados com o valor padrão. Sobrescreva o método `public String toString()` para retornar uma representação textual de cada um dos horários. Escreva uma segunda classe, chamada `TesteHoraEx9`, onde a classe `Hora` será testada de modo a mostrar suas funcionalidades. O formato da `String` retornada pelo método `toString()`, usando os dados do primeiro exemplo, é o seguinte:

| Exemplo

Formato `toString()`

1 10:20:30

Diretório base: ex11\$9/**Arquivos com a solução:**

- `Hora.java`
- `TesteHoraEx9.java` `main`

Entrada

```
Horas: 10
Minutos: 20
Segundos: 30
```

Saída

```
Hora: 10:20:30
```

Exercício 11.10:

Utilizando a solução do exercício anterior, forneça para a classe `Hora` mais um construtor que recebe uma quantidade de segundos como parâmetro e deve utilizá-lo para calcular a quantidade de horas, minutos e segundos correspondentes. Uma quantidade de segundos negativa representa um horário com todos os atributos valendo zero. Escreva uma segunda classe, chamada `TesteHoraEx10`, onde a classe `Hora` será testada de modo a mostrar suas funcionalidades.

Diretório base: ex11\$10/

Arquivos com a solução:

- `Hora.java`
- `TesteHoraEx10.java` `main`

Entrada

Quantidade segundos: 51234

Saída

Hora: 14:13:54

Exercício 11.11:

Crie uma classe chamada `Fracao` que representará uma fração, possuindo dois atributos decimais: o numerador e o denominador. Sobrescreva o método `public String toString()` para retornar uma representação textual de cada uma das frações. Forneça também quatro métodos responsáveis em criar novas frações como resultado das operações de adição, subtração, multiplicação e divisão de frações, onde a fração chamadora utilizará uma segunda fração, passada como parâmetro, e gerará uma nova fração como resultado da operação correspondente. A soma e a subtração de frações envolve a obtenção do mínimo múltiplo comum, entretanto, para simplificar a implementação, caso os denominadores sejam diferentes, calcule o denominador comum como a multiplicação dos dois denominadores. Além disso, as frações não precisam ser simplificadas. Como já informado, essas quatro operações devem ser implementadas através dos métodos apresentados abaixo:

- `public Fracao somar(Fracao outraFracao)`
- `public Fracao subtrair(Fracao outraFracao)`
- `public Fracao multiplicar(Fracao outraFracao)`
- `public Fracao dividir(Fracao outraFracao)`

Escreva uma segunda classe, chamada `TesteFracao`, onde a classe `Fracao` será testada de modo a mostrar suas funcionalidades. Caso não se lembre como essas operações funcionam, faça uma rápida pesquisa. Você encontrará facilmente diversos materiais sobre o assunto. O formato da `String` retornada pelo método `toString()`, usando os dados do primeiro exemplo, é o seguinte:

| Exemplo**Formato `toString()`**

```
1 1,00/2,00
```

Diretório base: ex11\$11/

Arquivos com a solução:

- `Fracao.java`
- `TesteFracao.java` **main**

Entrada

```
Fracao 1:  
    numerador: 1  
    denominador: 2  
Fracao 2:  
    numerador: 3  
    denominador: 4
```

Saída

```
1,00/2,00 + 3,00/4,00 = 10,00/8,00  
1,00/2,00 - 3,00/4,00 = -2,00/8,00  
1,00/2,00 * 3,00/4,00 = 3,00/8,00  
1,00/2,00 / 3,00/4,00 = 4,00/6,00
```

Exercício 11.12:

Crie uma classe chamada `Complexo` que representará um número do conjunto dos números complexos, possuindo dois atributos decimais: a parte real e a parte imaginária. Sobrescreva o método `public String toString()` para retornar uma representação textual de cada um dos números complexos. Forneça também três métodos responsáveis em criar novas números complexos como resultado das operações de adição, subtração e multiplicação de números complexos, onde o número complexo chamador utilizará um segundo número complexo, passado como parâmetro, e gerará um novo número complexo como resultado da operação correspondente. Essas três operações devem ser implementadas através dos métodos apresentados abaixo:

- `public Complexo somar(Complexo outroComplexo)`
- `public Complexo subtrair(Complexo outraFracaoComplexo)`
- `public Complexo multiplicar(Complexo outraFracaoComplexo)`

Escreva uma segunda classe, chamada `TesteComplexo`, onde a classe `Complexo` será testada de modo a mostrar suas funcionalidades. Caso não se lembre como essas operações funcionam nesse conjunto numérico, faça uma rápida pesquisa. Você encontrará facilmente diversos materiais sobre o assunto. O formato da `String` retornada pelo método `toString()`, usando os dados do primeiro exemplo, é o seguinte:

| Exemplo

Formato `toString()`

```
1 (1,00 + 2,00i)
```

Diretório base: ex11\$12/

Arquivos com a solução:

- `Complexo.java`
- `TesteComplexo.java` `main`

Entrada

```
Complexo 1:  
    real: 1  
    imaginario: 2  
Complexo 2:  
    real: 3  
    imaginario: 4
```

Saída

```
(1,00 + 2,00i) + (3,00 + 4,00i) = (4,00 + 6,00i)  
(1,00 + 2,00i) - (3,00 + 4,00i) = (-2,00 + -2,00i)  
(1,00 + 2,00i) * (3,00 + 4,00i) = (-5,00 + 10,00i)
```

11.4 Exercícios Criativos

Praticamente todos os métodos para desenho possuem versões sobrecarregadas que têm parâmetros e/ou retornam valores de tipos de classes que ainda não usamos. Os métodos que aprendemos no primeiro Capítulo do livro têm variações que, ao invés de receber coordenadas x e y separadas, as recebem como um objeto do tipo `Vector2` que possui dois atributos, um `double x` e um `double y`. Veremos vários outros métodos nos próximos exercícios. Ainda, antes de partirmos para eles, vamos ver dois exemplos de como podemos manipular sinais vindos do mouse e do teclado usando a abstração implementada, assim cobriremos toda a parte básica de como utilizá-la e estaremos aptos a implementar alguns jogos simples na seção de projetos deste Capítulo!

11.5 Exemplos em Linguagem Java

Exemplos de tratamento de eventos relacionados ao controle do mouse

```

1 import br.com.davidbuzatto.jsge.core.engine.EngineFrame;
2 import br.com.davidbuzatto.jsge.math.Vector2;
3
4 /**
5  * Exemplo de tratamento de eventos do mouse.
6  *
7  * @author Prof. Dr. David Buzatto
8  * @copyright Copyright (c) 2024
9  */
10 public class Main extends EngineFrame {
11
12     private class Linha {
13         Vector2 ini = new Vector2();
14         Vector2 fim = new Vector2();
15         boolean desenhar;
16     }
17
18     private Linha linhaTemporaria;
19     private Linha[] linhas;
20     private int quantidadeLinhas;
21
22     public Main() {
23         super( 600, 400, "Exemplo de Tratamento do Mouse", 60, true );
24         linhas = new Linha[100];
25         quantidadeLinhas = 0;
26     }
27
28     @Override
29     public void create() {
30     }
31
32     @Override
33     public void update( double delta ) {

```

```
34
35 // verifica se o botão da esquerda do mouse foi pressionado uma vez
36 if ( isMouseButtonPressed( MOUSE_BUTTON_LEFT ) ) {
37
38     System.out.print( "botao da esquerda pressionado: " );
39
40     linhaTemporaria = new Linha();
41
42     /* obtém a posição em x do mouse
43      * int getX();
44      */
45
46     /* obtém a posição em y do mouse
47      * int getY();
48      */
49     System.out.printf( "(%d, %d)\n", getX(), getY() );
50
51     linhaTemporaria.ini.x = getX();
52     linhaTemporaria.ini.y = getY();
53
54     // verifica se o botão da esquerda do mouse foi solto
55 } else if ( isMouseButtonReleased( MOUSE_BUTTON_LEFT ) ) {
56
57     System.out.println( "botao da esquerda solto!" );
58     linhas[quantidadeLinhas++] = linhaTemporaria;
59
60     // verifica se o botão da esquerda do mouse continua pressionado
61 } else if ( isMouseButtonDown( MOUSE_BUTTON_LEFT ) ) {
62
63     if ( linhaTemporaria != null ) {
64
65         System.out.println( "botao continua pressionado!" );
66
67         linhaTemporaria.fim.x = getX();
68         linhaTemporaria.fim.y = getY();
69         linhaTemporaria.desenhar = true;
70
71     }
72
73 }
74
75 }
76
77 @Override
78 public void draw() {
79
80     if ( linhaTemporaria != null && linhaTemporaria.desenhar ) {
81         desenharLinha( linhaTemporaria );
82     }
}
```

```
83
84     for ( int i = 0; i < quantidadeLinhas; i++ ) {
85         desenharLinha( linhas[i] );
86     }
87 }
88
89     void desenharLinha( Linha linha ) {
90         drawLine( linha.ini.x, linha.ini.y,
91                   linha.fim.x, linha.fim.y, DARKPURPLE );
92     }
93
94     public static void main( String[] args ) {
95         new Main();
96     }
97 }
98 }
99 }
```

Exemplos de tratamento de eventos relacionados ao controle do teclado

```
1 import br.com.davidbuzatto.jsge.core.engine.EngineFrame;
2
3 /**
4 * Exemplo de tratamento de eventos do teclado.
5 *
6 * @author Prof. Dr. David Buzatto
7 * @copyright Copyright (c) 2024
8 */
9 public class Main extends EngineFrame {
10
11     private char[] caracteres;
12     private int quantidadeCaracteres;
13
14     public Main() {
15         super( 600, 400, "Exemplo de Tratamento do Teclado", 60, true );
16         caracteres = new char[100];
17         quantidadeCaracteres = 0;
18     }
19
20     @Override
21     public void create() {
22     }
23
24     @Override
25     public void update( double delta ) {
26
27         if ( isKeyPressed( KEY_SPACE ) ) {
28             System.out.println( "tecla <espaco> pressionada!" );
29         }
30
31         if ( isKeyReleased( KEY_SPACE ) ) {
32             System.out.println( "tecla <espaco> solta!" );
33         }
34
35         char c = getCharPressed();
36         if ( c != KEY_NULL ) {
37             if ( quantidadeCaracteres < 100 ) {
38                 caracteres[quantidadeCaracteres++] = c;
39             }
40         }
41
42         int k = getKeyPressed();
43         if ( k != KEY_NULL ) {
44             System.out.printf( "codigo tecla pressionada: %d\n", k );
45         }
46     }
47 }
```

```
48
49     @Override
50     public void draw() {
51
52         int x;
53         int y = 10;
54         int k = 0;
55
56         int tamanhoFonte = 20;
57
58         for ( int i = 0; i < quantidadeCaracteres; i++ ) {
59             char caractere = caracteres[i];
60             if ( caractere == '\n' ) {
61                 y += tamanhoFonte;
62                 k = 0;
63             } else {
64
65                 x = 10 + k * tamanhoFonte;
66                 k++;
67
68                 drawText( String.format( "%c", caracteres[i] ),
69                           x, y + 10,
70                           tamanhoFonte, DARKPURPLE );
71             }
72         }
73     }
74
75 }
76
77 public static void main( String[] args ) {
78     new Main();
79 }
80
81 }
```

Vamos aos exercícios!

Exercício Criativo 11.1:

Escreva um programa que realize o desenho apresentado abaixo. Você está livre para usar as cores e as dimensões que quiser. Para executar essa tarefa utilize os métodos `drawCircleSector` e `fillCircleSector`, detalhados abaixo:

Método:

```
1 void drawCircleSector( double x, double y, double radius,
2                         double startAngle, double endAngle,
3                         Paint paint );
```

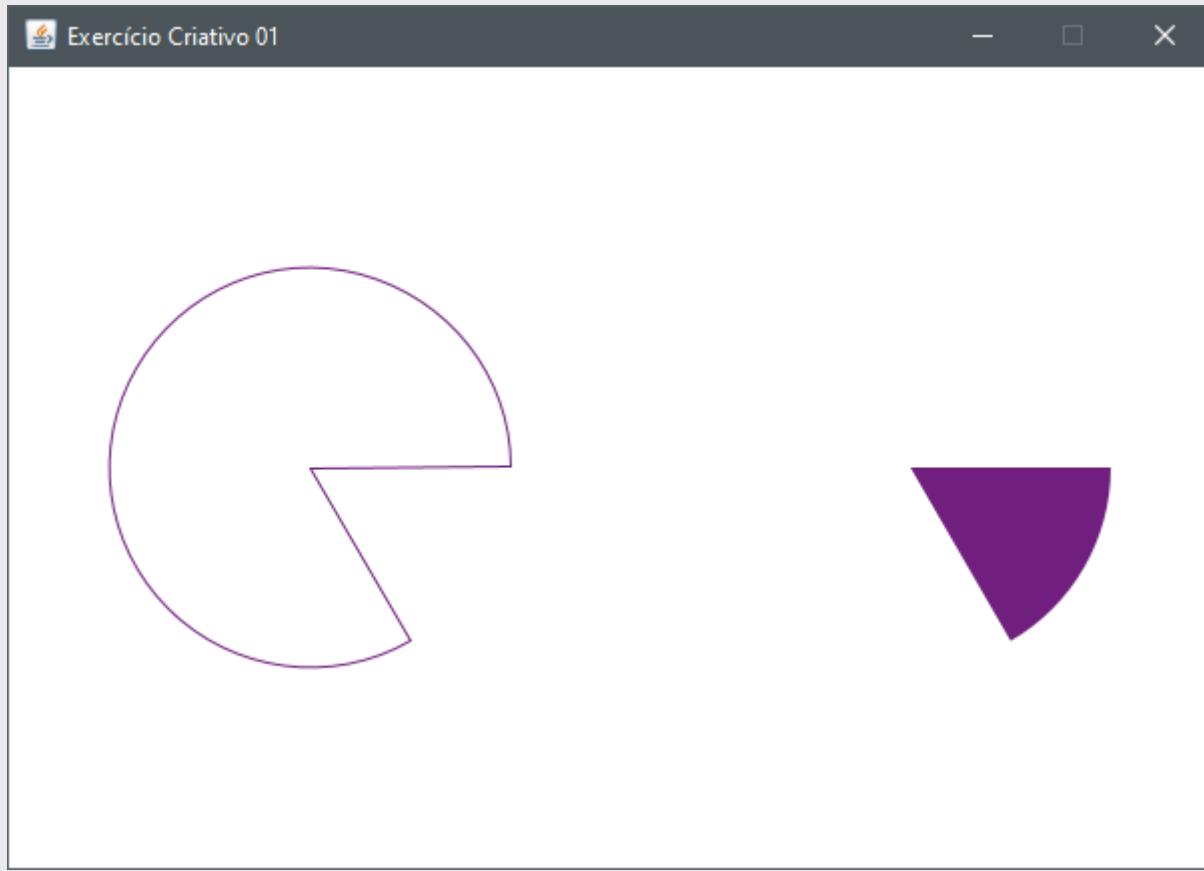
- **Nome:** `drawCircleSector`
- **Descrição:** Desenha o contorno de um setor circular.
- **Entrada/Parâmetro(s):**
 1. `x`: um número decimal que representa a coordenada x do centro do setor circular;
 2. `y`: um número decimal que representa a coordenada y do centro do setor circular;
 3. `radius`: o raio do setor circular;
 4. `startAngle`: o ângulo inicial, em graus (sentido horário);
 5. `endAngle`: o ângulo final, em graus (sentido horário);
 6. `paint`: paint para o desenho.
- **Saída/Retorno:** Esse método não retorna nenhum valor.

Método:

```
1 void fillCircleSector( double x, double y, double radius,
2                         double startAngle, double endAngle,
3                         Paint paint );
```

- **Nome:** `fillCircleSector`
- **Descrição:** Desenha um setor circular, preenchendo-o.
- **Entrada/Parâmetro(s):**
 1. `x`: um número decimal que representa a coordenada x do centro do setor circular;
 2. `y`: um número decimal que representa a coordenada y do centro do setor circular;
 3. `radius`: o raio do setor circular;
 4. `startAngle`: o ângulo inicial, em graus (sentido horário);
 5. `endAngle`: o ângulo final, em graus (sentido horário);
 6. `paint`: paint para o desenho.
- **Saída/Retorno:** Esse método não retorna nenhum valor.

Saída:



Exercício Criativo 11.2:

Escreva um programa que realize o desenho apresentado abaixo. Você está livre para usar as cores e as dimensões que quiser. Para executar essa tarefa utilize os métodos `drawRing` e `fillRing`, detalhados abaixo:

Método:

```

1 void drawRing( double x, double y,
2                 double innerRadius, double outerRadius,
3                 double startAngle, double endAngle,
4                 Paint paint );

```

- **Nome:** `drawRing`
- **Descrição:** Desenha o contorno de um anel.
- **Entrada/Parâmetro(s):**
 1. `x`: um número decimal que representa a coordenada *x* do centro do anel;
 2. `y`: um número decimal que representa a coordenada *y* do centro do anel;
 3. `innerRadius`: o raio interno do anel;
 4. `outerRadius`: o raio externo do anel;
 5. `startAngle`: o ângulo inicial, em graus (sentido horário);
 6. `endAngle`: o ângulo final, em graus (sentido horário);
 7. `paint`: paint para o desenho.
- **Saída/Retorno:** Esse método não retorna nenhum valor.

Método:

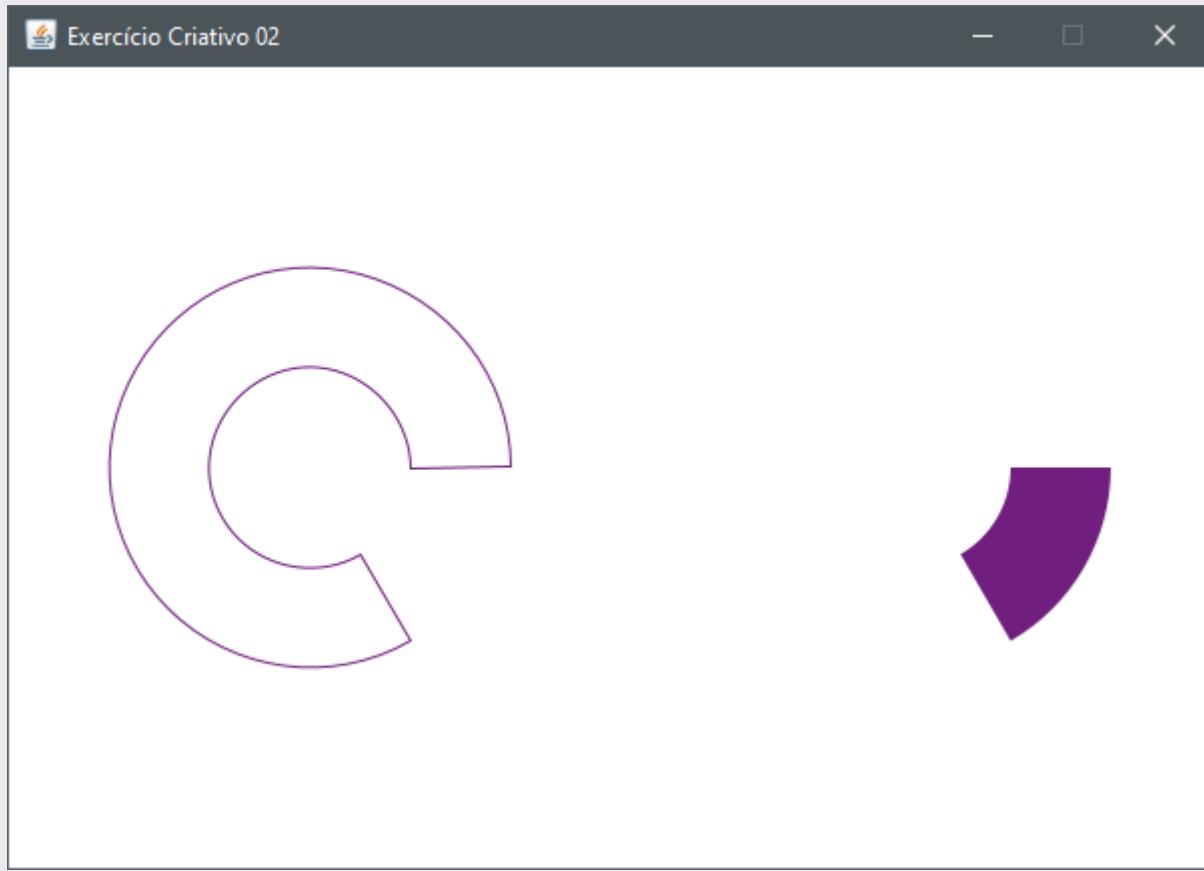
```

1 void fillRing( double x, double y,
2                 double innerRadius, double outerRadius,
3                 double startAngle, double endAngle,
4                 Paint paint );

```

- **Nome:** `fillRing`
- **Descrição:** Desenha um anel, preenchendo-o.
- **Entrada/Parâmetro(s):**
 1. `x`: um número decimal que representa a coordenada *x* do centro do anel;
 2. `y`: um número decimal que representa a coordenada *y* do centro do anel;
 3. `innerRadius`: o raio interno do anel;
 4. `outerRadius`: o raio externo do anel;
 5. `startAngle`: o ângulo inicial, em graus (sentido horário);
 6. `endAngle`: o ângulo final, em graus (sentido horário);
 7. `paint`: paint para o desenho.
- **Saída/Retorno:** Esse método não retorna nenhum valor.

Saída:



Exercício Criativo 11.3:

Escreva um programa que realize o desenho apresentado abaixo. Você está livre para usar as cores e as dimensões que quiser. Para executar essa tarefa utilize o método `fillRectangle`, detalhado abaixo. Além disso, caso queira que o desenho se assemelhe ao apresentado, foram usadas cores com variação no canal alfa (transparência). Para criar cores com o canal alfa alterado, baseadas numa cor base, consulte na documentação da classe `Color`² como criar cores com variação no canal alfa ou então use o método `fade` da classe `ColorUtils` da JSGE:

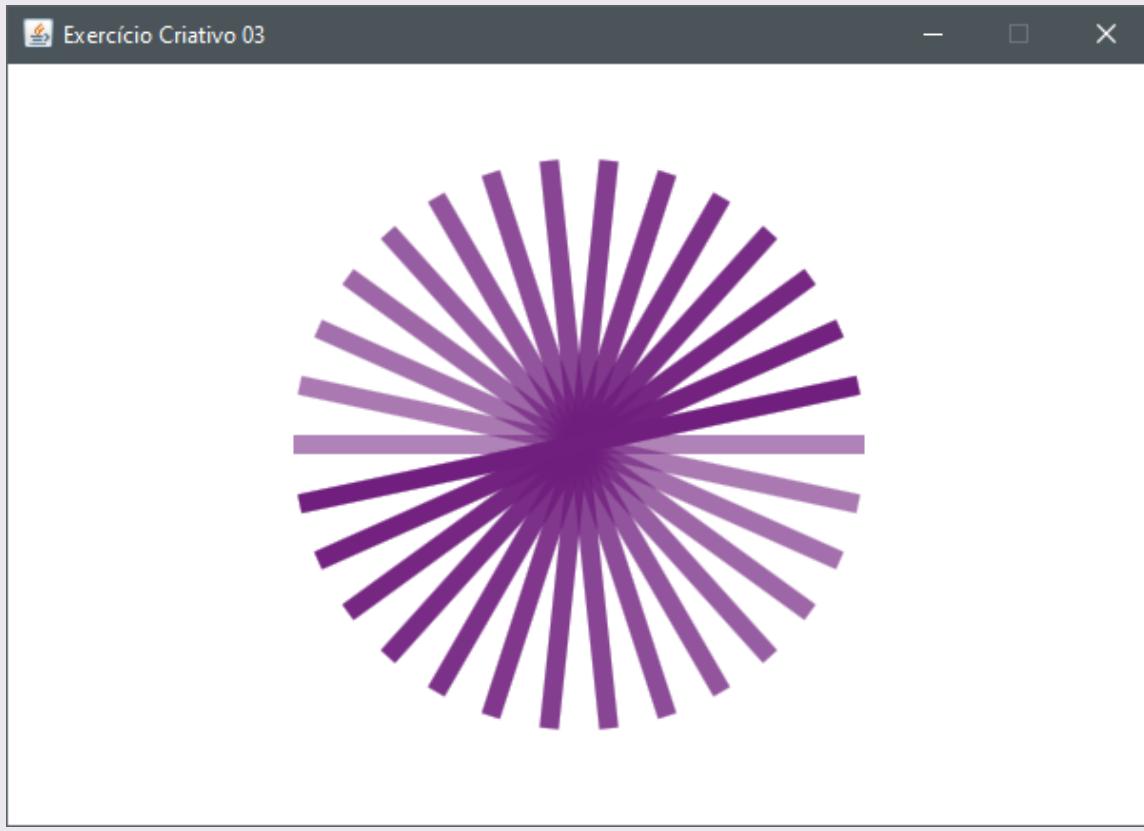
Método:

```
1 void fillRectangle( double x, double y,
2                     double width, double height,
3                     double originX, double originY,
4                     double rotation, Paint paint );
```

- **Nome:** `fillRectangle`
- **Descrição:** Desenha e pinta um retângulo rotacionado.
- **Entrada/Parâmetro(s):**
 1. `x`: um número decimal que representa a coordenada *x* do vértice superior esquerdo do retângulo;
 2. `y`: um número decimal que representa a coordenada *y* do vértice superior esquerdo do retângulo;
 3. `width`: a largura do retângulo;
 4. `height`: a altura do retângulo;
 5. `originX`: coordenada *x* do pivô de rotação;
 6. `originY`: coordenada *y* do pivô de rotação;
 7. `rotation`: o ângulo de rotação, em graus (sentido horário);
 8. `paint`: paint para o desenho.
- **Saída/Retorno:** Esse método não retorna nenhum valor.

²<<https://docs.oracle.com/en/java/javase/21/docs/api/java.desktop/java.awt/Color.html>>

Saída:



Exercício Criativo 11.4:

Escreva um programa que realize o desenho apresentado abaixo. Você está livre para usar as cores e as dimensões que quiser. Para executar essa tarefa utilize os métodos `drawRoundRectangle` e `fillRoundRectangle`, detalhados abaixo:

Método:

```

1 void drawRoundRectangle( double x, double y,
2                           double width, double height,
3                           double roundness,
4                           Paint paint );

```

- **Nome:** `drawRoundRectangle`
- **Descrição:** Desenha o contorno de um retângulo com os cantos arredondados.
- **Entrada/Parâmetro(s):**
 1. `x`: um número decimal que representa a coordenada *x* do vértice superior esquerdo do retângulo;
 2. `y`: um número decimal que representa a coordenada *y* do vértice superior esquerdo do retângulo;
 3. `width`: a largura do retângulo;
 4. `height`: a altura do retângulo;
 5. `roundness`: o arredondamento dos cantos;
 6. `paint`: paint para o desenho.
- **Saída/Retorno:** Esse método não retorna nenhum valor.

Método:

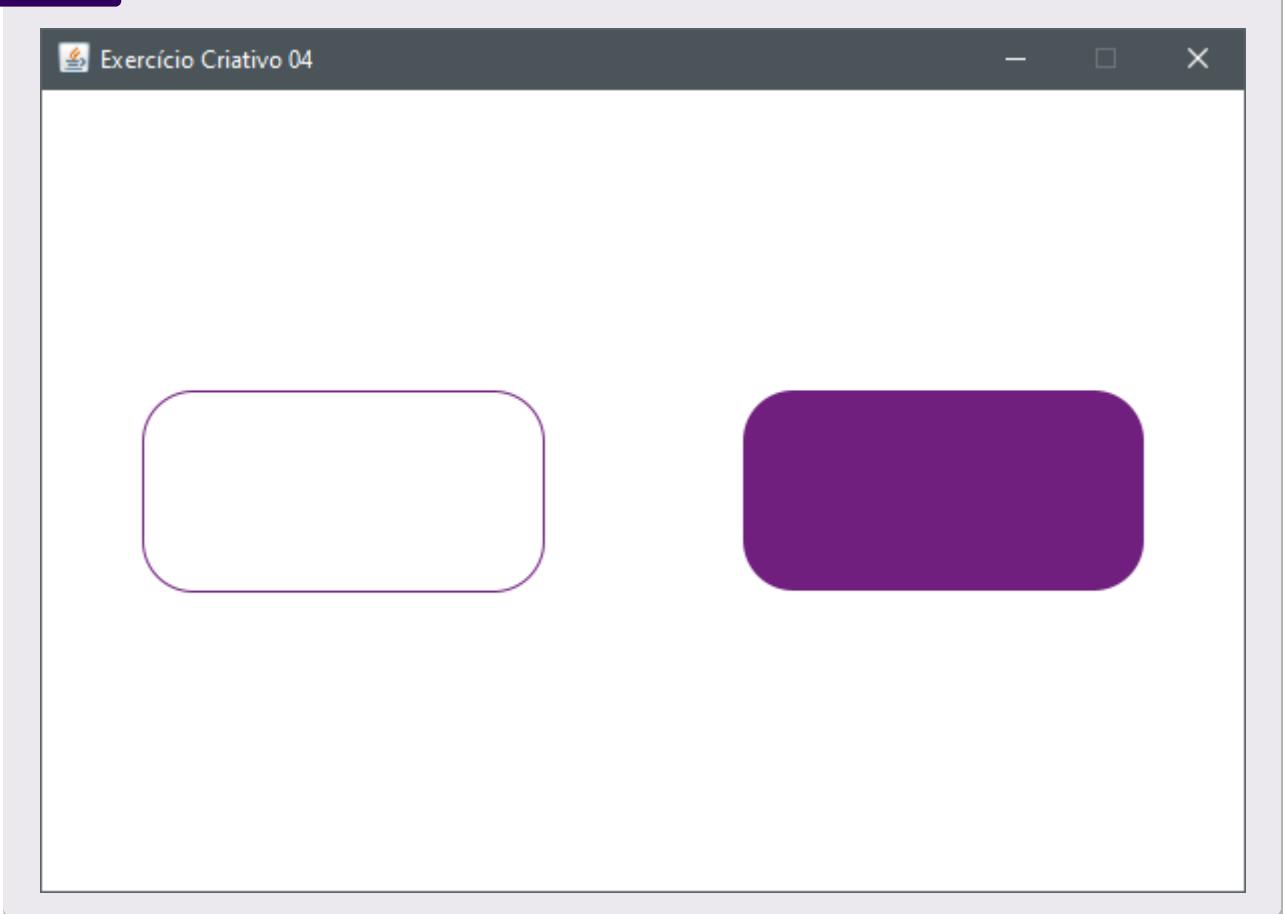
```

1 void fillRoundRectangle( double x, double y,
2                           double width, double height,
3                           double roundness,
4                           Paint paint );

```

- **Nome:** `fillRoundRectangle`
- **Descrição:** Desenha um retângulo com os cantos arredondados, preenchendo-o.
- **Entrada/Parâmetro(s):**
 1. `x`: um número decimal que representa a coordenada *x* do vértice superior esquerdo do retângulo;
 2. `y`: um número decimal que representa a coordenada *y* do vértice superior esquerdo do retângulo;
 3. `width`: a largura do retângulo;
 4. `height`: a altura do retângulo;
 5. `roundness`: o arredondamento dos cantos;
 6. `paint`: paint para o desenho.
- **Saída/Retorno:** Esse método não retorna nenhum valor.

Saída:



Exercício Criativo 11.5:

Escreva um programa que realize o desenho apresentado abaixo. Você está livre para usar as cores e as dimensões que quiser. Para executar essa tarefa utilize os métodos `drawTriangle` e `fillTriangle`, detalhados abaixo:

Método:

```

1 void drawTriangle( double v1x, double v1y,
2                     double v2x, double v2y,
3                     double v3x, double v3y,
4                     Paint paint );

```

- **Nome:** `drawTriangle`
- **Descrição:** Desenha o contorno de um triângulo. Atenção, os vértices precisam ser fornecidos no sentido horário!
- **Entrada/Parâmetro(s):**
 1. `v1x`: a coordenada x do primeiro vértice;
 2. `v1y`: a coordenada y do primeiro vértice;
 3. `v2x`: a coordenada x do segundo vértice;
 4. `v2y`: a coordenada y do segundo vértice;
 5. `v3x`: a coordenada x do terceiro vértice;
 6. `v3y`: a coordenada y do terceiro vértice;
 7. `paint`: paint para o desenho.
- **Saída/Retorno:** Esse método não retorna nenhum valor.

Método:

```

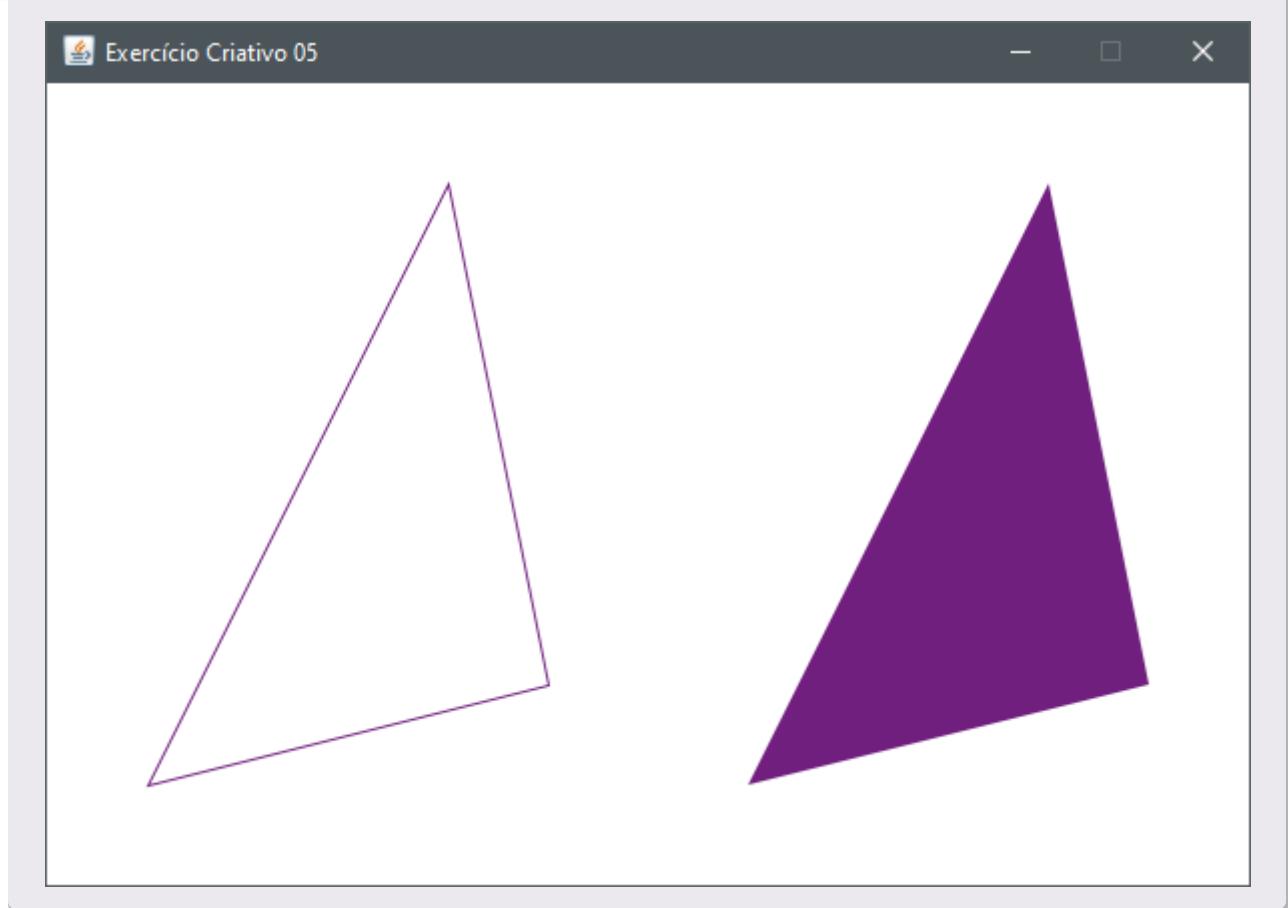
1 void fillTriangle( double v1x, double v1y,
2                     double v2x, double v2y,
3                     double v3x, double v3y,
4                     Paint paint );

```

- **Nome:** `fillTriangle`
- **Descrição:** Desenha um triângulo, preenchendo-o. Atenção, os vértices precisam ser fornecidos no sentido horário!
- **Entrada/Parâmetro(s):**
 1. `v1x`: a coordenada x do primeiro vértice;
 2. `v1y`: a coordenada y do primeiro vértice;
 3. `v2x`: a coordenada x do segundo vértice;
 4. `v2y`: a coordenada y do segundo vértice;
 5. `v3x`: a coordenada x do terceiro vértice;
 6. `v3y`: a coordenada y do terceiro vértice;
 7. `paint`: paint para o desenho.

- **Saída/Retorno:** Esse método não retorna nenhum valor.

Saída:



Exercício Criativo 11.6:

Escreva um programa que realize o desenho apresentado abaixo. Você está livre para usar as cores e as dimensões que quiser. Para executar essa tarefa utilize os métodos `drawPolygon` e `fillPolygon`, detalhados abaixo:

Método:

```
1 void drawPolygon( double x, double y,
2                   int sides, double radius,
3                   double rotation, Paint paint );
```

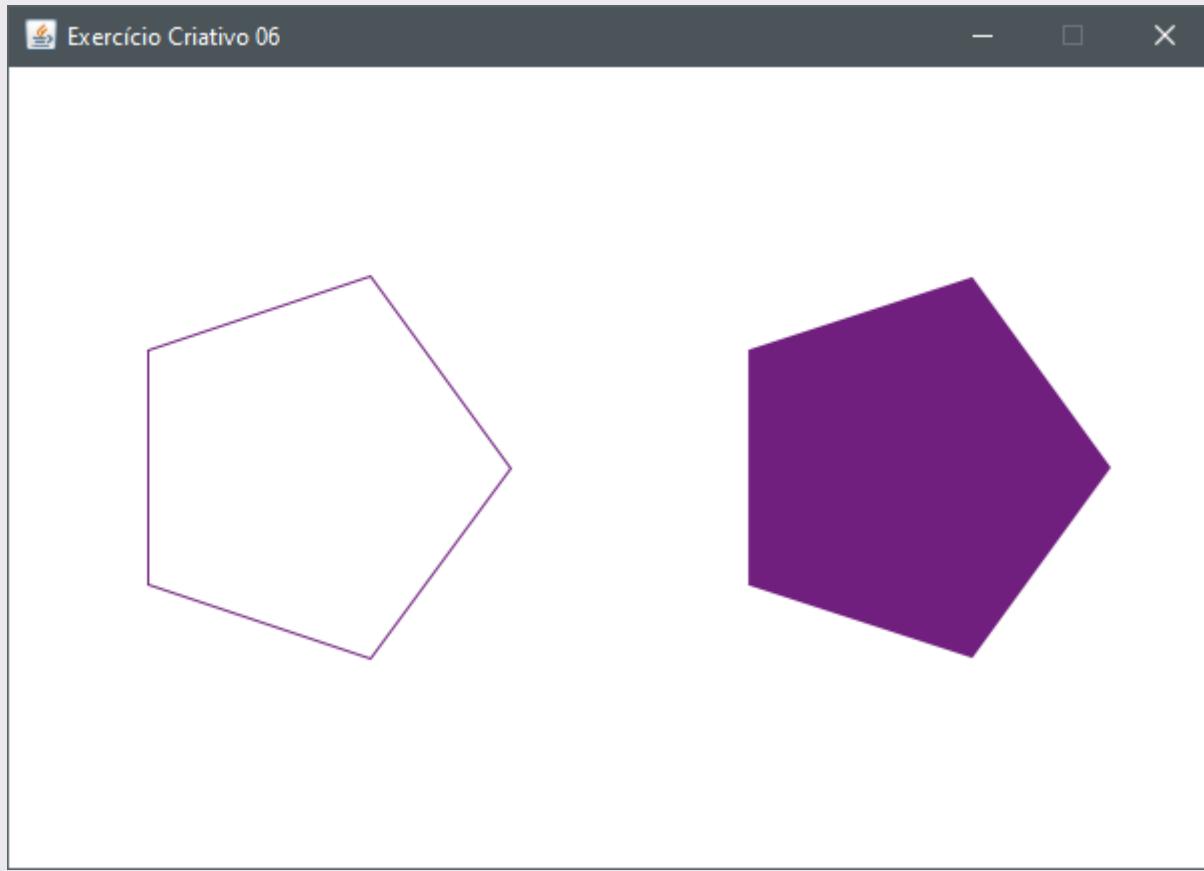
- **Nome:** `drawPolygon`
- **Descrição:** Desenha o contorno de um polígono regular.
- **Entrada/Parâmetro(s):**
 1. `x`: um número decimal que representa a coordenada x do centro do polígono;
 2. `y`: um número decimal que representa a coordenada y do centro do polígono;
 3. `sides`: a quantidade de lados;
 4. `radius`: o raio do polígono;
 5. `rotation`: a rotação do polígono, em graus (sentido horário);
 6. `paint`: paint para o desenho.
- **Saída/Retorno:** Esse método não retorna nenhum valor.

Método:

```
1 void fillPolygon( double x, double y,
2                   int sides, double radius,
3                   double rotation, Paint paint );
```

- **Nome:** `fillPolygon`
- **Descrição:** Desenha um polígono regular, preenchendo-o.
- **Entrada/Parâmetro(s):**
 1. `x`: um número decimal que representa a coordenada x do centro do polígono;
 2. `y`: um número decimal que representa a coordenada y do centro do polígono;
 3. `sides`: a quantidade de lados;
 4. `radius`: o raio do polígono;
 5. `rotation`: a rotação do polígono, em graus (sentido horário);
 6. `paint`: paint para o desenho.
- **Saída/Retorno:** Esse método não retorna nenhum valor.

Saída:



Exercício Criativo 11.7:

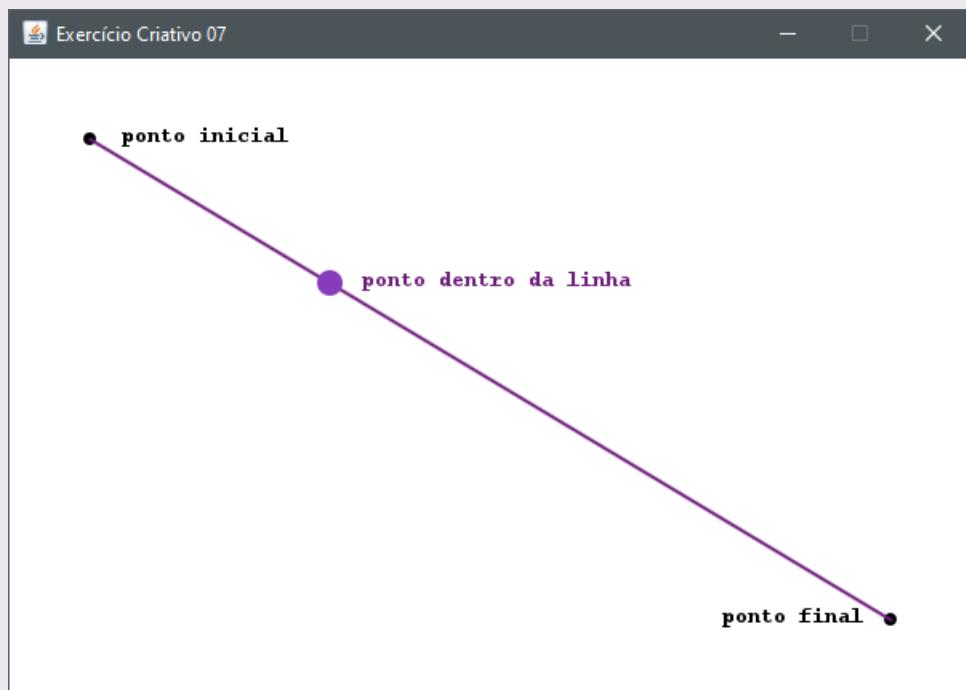
Escreva um programa que realize o desenho apresentado abaixo. Você está livre para usar as cores e as dimensões que quiser. Para executar essa tarefa utilize o método `drawLine`, já aprendido, e o método `getPointAtLine`, esse da classe `CurveUtils`, detalhado abaixo:

Método:

```
1 Vector2 CurveUtils.getPointAtLine( double p1x, double p1y,
2                                     double p2x, double p2y,
3                                     double amount );
```

- **Nome:** `getPointAtLine`
- **Descrição:** Obtém um ponto dentro de uma linha.
- **Entrada/Parâmetro(s):**
 1. `p1x`: coordenada x do ponto inicial;
 2. `p1y`: coordenada y do ponto inicial;
 3. `p2x`: coordenada x do ponto final;
 4. `p2y`: coordenada y do ponto final;
 5. `amount`: a localização percentual do ponto desejado dentro da linha, um valor no intervalo [0..1];
- **Saída/Retorno:** o ponto localizado no percentual `amount` dentro da linha (`Vector2`).

Saída:



Exercício Criativo 11.8:

Escreva um programa que realize o desenho apresentado abaixo. Você está livre para usar as cores e as dimensões que quiser. Para executar essa tarefa utilize os métodos `drawQuadCurve` e `getPointAtQuadCurve`, esse da classe `CurveUtils`, detalhados abaixo:

Método:

```
1 void drawQuadCurve( double p1x, double p1y,
2                     double cx, double cy,
3                     double p2x, double p2y,
4                     Paint paint );
```

- **Nome:** `drawQuadCurve`
- **Descrição:** Desenha um segmento de uma curva do tipo Bézier Quadrática (um ponto de controle).
- **Entrada/Parâmetro(s):**
 1. `p1x`: coordenada x do ponto inicial;
 2. `p1y`: coordenada y do ponto inicial;
 3. `cx`: coordenada x do ponto de controle;
 4. `cy`: coordenada y do ponto de controle;
 5. `p2x`: coordenada x do ponto final;
 6. `p2y`: coordenada y do ponto final;
 7. `paint`: paint para o desenho.
- **Saída/Retorno:** Esse método não retorna nenhum valor.

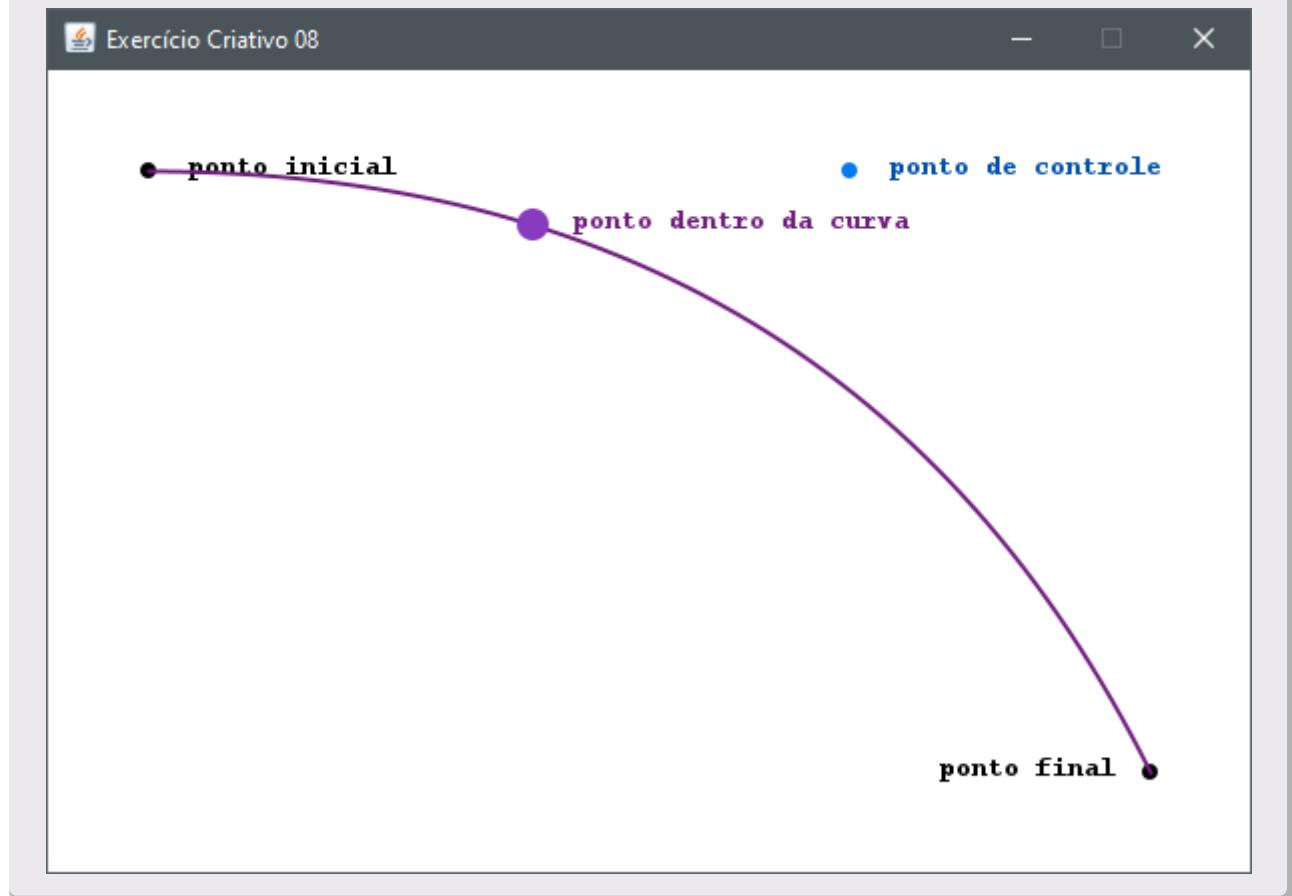
Método:

```
1 Vector2 CurveUtils.getPointAtQuadCurve( double p1x, double p1y,
2                                         double cx, double cy,
3                                         double p2x, double p2y,
4                                         double amount );
```

- **Nome:** `getPointAtQuadCurve`
- **Descrição:** Obtém um ponto dentro de uma curva Bézier Quadrática.
- **Entrada/Parâmetro(s):**
 1. `p1x`: coordenada x do ponto inicial;
 2. `p1y`: coordenada y do ponto inicial;
 3. `cx`: coordenada x do ponto de controle;
 4. `cy`: coordenada y do ponto de controle;
 5. `p2x`: coordenada x do ponto final;
 6. `p2y`: coordenada y do ponto final;
 7. `amount`: a localização percentual do ponto desejado dentro da curva, um valor no intervalo [0..1];

- **Saída/Retorno:** o ponto localizado no percentual `amount` dentro da curva `(Vector2)`.

Saída:



Exercício Criativo 11.9:

Escreva um programa que realize o desenho apresentado abaixo. Você está livre para usar as cores e as dimensões que quiser. Para executar essa tarefa utilize os métodos `drawCubicCurve` e `getPointAtCubicCurve`, esse da classe `CurveUtils`, detalhados abaixo:

Método:

```

1 void drawCubicCurve( double p1x, double p1y,
2                      double c1x, double c1y,
3                      double c2x, double c2y,
4                      double p2x, double p2y,
5                      Paint paint );

```

- **Nome:** `drawCubicCurve`
- **Descrição:** Desenha um segmento de uma curva do tipo Bézier Cúbica (dois pontos de controle).
- **Entrada/Parâmetro(s):**
 1. `p1x`: coordenada x do ponto inicial;
 2. `p1y`: coordenada y do ponto inicial;
 3. `c1x`: coordenada x do primeiro ponto de controle;
 4. `c1y`: coordenada y do primeiro ponto de controle;
 5. `c2x`: coordenada x do segundo ponto de controle;
 6. `c2y`: coordenada y do segundo ponto de controle;
 7. `p2x`: coordenada x do ponto final;
 8. `p2y`: coordenada y do ponto final;
 9. `paint`: paint para o desenho.
- **Saída/Retorno:** Esse método não retorna nenhum valor.

Método:

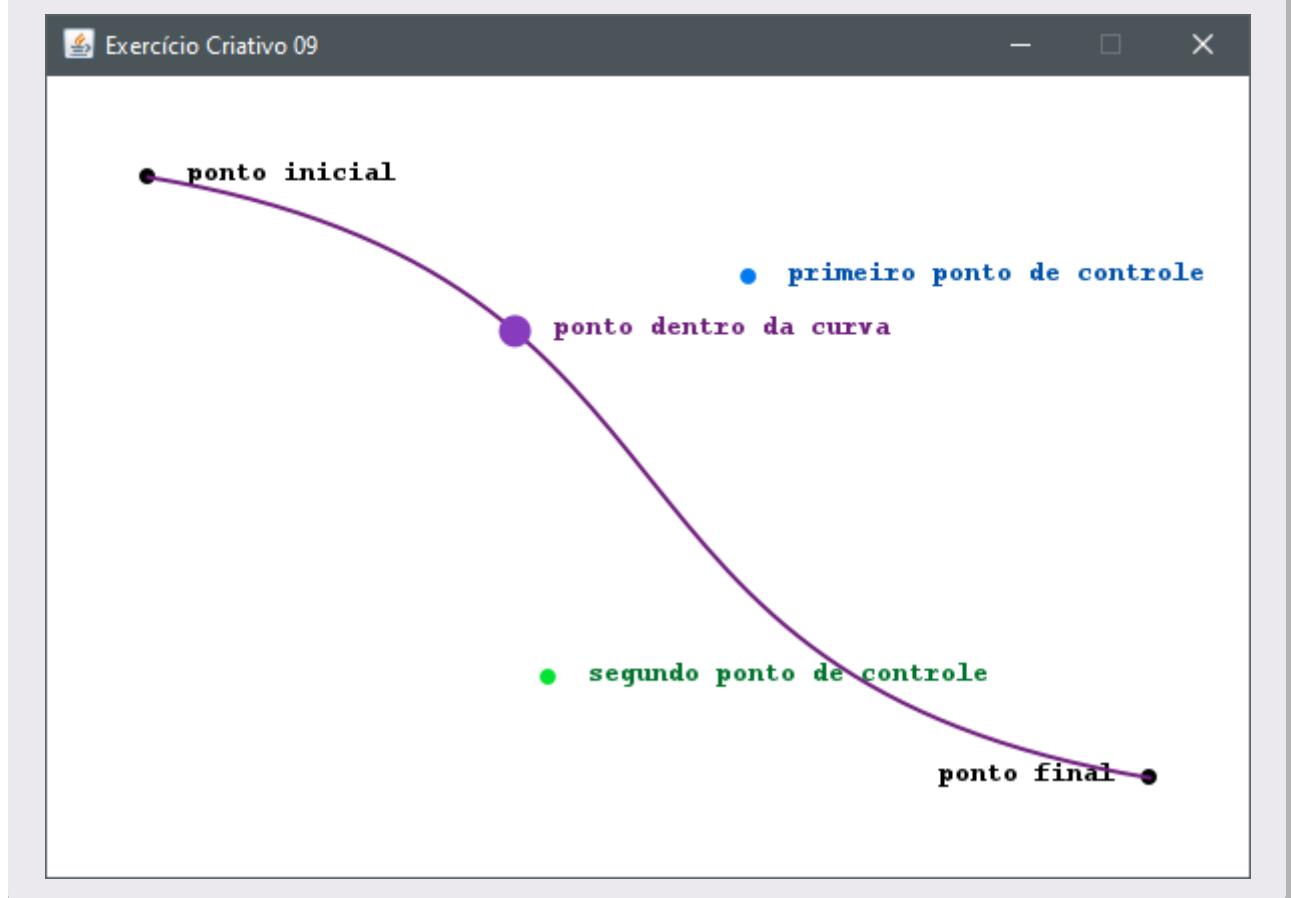
```

1 Vector2 CurveUtils.getPointAtCubicCurve( double p1x, double p1y,
2                                         double c1x, double c1y,
3                                         double c2x, double c2y,
4                                         double p2x, double p2y,
5                                         double amount );

```

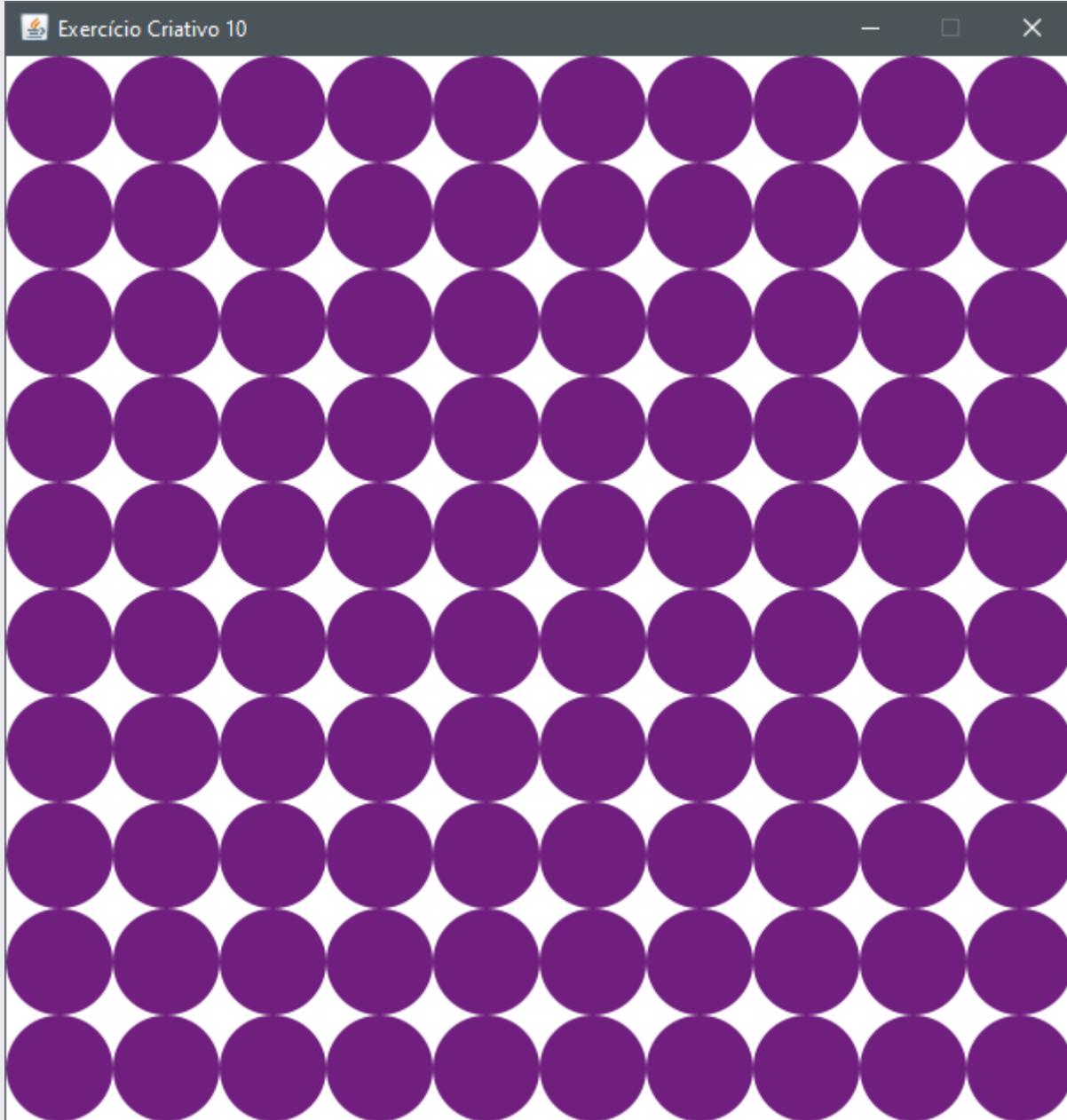
- **Nome:** `getPointAtCubicCurve`
- **Descrição:** Obtém um ponto dentro de uma curva Bézier Cúbica.
- **Entrada/Parâmetro(s):**
 1. `p1x`: coordenada x do ponto inicial;
 2. `p1y`: coordenada y do ponto inicial;
 3. `c1x`: coordenada x do primeiro ponto de controle;
 4. `c1y`: coordenada y do primeiro ponto de controle;

5. `c2x`: coordenada x do segundo ponto de controle;
 6. `c2y`: coordenada y do segundo ponto de controle;
 7. `p2x`: coordenada x do ponto final;
 8. `p2y`: coordenada y do ponto final;
 9. `amount`: a localização percentual do ponto desejado dentro da curva, um valor no intervalo [0..1];
- **Saída/Retorno:** o ponto localizado no percentual `amount` dentro da curva (`Vector2`).

Saída:

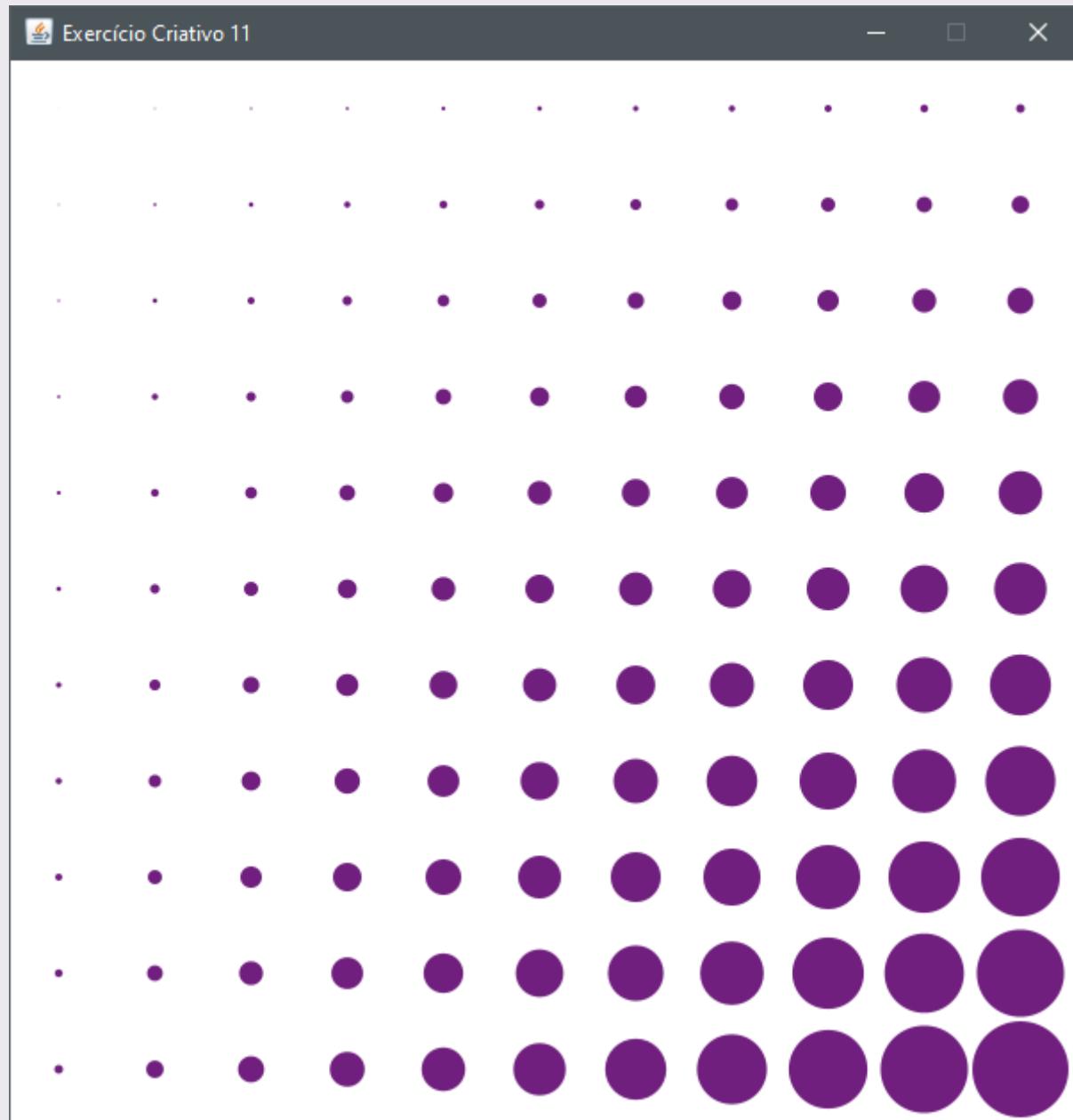
Exercício Criativo 11.10:

Escreva um programa que desenhe o padrão abaixo. Você está livre para usar as cores que quiser. A única restrição é que sua tela deve ter dimensões 600x600 pixels.

Saída:

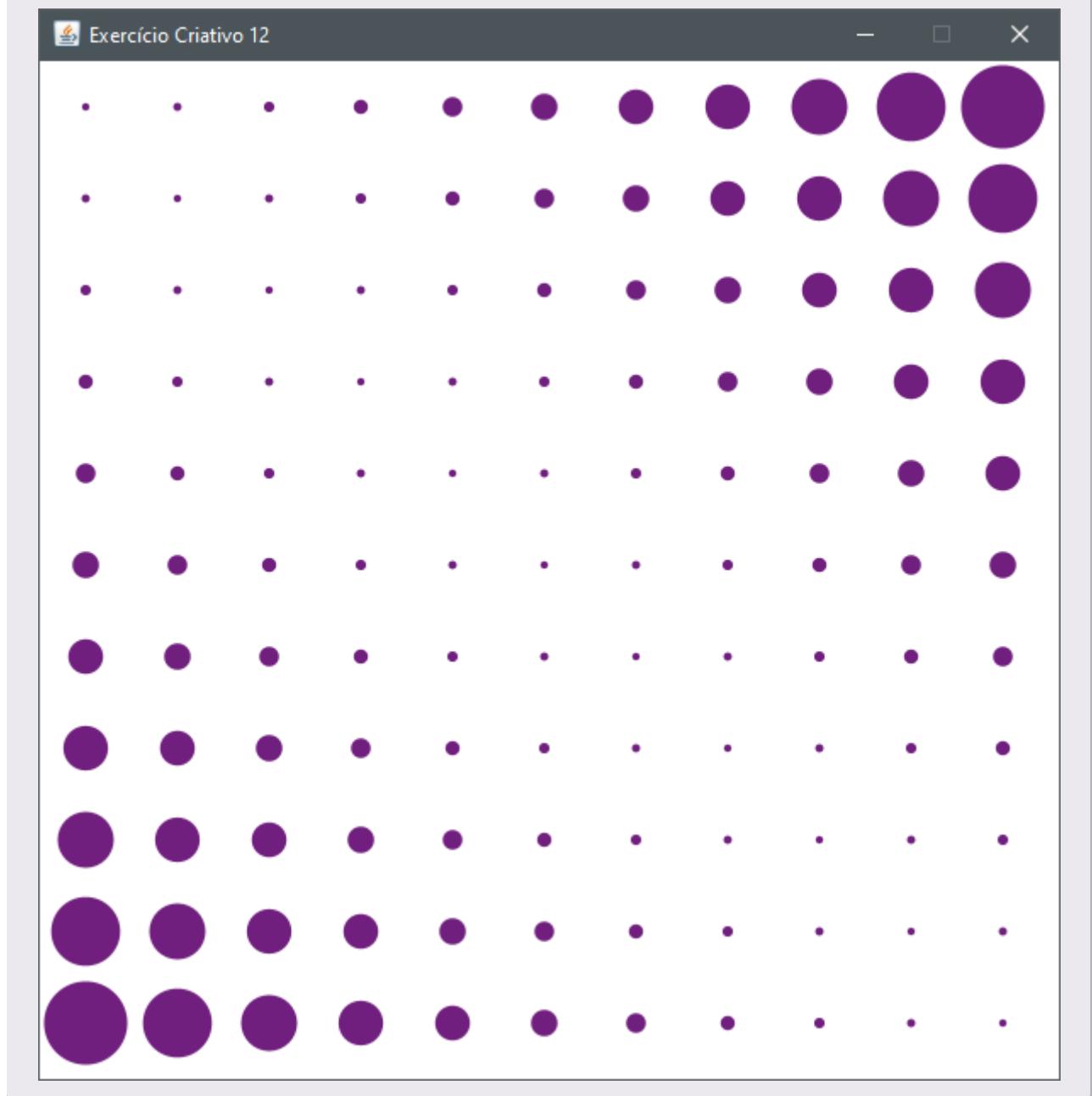
Exercício Criativo 11.11:

Escreva um programa que desenhe o padrão abaixo. Você está livre para usar as cores que quiser. A única restrição é que sua tela deve ter dimensões 600x600 pixels.

Saída:

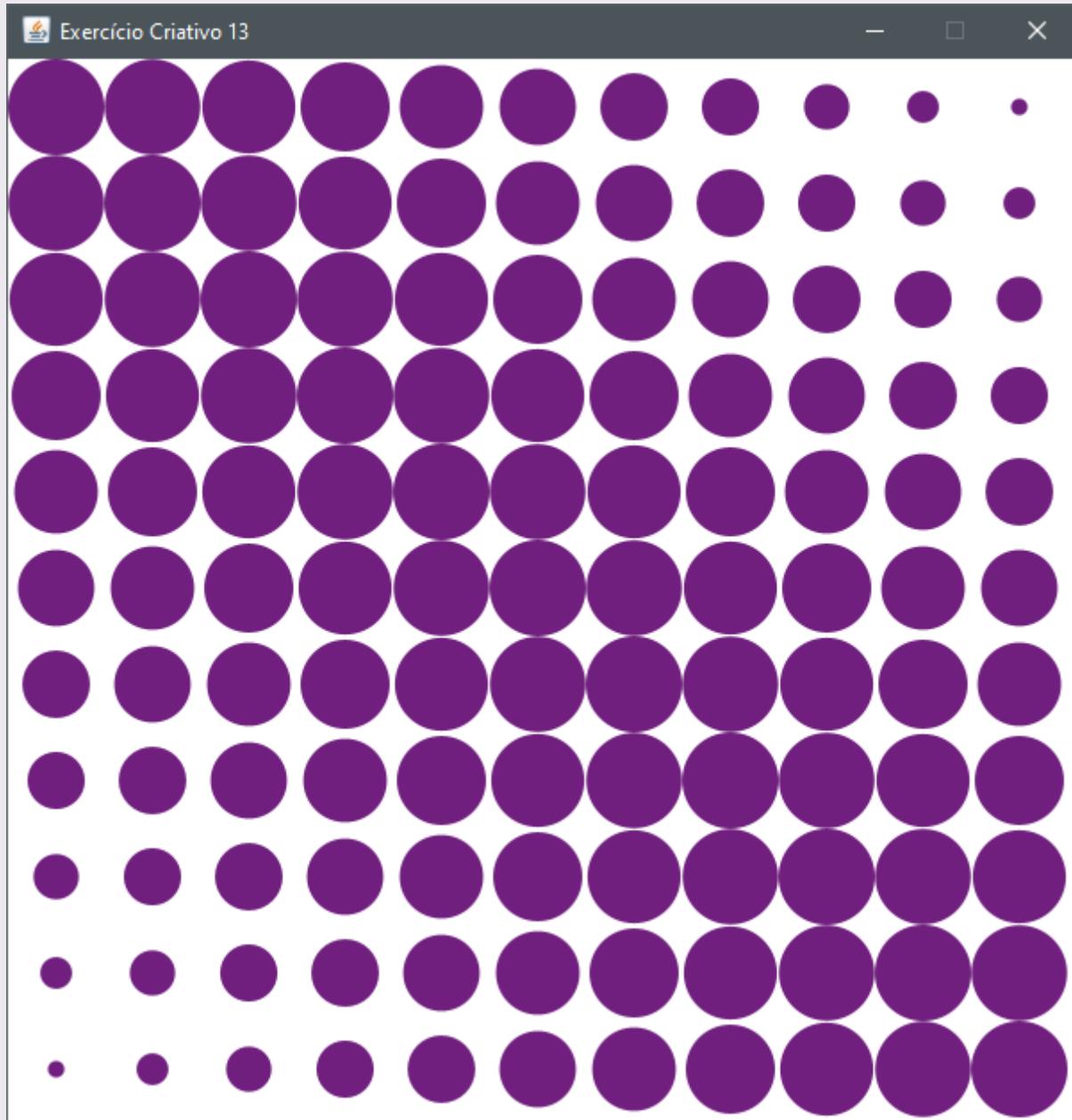
Exercício Criativo 11.12:

Escreva um programa que desenhe o padrão abaixo. Você está livre para usar as cores que quiser. A única restrição é que sua tela deve ter dimensões 600x600 pixels.

Saída:

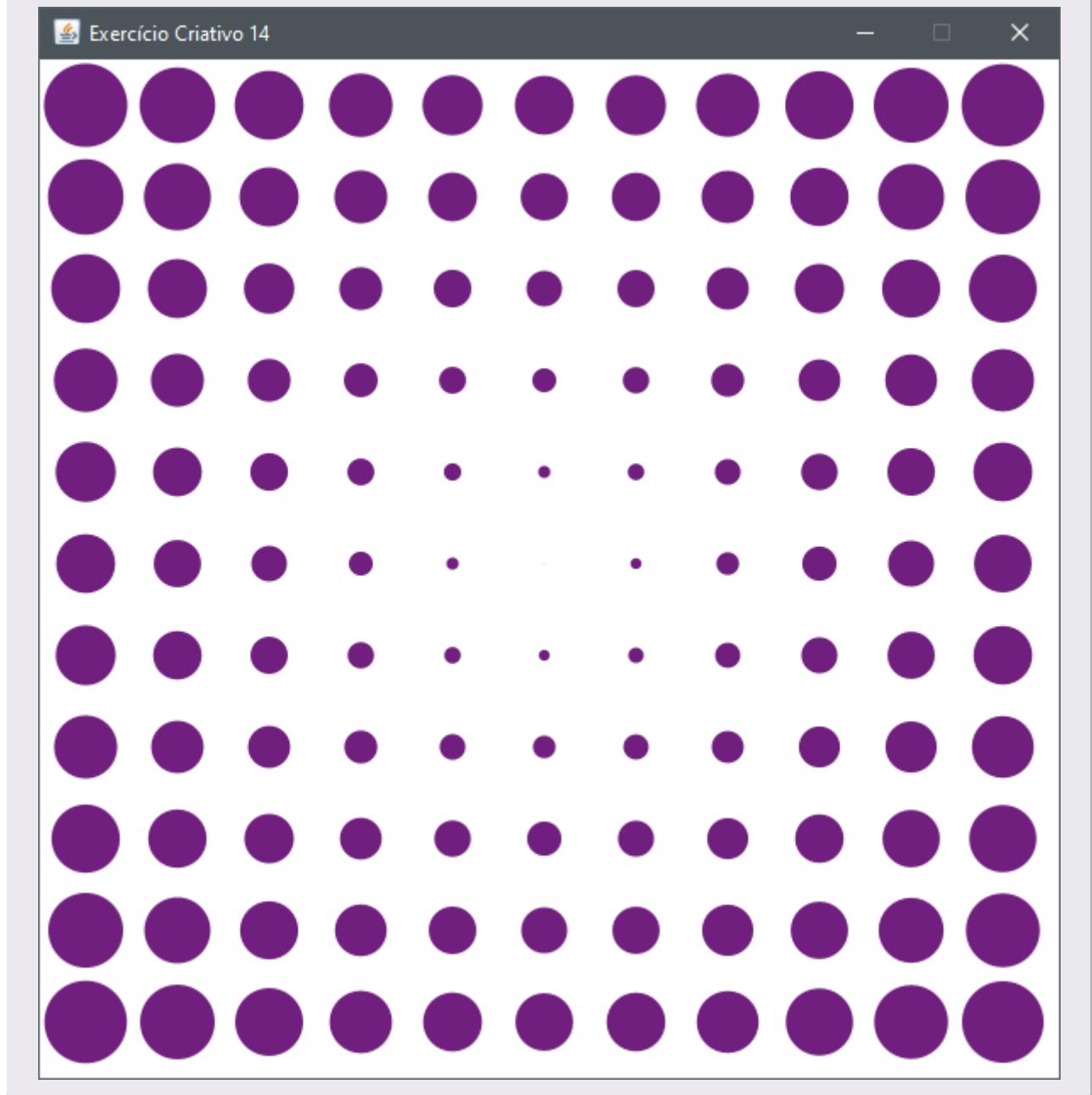
Exercício Criativo 11.13:

Escreva um programa que desenhe o padrão abaixo. Você está livre para usar as cores que quiser. A única restrição é que sua tela deve ter dimensões 600x600 pixels.

Saída:

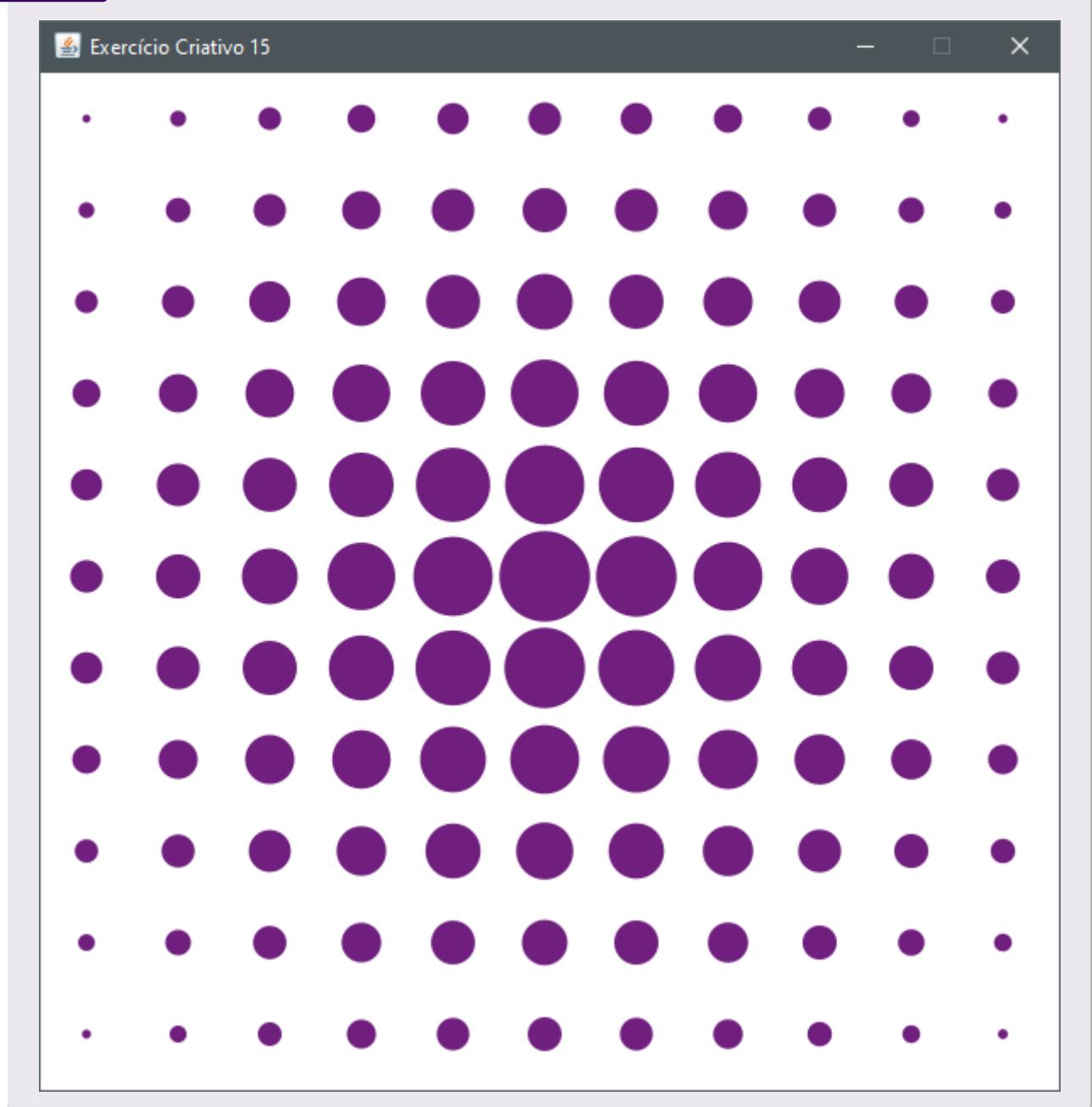
Exercício Criativo 11.14:

Escreva um programa que desenhe o padrão abaixo. Você está livre para usar as cores que quiser. A única restrição é que sua tela deve ter dimensões 600x600 pixels.

Saída:

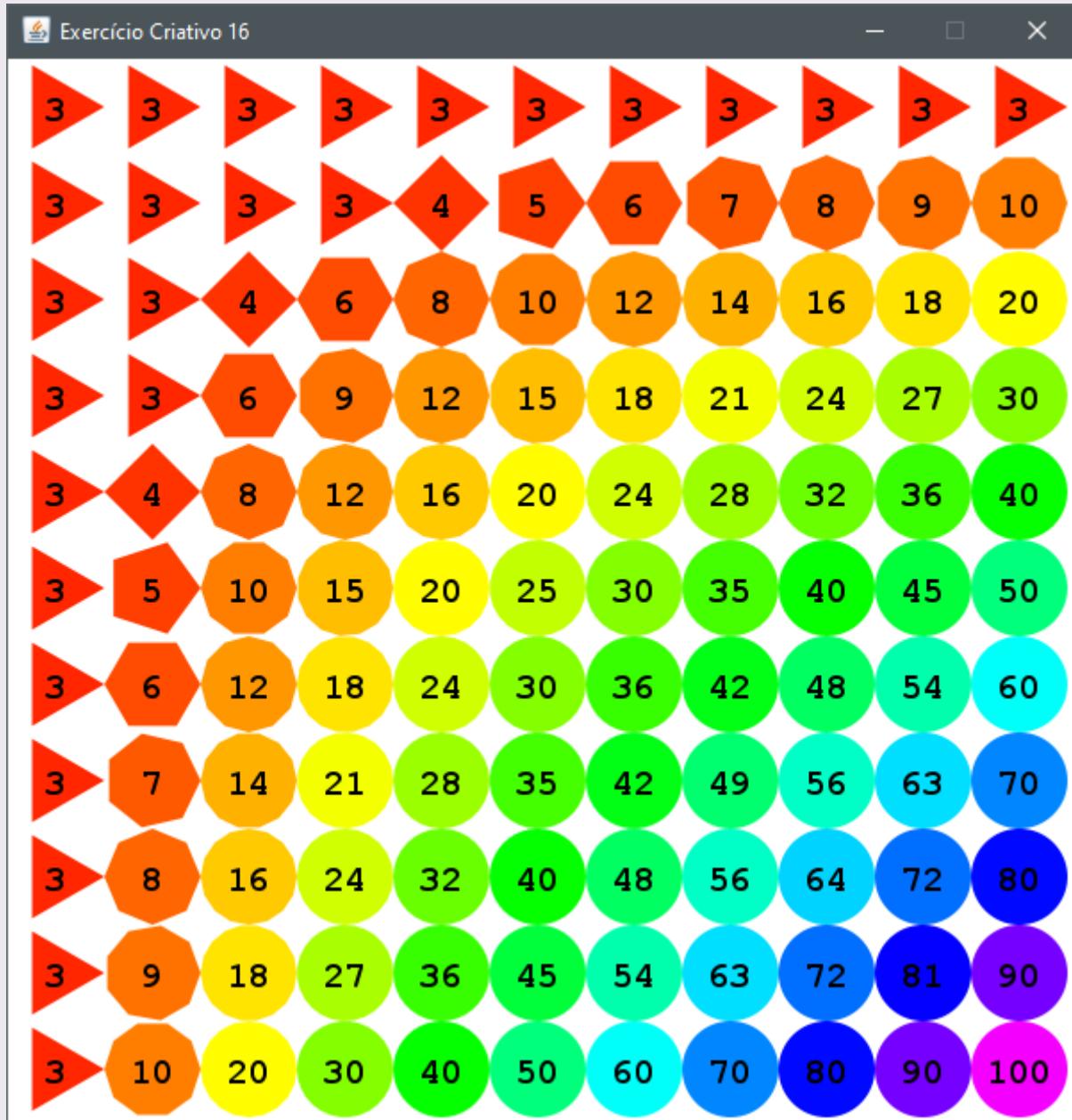
Exercício Criativo 11.15:

Escreva um programa que desenhe o padrão abaixo. Você está livre para usar as cores que quiser. A única restrição é que sua tela deve ter dimensões 600x600 pixels.

Saída:

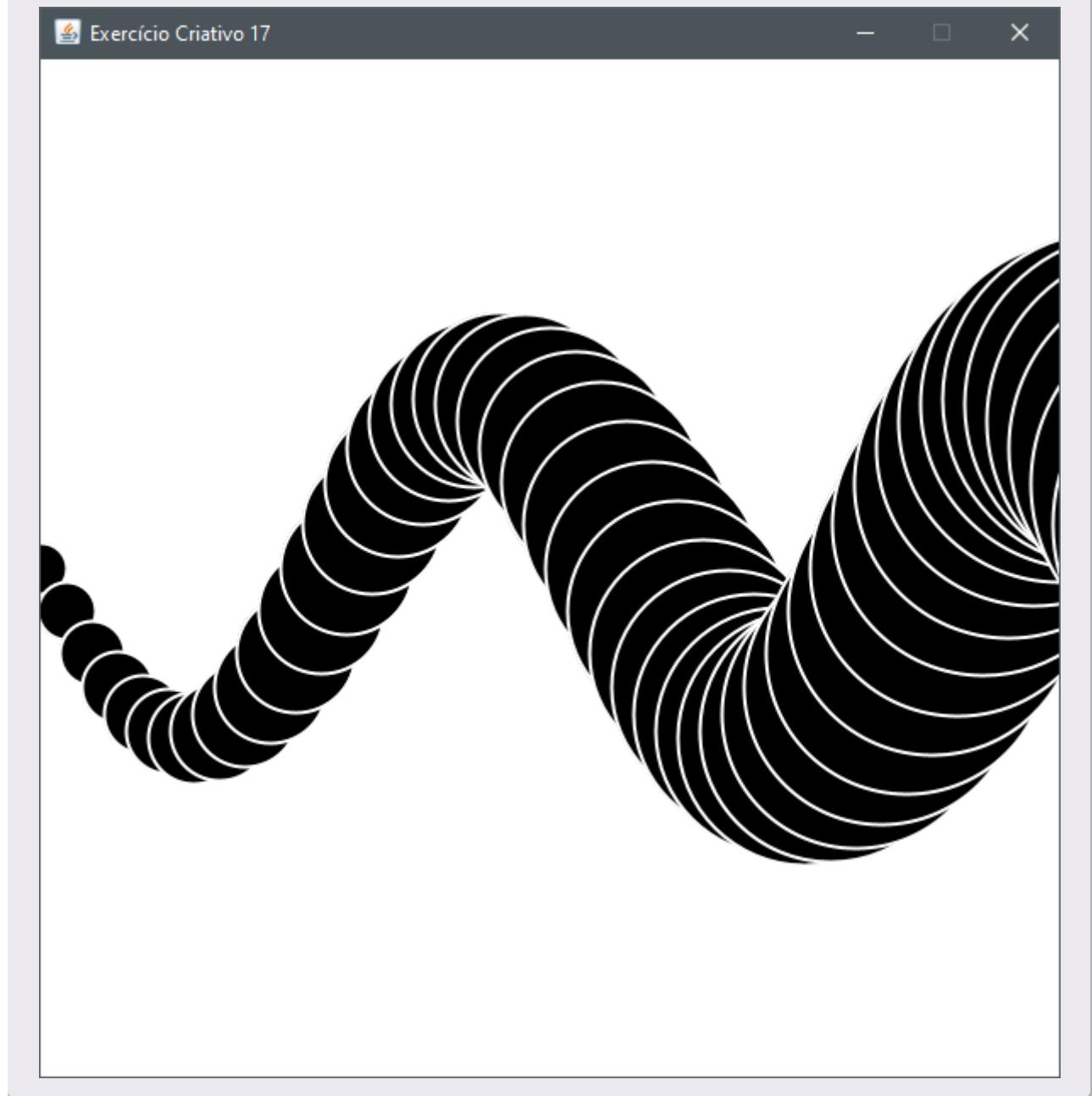
Exercício Criativo 11.16:

Escreva um programa que desenhe o padrão abaixo. Você está livre para usar as cores que quiser. A única restrição é que sua tela deve ter dimensões 600x600 pixels.

Saída:

Exercício Criativo 11.17:

Escreva um programa que desenhe o padrão abaixo. Você está livre para usar as cores que quiser. A única restrição é que sua tela deve ter dimensões 600x600 pixels.

Saída:

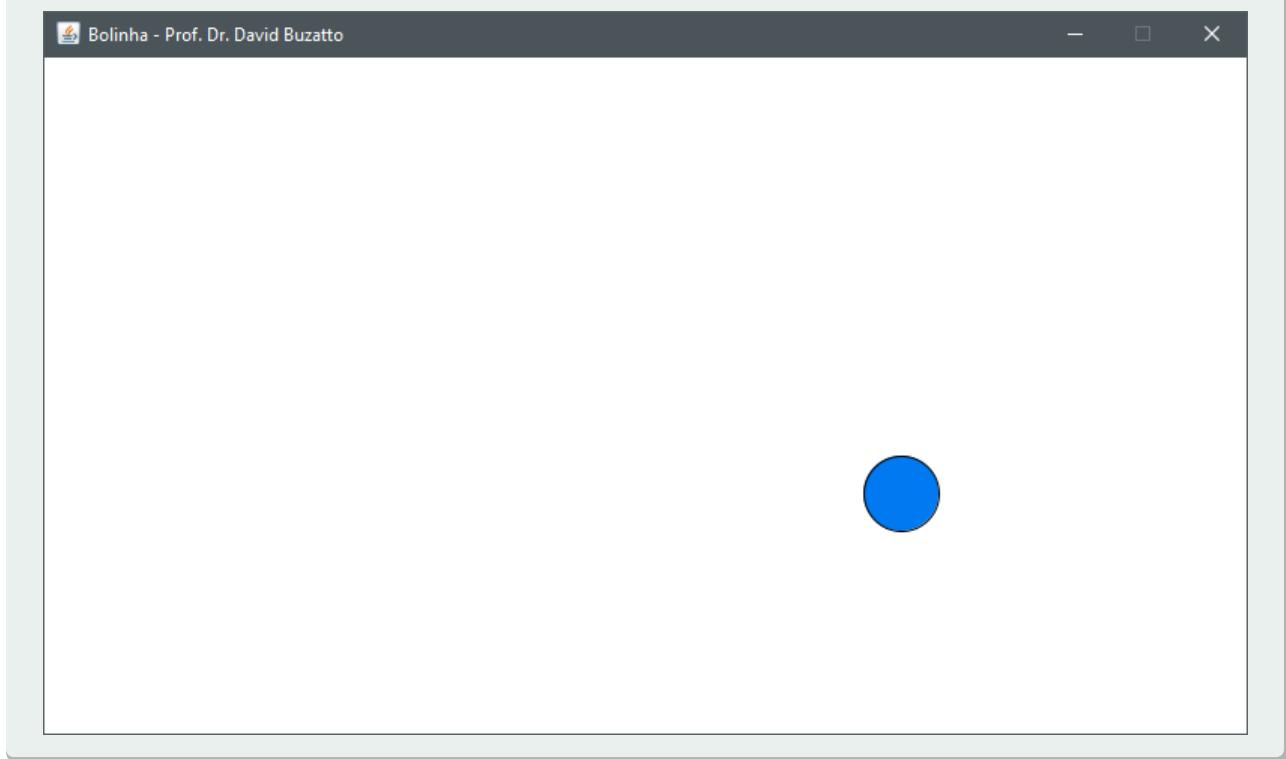
11.6 Projetos

Agora vamos realizar dois projetos guiados. Faremos passo a passo juntos, em aula. O primeiro deles será um tipo de simulação física com uma bolinha e o outro será um jogo do tipo Pong. Ambos os projetos podem ser encontrados no pacote de código fonte do livro.

Projeto 11.1:

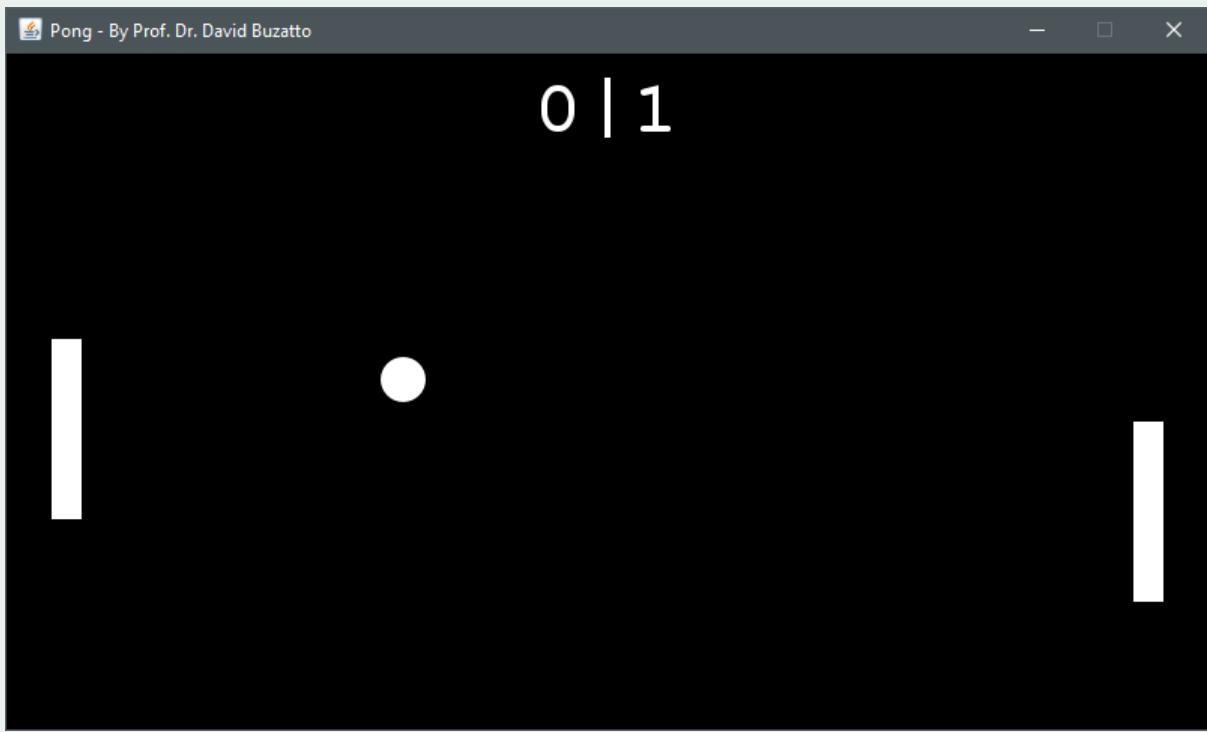
Desenvolva um programa que simule velocidade, atrito, elasticidade e gravidade em uma “bolinha”, que deve ficar restrita à tela da simulação.

Saída:



Projeto 11.2:

Desenvolva um jogo do tipo Pong para dois jogadores.

Saída:

ENCAPSULAMENTO

“O segredo da criatividade é saber como esconder as fontes”.

Cyril Edwin Mitchinson Joad



ESTE Capítulo trabalharemos com o conceito de Encasulamento, um dos alicerces do Paradigma Orientado a Objetos. Para isso aplicaremos aos atributos das nossas classes um padrão da plataforma chamado de Java Beans. Os conceitos e suas respectivas implementações serão apresentados, como de praxe, nos exemplos de código. Vamos lá.

12.1 Exemplos em Linguagem Java

Aplicação do Encapsulamento na classe Data

```
1  /*
2  * Arquivo: capitulo12/Data.java
3  * Autor: Prof. Dr. David Buzatto
4  *
5  * Um dos conceitos importantes relacionados ao Paradigma Orientado a
6  * Objetos é o Encapsulamento. Como o próprio nome diz, o encapsulamento
7  * tem como objetivo encapsular alguma coisa na nossa implementação, mas
8  * o que seria essa coisa?
9  *
10 * A ideia gira em torno do processo de esconder a implementação da classe
11 * de seu cliente. Quando dizemos cliente, estamos nos referindo a quem
12 * usa a classe em questão, ou seja, outras classes, que por sua vez
13 * são escritas por outros programadores ou até nós mesmos.
14 *
15 * O cliente da classe precisa saber apenas qual é a interface pública
16 * da mesma, ou seja, quais são os membros públicos que podem ser acessados.
17 * Como esses membros foram programados não é de responsabilidade do cliente
18 * da classe. O desenvolvedor da classe, por sua vez, precisa garantir que
19 * através da interface pública da mesma, o usuário não conseguirá colocar
20 * objetos criados a partir dela em um estado inválido.
21 *
22 * De forma análoga a um exemplo da vida real, podemos pensar em medicações.
23 * Cada cápsula ou comprimido de um remédio é feito para ser consumido em
24 * determinados intervalos de tempo por um determinado período. Nós sabemos
25 * os componentes do remédio, temos ideia de para o que ele serve, mas não
26 * sabemos exatamente quais são as vias metabólicas que serão ativadas ou
27 * reprimidas no organismo que está recebendo aquela medicação. Veja, a
28 * ideia é a mesma. Sabemos para que serve e como deve ser usado, mas não
29 * precisamos saber exatamente como o remédio foi feito, desde que ele
30 * funcione.
31 *
32 * No contexto da classe Data, um exemplo poderia ser um mês fora do
33 * intervalo aceitável que é entre 1 e 12, ambos inclusivos. Há situações
34 * onde queremos criar classes para representar tipos com o objetivo de
35 * apenas transportar dados e, nesses casos, na maioria das vezes, não há
36 * a necessidade de se usar o Encapsulamento de forma tão restritiva.
37 *
38 * Nesse e nos próximos exemplos iremos lidar com um padrão da plataforma
39 * Java chamado Java Beans, onde a ideia é definir uma interface pública
40 * com nomenclatura padronizada para que possamos encapsular principalmente
41 * os atributos das nossas classes e, ao mesmo tempo, torná-las componentes
42 * reutilizáveis. Há uma certa burocracia envolvida e existem assistentes
43 * nas ferramentas de desenvolvimento que permitem a criação automática de
44 * alguns trechos de código para nós, além de bibliotecas dedicadas a isso,
45 * como a Lombok, mas primeiramente vamos ver e entender como é feito
```

```
46 * manualmente.  
47 *  
48 * Note ainda que Java Beans não é necessariamente sinônimo de  
49 * Encapsulamento. Encapsulamento é um conceito mais abrangente do que uma  
50 * padronização da linguagem que estamos trabalhando. Por fim, o padrão  
51 * Java Beans ainda engloba diversas outras regras a serem seguidas, mas que  
52 * não veremos por enquanto.  
53 */  
54 public class Data {  
55  
56     // atributos privados  
57     private int dia;  
58     private int mes;  
59     private int ano;  
60  
61     /**  
62      * Construtor padrão.  
63      */  
64     public Data() {  
65         dia = 1;  
66         mes = 1;  
67         ano = 1970;  
68     }  
69  
70     /**  
71      * Construtor com três parâmetros.  
72      *  
73      * @param dia O dia da Data  
74      * @param mes O mês da Data  
75      * @param ano O ano da Data  
76      */  
77     public Data( int dia, int mes, int ano ) {  
78         this.dia = dia;  
79         this.mes = mes;  
80         this.ano = ano;  
81     }  
82  
83     /*  
84      * ***** métodos acessores ou getters (get) *****  
85      *  
86      * Os métodos get (obter) têm a função de RETORNAR o valor de um  
87      * atributo, seja ele representado de fato por um atributo privado  
88      * da instância da classe, ou algo que foi calculado e retornado.  
89      *  
90      * O usuário da classe não precisa saber qual ou quais atributos foram  
91      * usados para a obtenção do valor em questão.  
92      *  
93      * No padrão Java Beans, cada atributo que queremos que seja exposto à  
94      * leitura do cliente receberá um método get correspondente.
```

```
95  *
96  * O padrão para definição dos métodos get é:
97  *
98  * public tipo getNome(),
99  * onde:
100 *     O nome do método inicia obrigatoriamente com o prefixo get,
101 *     seguido do nome do atributo que se quer acessar;
102 *     O método é público;
103 *     O tipo de retorno é o mesmo do atributo sendo retornado.
104 */
105
106 /**
107 * Obtém o dia da Data em questão.
108 *
109 * @return O dia da Data.
110 */
111 public int getDia() {
112     return dia;
113 }
114
115 public int getMes() {
116     return mes;
117 }
118
119 public int getAno() {
120     return ano;
121 }
122
123 /*
124 * ***** métodos modificadores ou setters (set) *****
125 *
126 * Os métodos set (modificar/colocar/determinar) têm a função de ALTERAR
127 * o valor de um atributo, seja ele representado de fato por um atributo
128 * privado da instância da classe, ou algo que será calculado.
129 *
130 * O usuário da classe não precisa saber qual ou quais atributos foram
131 * usados para modificação do valor em questão.
132 *
133 * No padrão Java Beans, cada atributo que queremos que seja exposto à
134 * alteração do cliente receberá um método set correspondente.
135 *
136 * O padrão para definição dos métodos set é:
137 *
138 * public void setNome( tipo parametro ),
139 * onde:
140 *     O nome do método inicia obrigatoriamente com o prefixo set,
141 *     seguido do nome do atributo que se quer alterar;
142 *     O método é público;
143 *     O método é void, pois a ideia é injetar um valor novo no objeto;
```

```
144     *      Há obrigatoriamente um parâmetro do mesmo tipo do atributo que
145     *      se quer alterar. O valor desse parâmetro será usado para alterar
146     *      o atributo do objeto, mudando assim seu estado.
147     */
148
149 /**
150 * Altera o dia da Data em questão.
151 *
152 * @param dia O novo valor do dia para a Data.
153 */
154 public void setDia( int dia ) {
155     this.dia = dia;
156 }
157
158 public void setMes( int mes ) {
159     this.mes = mes;
160 }
161
162 public void setAno( int ano ) {
163     this.ano = ano;
164 }
165
166 /*
167 * É nos métodos set que poderíamos talvez aplicar validações para os
168 * atributos que estão sendo definidos. No nosso exemplo da classe Data
169 * ficaremos na forma mais simples possível e deixaremos as validações
170 * para a classe Conta. No momento a ideia é fixar a ideia dos métodos
171 * get e set.
172 */
173
174 @Override
175 public String toString() {
176     return String.format( "%02d/%02d/%04d", dia, mes, ano );
177 }
178
179 }
```

Teste da classe Data

```
1  /*
2   * Arquivo: capitulo12/TesteData.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6  public class TesteData {
7
8      public static void main( String[] args ) {
9
10         Data data = new Data( 25, 2, 1985 );
11         System.out.println( "Data: " + data );
12
13         // alterando o dia da Data
14         data.setDia( 28 );
15         System.out.println( "dia alterado: " + data );
16
17         // alterando o mês da Data
18         data.setMes( 8 );
19         System.out.println( "mes alterado: " + data );
20
21         // alterando o ano da Data
22         data.setAno( 2019 );
23         System.out.println( "ano alterado: " + data );
24
25         // obtendo os atributos
26         System.out.println( "Atributos da Data:" );
27         System.out.println( "    dia: " + data.getDia() );
28         System.out.println( "    mes: " + data.getMes() );
29         System.out.println( "    ano: " + data.getAno() );
30
31     }
32
33 }
```

Aplicação do Encapsulamento na classe Pessoa

```
1  /*
2   * Arquivo: capitulo12/Pessoa.java
3   * Autor: Prof. Dr. David Buzatto
4   *
5   * Classe que representa uma Pessoa.
6   */
7  public class Pessoa {
8
9      private String nome;
10     private String sobrenome;
11     private double peso;
12
13     public Pessoa() {
14         nome = "";
15         sobrenome = "";
16         peso = 50;
17     }
18
19     public Pessoa( String nome, String sobrenome, double peso ) {
20         this.nome = nome;
21         this.sobrenome = sobrenome;
22         this.peso = peso;
23     }
24
25     public String getNome() {
26         return nome;
27     }
28
29     public String getSobrenome() {
30         return sobrenome;
31     }
32
33     public double getPeso() {
34         return peso;
35     }
36
37     public void setNome( String nome ) {
38         this.nome = nome;
39     }
40
41     public void setSobrenome( String sobrenome ) {
42         this.sobrenome = sobrenome;
43     }
44
45     public void setPeso( double peso ) {
46         this.peso = peso;
47     }
```

```
48  
49     @Override  
50     public String toString() {  
51  
52         String n = nome;  
53         String s = sobrenome;  
54  
55         if ( nome == null || nome.isEmpty() ) {  
56             n = "sem nome";  
57         }  
58  
59         if ( sobrenome == null || sobrenome.isEmpty() ) {  
60             s = "sem sobrenome";  
61         }  
62  
63         return String.format( "%s, %s: (%.2f)", s, n, peso );  
64     }  
65 }  
66  
67 }
```

Teste da classe Pessoa

```
1  /*
2   * Arquivo: capitulo12/TestePessoa.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5  public class TestePessoa {
6
7      public static void main( String[] args ) {
8
9          Pessoa pessoa = new Pessoa( "David", "Buzatto", 104.5 );
10         System.out.println( "Pessoa: " + pessoa );
11
12         // alterando o nome da Pessoa
13         pessoa.setNome( "Aurora" );
14         System.out.println( "nome alterado: " + pessoa );
15
16         // alterando o sobrenome da Pessoa
17         pessoa.setSobrenome( "Buzatto" );
18         System.out.println( "sobrenome alterado: " + pessoa );
19
20         // alterando o peso da Pessoa
21         pessoa.setPeso( 15.5 );
22         System.out.println( "peso alterado: " + pessoa );
23
24         // obtendo os atributos
25         System.out.println( "Atributos da Pessoa:" );
26         System.out.println( "    nome: " + pessoa.getNome() );
27         System.out.println( "    sobrenome: " + pessoa.getSobrenome() );
28         System.out.println( "    peso: " + pessoa.getPeso() );
29
30     }
31
32 }
```

Aplicação do Encapsulamento na classe Conta

```
1  /*
2   * Arquivo: capitulo12/Conta.java
3   * Autor: Prof. Dr. David Buzatto
4   *
5   * Classe que representa uma Conta.
6   */
7  public class Conta {
8
9      private String numero;
10     private double saldo;
11     private double limite;
12
13     public Conta() {
14         // usando sets para configurar os atributos
15         // compare com a versão do Capítulo anterior!
16         setNumero( "" );
17         setSaldo( 0 );
18         setLimite( 500 );
19     }
20
21     public Conta( String numero, double saldo, double limite ) {
22         // usando sets para configurar os atributos
23         // compare com a versão do Capítulo anterior!
24         setNumero( numero );
25         setSaldo( saldo );
26         setLimite( limite );
27     }
28
29     public boolean sacar( double valor ) {
30
31         if ( valor > 0 ) {
32
33             if ( saldo + limite - valor >= 0 ) {
34                 saldo -= valor;
35                 return true;
36             } else {
37                 return false;
38             }
39
40         }
41
42         return false;
43     }
44
45     public void depositar( double valor ) {
46
47 }
```

```
48     if ( valor > 0 ) {
49         saldo += valor;
50     }
51 }
52 }
53
54 public String getNumero() {
55     return numero;
56 }
57
58 public double getSaldo() {
59     return saldo;
60 }
61
62 public double getLimite() {
63     return limite;
64 }
65
66 /*
67 * Note que não existe um atributo chamado "situacao", mas podemos
68 * calcular essa propriedade/caractéristica do objeto usando um ou
69 * mais outros atributos. A situação da conta depende do seu saldo.
70 * Uma conta com saldo negativo é devedora, enquanto uma com saldo zero
71 * ou positivo é credora. Perceba também que o usuário da classe não
72 * precisa saber que não existe um atributo chamado situação, ele só
73 * deve estar interessado em usar o método getSituacao e que esse método
74 * funcione apropriadamente, ou seja, que reflita de fato o estado do
75 * objeto chamador.
76 */
77
78 /**
79 * Obtém a situação da Conta.
80 *
81 * @return Retorna "credora" caso o saldo seja zero ou positivo ou
82 * "devedora" caso o saldo seja negativo.
83 */
84 public String getSituacao() {
85     return saldo < 0 ? "devedora" : "credora";
86 }
87
88 public void setNumero( String numero ) {
89
90     // remove os espaços do início e do fim, caso existam
91     numero = numero.trim();
92
93     // se ficar vazio ou tiver uma quantidade de caracteres
94     // diferente de 5
95     if ( numero.isEmpty() || numero.length() != 5 ) {
```

```
97         // configura com o valor padrão "00001"
98         this.numero = "00001";
99
100        // se corresponde ao valor desejado, ou seja,
101        // um "número" com exatamente cinco dígitos, o usa
102        // para configurar a conta
103    } else {
104        this.numero = numero;
105    }
106}
107}
108
109 public void setSaldo( double saldo ) {
110
111        // validando o valor que foi passado
112    if ( saldo >= 0 ) {
113        this.saldo = saldo;
114    } else {
115        this.saldo = 0;
116    }
117}
118}
119
120 public void setLimite( double limite ) {
121
122        // validando o valor que foi passado
123    if ( limite >= 0 ) {
124        this.limite = limite;
125    } else {
126        this.limite = 0;
127    }
128}
129}
130
131 @Override
132 public String toString() {
133
134     String dados = String.format( "Conta: %s\n", numero );
135     dados += String.format( "Saldo: %sR$%.2f\n",
136             saldo < 0 ? "-" : "",
137             saldo < 0 ? -saldo : saldo );
138     dados += String.format( "Limite: R$%.2f\n", limite );
139     dados += "Situacao: " + getSituacao();
140
141     return dados;
142 }
143}
144}
```

Teste da classe Conta

```
1  /*
2   * Arquivo: capitulo12/TesteConta.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5  public class TesteConta {
6
7      public static void main( String[] args ) {
8
9          Conta conta = new Conta( "123", -1000, -500 );
10         System.out.println( conta + "\n" );
11
12         System.out.println( "Depositando R$500,00" );
13         conta.depositar( 500 );
14         System.out.println( conta + "\n" );
15
16         System.out.print( "Sacando R$1000,00: " );
17         if ( conta.sacar( 1000 ) ) {
18             System.out.println( "Saque realizado!" );
19         } else {
20             System.out.println( "Limite insuficiente!" );
21         }
22         System.out.println( conta + "\n" );
23
24         System.out.print( "Sacando R$2000,00: " );
25         if ( conta.sacar( 2000 ) ) {
26             System.out.println( "Saque realizado!" );
27         } else {
28             System.out.println( "Limite insuficiente!" );
29         }
30         System.out.println( conta + "\n" );
31
32         System.out.print( "Sacando R$200,00: " );
33         if ( conta.sacar( 200 ) ) {
34             System.out.println( "Saque realizado!" );
35         } else {
36             System.out.println( "Limite insuficiente!" );
37         }
38         System.out.println( conta + "\n" );
39
40         System.out.print( "Sacando R$400,00: " );
41         if ( conta.sacar( 400 ) ) {
42             System.out.println( "Saque realizado!" );
43         } else {
44             System.out.println( "Limite insuficiente!" );
45         }
46         System.out.println( conta + "\n" );
47     }
```

```
48     System.out.print( "Sacando R$300,00: " );
49     if ( conta.sacar( 300 ) ) {
50         System.out.println( "Saque realizado!" );
51     } else {
52         System.out.println( "Limite insuficiente!" );
53     }
54     System.out.println( conta + "\n" );
55
56     System.out.println( "Depositando R$1800,00" );
57     conta.depositar( 1800 );
58     System.out.println( conta );
59
60 }
61
62 }
```

12.2 Representação em UML

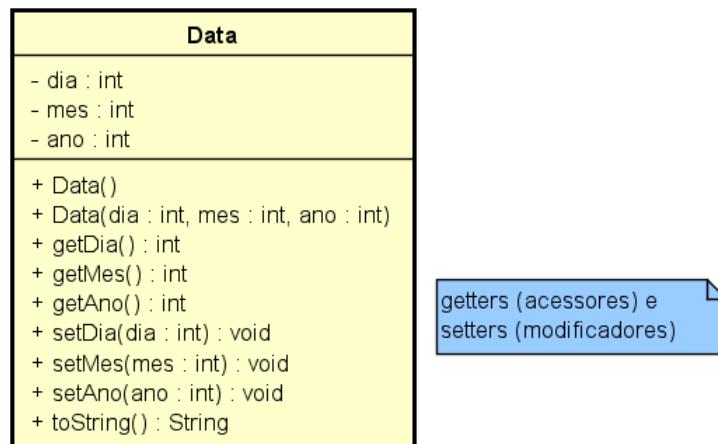


Figura 12.1: A classe Data em UML no padrão Java Beans

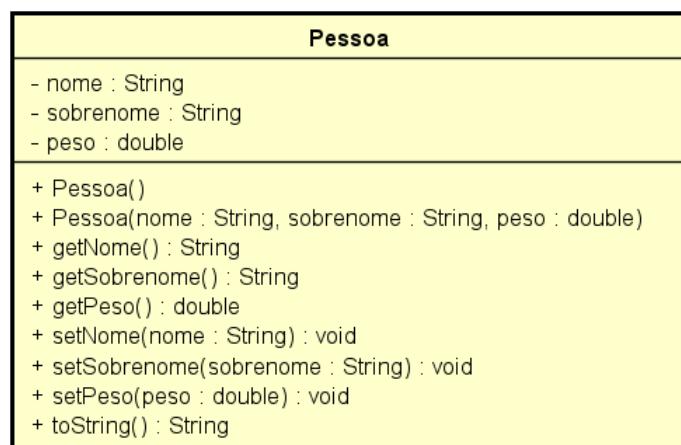


Figura 12.2: A classe Pessoa em UML no padrão Java Beans

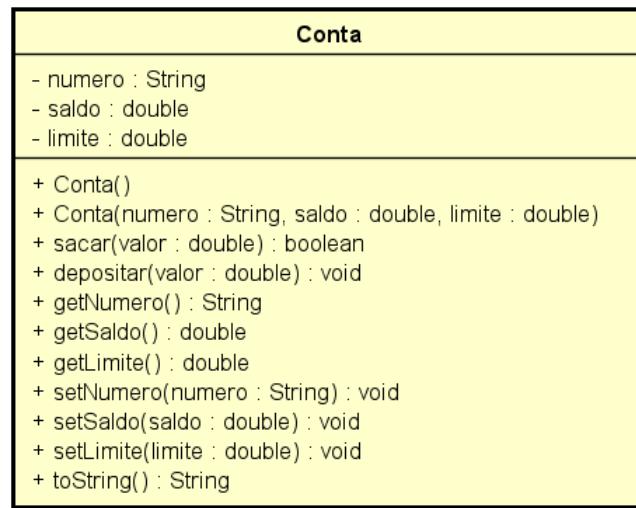


Figura 12.3: A classe Conta em UML no padrão Java Beans

Neste Capítulo não há exercícios para serem feitos! Vamos para o próximo!

COMPOSIÇÃO, AGREGAÇÃO E ASSOCIAÇÃO

“divide et impera”

“divide ut regnes”

“dividir para conquistar”



ESTE Capítulo exploraremos os principais tipos de associações entre classes que podem ser aplicados no Paradigma Orientado a Objetos. Basicamente, do ponto de vista das linguagens Orientadas a Objetos, todas essas associações são denominadas de Composição, onde a estrutura de uma classe pode ser composta por outras classes num relacionamento de Todo e Parte. No contexto da UML, essa composição entre objetos recebe diferentes tipos de nomes, dependendo da informação que se quer atribuir à esses relacionamentos.

13.1 Exemplos em Linguagem Java

Classe Data

```
1  /*
2  * Arquivo: capitulo13/Data.java
3  * Autor: Prof. Dr. David Buzatto
4  *
5  * Classe que representa uma Data.
6  */
7 public class Data {
8
9     private int dia;
10    private int mes;
11    private int ano;
12
13    public Data() {
14        setDia( 1 );
15        setMes( 1 );
16        setAno( 1970 );
17    }
18
19    public Data( int dia, int mes, int ano ) {
20        setDia( dia );
21        setMes( mes );
22        setAno( ano );
23    }
24
25    public int getDia() {
26        return dia;
27    }
28
29    public int getMes() {
30        return mes;
31    }
32
33    public int getAno() {
34        return ano;
35    }
36
37    public void setDia( int dia ) {
38        this.dia = dia;
39    }
40
41    public void setMes( int mes ) {
42        this.mes = mes;
43    }
44
45    public void setAno( int ano ) {
```

```
46         this.ano = ano;
47     }
48
49     @Override
50     public String toString() {
51         return String.format( "%02d/%02d/%04d", dia, mes, ano );
52     }
53
54 }
```

Classe Cliente

```
1  /*
2   * Arquivo: capitulo13/Cliente.java
3   * Autor: Prof. Dr. David Buzatto
4   *
5   * Classe que representa um Cliente.
6   */
7  public class Cliente {
8
9      /*
10     * Um cliente é composto por duas Strings, uma Data e um CPF.
11     *
12     * Um CPF é exclusivo do Cliente, ou seja, o CPF associado a um Cliente
13     * é dele e somente dele, pois não é compartilhado com nenhum outro
14     * Cliente. Se um Cliente deixa de existir, seu CPF também deixará.
15     *
16     * Sendo assim, essa associação é chamada de Composição e na UML é
17     * denotada por uma associação com um losango pintado.
18     */
19  private String nome;
20  private String sobrenome;
21  private Data dataNascimento;
22  private CPF documento;
23
24  public Cliente() {
25      setNome( "" );
26      setSobrenome( "" );
27      setDataNascimento( new Data( 1, 1, 1970 ) );
28      setDocumento( new CPF( "" ) );
29  }
30
31  public Cliente( String nome, String sobrenome,
32                  Data dataNascimento, CPF documento ) {
33
34      setNome( nome );
35      setSobrenome( sobrenome );
36      setDataNascimento( dataNascimento );
37      setDocumento( documento );
38
39  }
40
41  public String getNome() {
42      return nome;
43  }
44
45  public String getSobrenome() {
46      return sobrenome;
47  }
```

```
48
49     public Data getDataNascimento() {
50         return dataNascimento;
51     }
52
53     public CPF getDocumento() {
54         return documento;
55     }
56
57     public void setNome( String nome ) {
58         this.nome = nome;
59     }
60
61     public void setSobrenome( String sobrenome ) {
62         this.sobrenome = sobrenome;
63     }
64
65     public void setDataNascimento( Data dataNascimento ) {
66         this.dataNascimento = dataNascimento;
67     }
68
69     public void setDocumento( CPF documento ) {
70         this.documento = documento;
71     }
72
73     @Override
74     public String toString() {
75
76         String n = nome;
77         String s = sobrenome;
78
79         if ( nome == null || nome.isEmpty() ) {
80             n = "sem nome";
81         }
82
83         if ( sobrenome == null || sobrenome.isEmpty() ) {
84             s = "sem sobrenome";
85         }
86
87         return String.format( "%s, %s: (%s) | CPF: %s",
88                             dataNascimento, documento );
89     }
90 }
91
92 }
```

Classe Gerente

```
1  /*
2   * Arquivo: capitulo13/Gerente.java
3   * Autor: Prof. Dr. David Buzatto
4   *
5   * Classe que representa um Gerente.
6   */
7
8 import java.util.ArrayList;
9 import java.util.List;
10
11 public class Gerente {
12
13     /*
14      * Um Gerente é composto por três Strings, um CPF e uma lista de contas.
15      *
16      * O comportamento do CPF na classe Gerente é o mesmo definido na classe
17      * Cliente. Aqui poderíamos ter um problema. Por exemplo, caso um
18      * Gerente fosse também um Cliente.
19      *
20      * As contas que um determinado Gerente é responsável é representada no
21      * diagrama como uma associação chamada Agregação. Esse tipo de associa-
22      * ção é menos "forte" que a composição pois, caso um Gerente deixe de
23      * existir, as Contas a ele associada precisam continuar existindo,
24      * sendo atribuídas a outro ou outros gerentes.
25      */
26     private String matricula;
27     private String nome;
28     private String sobrenome;
29     private CPF documento;
30     private List<Conta> contas;
31
32     public Gerente() {
33         setMatricula( "" );
34         setNome( "" );
35         setSobrenome( "" );
36         setDocumento( new CPF( "" ) );
37         setContas( new ArrayList<>() );
38     }
39
40     public Gerente( String matricula, String nome,
41                     String sobrenome, CPF documento ) {
42
43         setMatricula( matricula );
44         setNome( nome );
45         setSobrenome( sobrenome );
46         setDocumento( documento );
47         setContas( new ArrayList<>() );
```

```
48
49 }
50
51     public String getMatricula() {
52         return matricula;
53     }
54
55     public String getNome() {
56         return nome;
57     }
58
59     public String getSobrenome() {
60         return sobrenome;
61     }
62
63     public CPF getDocumento() {
64         return documento;
65     }
66
67     public List<Conta> getContas() {
68         return contas;
69     }
70
71     public void setMatricula( String matricula ) {
72         this.matricula = matricula;
73     }
74
75     public void setNome( String nome ) {
76         this.nome = nome;
77     }
78
79     public void setSobrenome( String sobrenome ) {
80         this.sobrenome = sobrenome;
81     }
82
83     public void setDocumento( CPF documento ) {
84         this.documento = documento;
85     }
86
87     public void setContas( List<Conta> contas ) {
88         this.contas = contas;
89     }
90
91     @Override
92     public String toString() {
93
94         String n = nome;
95         String s = sobrenome;
```

```
97     if ( nome == null || nome.isEmpty() ) {
98         n = "sem nome";
99     }
100
101    if ( sobrenome == null || sobrenome.isEmpty() ) {
102        s = "sem sobrenome";
103    }
104
105    return String.format( "%s, %s: (%s) | CPF: %s", s, n,
106                          matricula, documento );
107
108}
109
110}
```

Classe CPF

```
1  /*
2   * Arquivo: capitulo13/CPF.java
3   * Autor: Prof. Dr. David Buzatto
4   *
5   * Classe que representa um CPF.
6   */
7  public class CPF {
8
9      private String numero;
10
11     public CPF() {
12         setNumero( "" );
13     }
14
15     public CPF( String numero ) {
16         setNumero( numero );
17     }
18
19     public String getNumero() {
20         return numero;
21     }
22
23     public void setNumero( String numero ) {
24         this.numero = numero;
25     }
26
27     @Override
28     public String toString() {
29         return numero;
30     }
31
32 }
```

Classe Conta

```
1  /*
2   * Arquivo: capitulo13/Conta.java
3   * Autor: Prof. Dr. David Buzatto
4   *
5   * Classe que representa uma Conta.
6   */
7  public class Conta {
8
9      /*
10     * Uma Conta é composta por uma String, dois doubles, uma Data, um
11     * Cliente e um Gerente.
12     *
13     * Um Cliente é exclusivo da Conta, ou seja, o Cliente associado a uma
14     * Conta é dela e somente dela, pois não é compartilhado com nenhuma
15     * outra Conta. Perceba que essa é a realidade para esse exemplo e a
16     * modelagem apresentada, mas poderíamos ter variações, como mais de
17     * um cliente por conta, a troca de cliente titular de uma conta etc.
18     *
19     * O caso do Gerente está representado no diagrama como uma associação
20     * navegável simples e seu outro lado está representado como uma outra
21     * associação, do Gerente para a Conta.
22     */
23     private String numero;
24     private double saldo;
25     private double limite;
26     private Data dataAbertura;
27     private Cliente titular;
28     private Gerente gerente;
29
30     public Conta() {
31         setNumero( "" );
32         setSaldo( 0 );
33         setLimite( 500 );
34         setDataAbertura( new Data( 1, 1, 2000 ) );
35         setTitular( null );
36         setGerente( null );
37     }
38
39     public Conta( String numero, double saldo, double limite,
40                 Data dataAbertura, Cliente titular,
41                 Gerente gerente ) {
42
43         setNumero( numero );
44         setSaldo( saldo );
45         setLimite( limite );
46         setDataAbertura( dataAbertura );
47         setTitular( titular );
```

```
48     setGerente( gerente );
49
50 }
51
52 public boolean sacar( double valor ) {
53
54     if ( valor > 0 ) {
55
56         if ( saldo + limite - valor >= 0 ) {
57             saldo -= valor;
58             return true;
59         } else {
60             return false;
61         }
62
63     }
64
65     return false;
66
67 }
68
69 public void depositar( double valor ) {
70
71     if ( valor > 0 ) {
72         saldo += valor;
73     }
74
75 }
76
77 public String getNumero() {
78     return numero;
79 }
80
81 public double getSaldo() {
82     return saldo;
83 }
84
85 public double getLimite() {
86     return limite;
87 }
88
89 public Data getDataAbertura() {
90     return dataAbertura;
91 }
92
93 public Cliente getTitular() {
94     return titular;
95 }
96
```

```
97     public Gerente getGerente() {
98         return gerente;
99     }
100
101    public String getSituacao() {
102        return saldo < 0 ? "devedora" : "credora";
103    }
104
105    public void setNumero( String numero ) {
106
107        numero = numero.trim();
108
109        if ( numero.isEmpty() || numero.length() != 5 ) {
110            this.numero = "00001";
111        } else {
112            this.numero = numero;
113        }
114    }
115
116
117    public void setSaldo( double saldo ) {
118
119        if ( saldo >= 0 ) {
120            this.saldo = saldo;
121        } else {
122            this.saldo = 0;
123        }
124    }
125
126
127    public void setLimite( double limite ) {
128
129        if ( limite >= 0 ) {
130            this.limite = limite;
131        } else {
132            this.limite = 0;
133        }
134    }
135
136
137    public void setDataAbertura( Data dataAbertura ) {
138        this.dataAbertura = dataAbertura;
139    }
140
141    public void setTitular( Cliente titular ) {
142        this.titular = titular;
143    }
144
145    public void setGerente( Gerente gerente ) {
```

```
146     this.gerente = gerente;
147 }
148
149 @Override
150 public String toString() {
151
152     String dados = String.format( "Conta: %s\n", numero );
153     dados += String.format( "Data de Abertura: %s\n", dataAbertura );
154     dados += String.format( "Titular: %s\n", titular );
155     dados += String.format( "Gerente: %s\n", gerente );
156     dados += String.format( "Saldo: %sR$%.2f\n",
157         saldo < 0 ? "-" : "",
158         saldo < 0 ? -saldo : saldo );
159     dados += String.format( "Limite: R$%.2f\n", limite );
160     dados += "Situacao: " + getSituacao();
161
162     return dados;
163 }
164 }
165 }
166 }
```

Teste das Associações

```
1  /*
2   * Arquivo: capitulo13/TesteAssociacoes.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5  public class TesteAssociacoes {
6
7      public static void main( String[] args ) {
8
9          Cliente cliente = new Cliente(
10             "João", "da Silva",
11             new Data( 20, 5, 1990 ),
12             new CPF( "123.123.123-12" ) );
13
14         Gerente gerente = new Gerente(
15             "6789", "Marcos", "Oliveira",
16             new CPF( "456.456.456-45" ) );
17
18         Conta conta = new Conta(
19             "88888", 5000, 2000,
20             new Data( 7, 7, 2023 ),
21             cliente, gerente );
22
23         gerente.getContas().add( conta );
24
25         System.out.println( "Cliente: " + cliente + "\n" );
26         System.out.println( "Gerente: " + gerente + "\n" );
27         System.out.println( conta );
28     }
29 }
30 }
31 }
```

13.2 Representação em UML

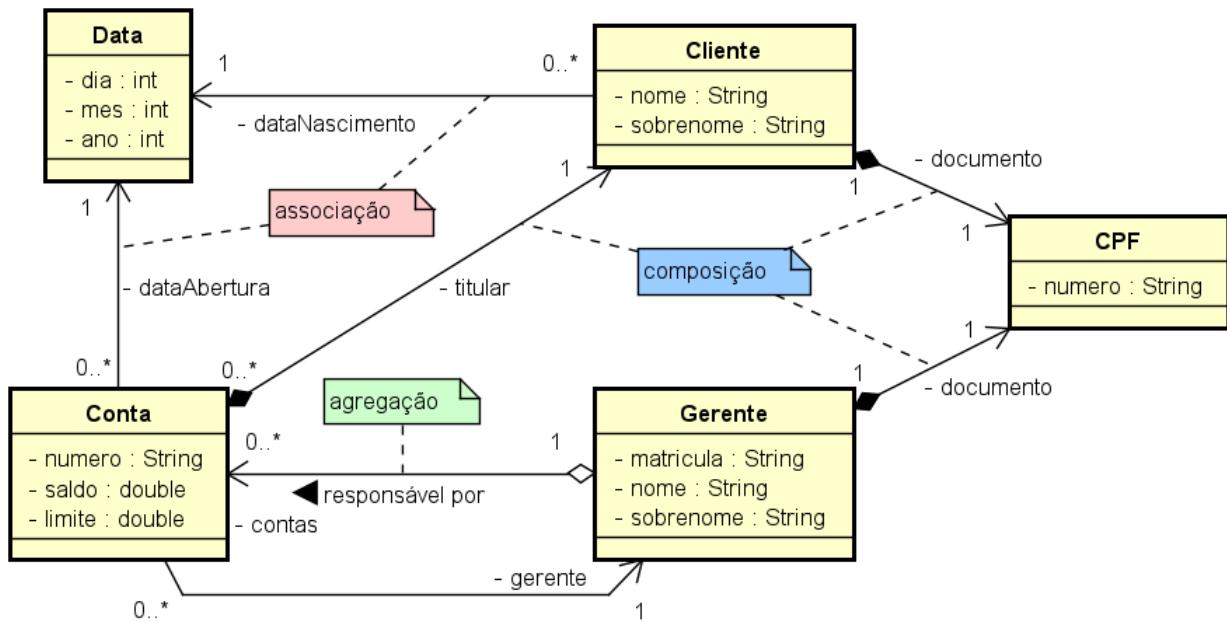


Figura 13.1: Representação de alguns dos diferentes tipos de Associações em UML

Vamos aos exercícios!

13.3 Exercícios

Exercício 13.1:

Projete uma classe chamada `Ponto` que possui os atributos `x` e `y`, ambos inteiros. Forneça um construtor padrão e um que inicialize cada um desses atributos. Lembre-se também de encapsulá-los. Sobrescreva o método `public String toString()` que deve gerar a representação em texto do ponto. Escreva uma segunda classe, chamada `TestePonto`, onde a classe `Ponto` será testada de modo a mostrar suas funcionalidades. Considere que seu programa só processará valores não negativos (maiores ou iguais à zero) e que o eixo `y` cresce para baixo, ou seja, o inverso do plano cartesiano tradicional. O formato da `String` retornada pelo método `toString()`, usando os dados do primeiro exemplo, é o seguinte:

| Exemplo

Formato `toString()`

```
1 (10; 20)
```

Diretório base: ex13\$1/

Arquivos com a solução:

- `Ponto.java`
- `TestePonto.java` `main`

Entrada

```
Ponto  
x: 10  
y: 20
```

Saída

```
Ponto criado: (10; 20)
```

Exercício 13.2:

Projete uma classe chamada `Linha` que possui os atributos `p1` e `p2`, ambos do tipo `Ponto` (exercício anterior). O atributo `p1` representa o ponto inicial da linha, enquanto o atributo `p2` representa o ponto final da linha. Forneça um construtor padrão e um que inicialize cada um desses atributos. Lembre-se também de encapsulá-los. Forneça o método `calcularComprimento`, apresentado abaixo:

- `public double calcularComprimento()`

Esse método deve retornar o valor comprimento da linha. Além desse método, sobrescreva o método `public String toString()` que deve gerar a representação em texto da linha. Escreva uma segunda classe, chamada `TesteLinha`, onde a classe `Linha` será testada de modo a mostrar suas funcionalidades. Considere que seu programa só processará valores não negativos (maiores ou iguais à zero) e que o eixo `y` cresce para baixo, ou seja, o inverso do plano cartesiano tradicional. O formato da `String` retornada pelo método `toString()`, usando os dados do primeiro exemplo, é o seguinte:

| Exemplo

Formato `toString()`

```
1 (10; 20) --- (20; 30)
```

Diretório base: ex13\$2/

Arquivos com a solução:

- `Linha.java`
- `Ponto.java`
- `TesteLinha.java` `main`

Entrada

Ponto inicial

x: 10
y: 20

Ponto final

x: 20
y: 30

Saída

Linha criada: (10; 20) --- (20; 30)

Comprimento: 14.14

Exercício 13.3:

Projete uma classe chamada `Retangulo` que possui os atributos `p1` e `p2`, ambos do tipo `Ponto`. O atributo `p1` representa o ponto superior esquerdo do retângulo, enquanto o atributo `p2` representa o ponto inferior direito do retângulo. Forneça um construtor padrão e um que inicialize cada um desses atributos. Lembre-se também de encapsulá-los. Forneça um método público chamado `double calcularArea()` que deve retornar o valor da área do retângulo. Além desse método, sobrescreva o método público `String toString()` que deve gerar a representação em texto do retângulo. Escreva uma segunda classe, chamada `TesteRetanguloEx3`, onde a classe `Retangulo` será testada de modo a mostrar suas funcionalidades. Considere que seu programa só processará valores não negativos (maiores ou iguais à zero) e que o eixo `y` cresce para baixo, ou seja, o inverso do plano cartesiano tradicional. Considere também que as coordenadas do ponto superior esquerdo sempre estarão à esquerda e acima das coordenadas do ponto inferior direito. O formato da `String` retornada pelo método `toString()`, usando os dados do primeiro exemplo, é o seguinte:

| Exemplo

Formato `toString()`

```

1 (10; 10)
2 ======
3 |
4 |
5 ======
6 (50; 30)

```

Diretório base: ex13\$3/

Arquivos com a solução:

- `Retangulo.java`
- `Ponto.java`
- `TesteRetanguloEx3.java` `main`

Entrada

Ponto superior esquerdo

x: 10

y: 10

Ponto inferior direito

x: 50

y: 30

Saída

Retangulo criado:

(10; 10)

|=====|

| |

| |

|=====|

(50; 30)

Area: 800.00

Exercício 13.4:

Com base no exercício anterior, adicione o método público `Ponto obterCentro()` à classe `Retangulo` que será responsável em retornar uma instância da classe `Ponto` com as coordenadas do ponto central do retângulo. Escreva a classe `TesteRetanguloEx4`, onde a classe `Retangulo` atualizada será testada de modo a mostrar suas funcionalidades. Considere que seu programa só processará valores não negativos (maiores ou iguais à zero) e que o eixo `y` cresce para baixo, ou seja, o inverso do plano cartesiano tradicional. Considere também que as coordenadas do ponto superior esquerdo sempre estarão à esquerda e acima das coordenadas do ponto inferior direito.

Diretório base: ex13\$4/

Arquivos com a solução:

- `Retangulo.java`
- `Ponto.java`
- `TesteRetanguloEx4.java` `main`

Entrada

```
Ponto superior esquerdo
  x: 10
  y: 10
Ponto inferior direito
  x: 50
  y: 30
```

Saída

```
Retangulo criado:
(10; 10)
+-----+
|           |
|           |
|           |
+-----+
(50; 30)
Centro: (30; 20)
```

Exercício 13.5:

Com base no exercício anterior, adicione o método público `mover`, apresentado abaixo, à classe `Retangulo`.

- `void mover(int quantidadeX, int quantidadeY)`

Esse método será responsável em mover o retângulo em uma quantidade de unidades em `x` e em `y`. Escreva a classe `TesteRetanguloEx5`, onde a classe `Retangulo` atualizada será testada de modo a mostrar suas funcionalidades. Considere que seu programa só processará valores não negativos (maiores ou iguais à zero) e que o eixo `y` cresce para baixo, ou seja, o inverso do plano cartesiano tradicional. Considere também que as coordenadas do ponto superior esquerdo sempre estarão à esquerda e acima das coordenadas do ponto inferior direito.

Diretório base: ex13\$5/

Arquivos com a solução:

- `Retangulo.java`
- `Ponto.java`
- `TesteRetanguloEx5.java` `main`

Entrada

```
Ponto superior esquerdo
  x: 10
  y: 10
Ponto inferior direito
  x: 50
  y: 30
Mover em
  x: 15
  y: 20
```

Saída

```
Retangulo criado:
(10; 10)
|=====
|       |
|       |
|=====|
(50; 30)
Retangulo movido:
(25; 30)
|=====
|       |
|       |
|=====|
(65; 50)
```

Exercício 13.6:

Com base no exercício anterior, adicione o método público `boolean contem(Ponto ponto)` à classe `Retangulo` que será responsável verificar se um ponto está contido no retângulo. Escreva a classe `TesteRetanguloEx6`, onde a classe `Retangulo` atualizada será testada de modo a mostrar suas funcionalidades. Considere que seu programa só processará valores não negativos (maiores ou iguais à zero) e que o eixo `y` cresce para baixo, ou seja, o inverso do plano cartesiano tradicional. Considere também que as coordenadas do ponto superior esquerdo sempre estarão à esquerda e acima das coordenadas do ponto inferior direito.

Diretório base: ex13\$6/

Arquivos com a solução:

- `Retangulo.java`
- `Ponto.java`
- `TesteRetanguloEx6.java` `main`

Entrada

```
Ponto superior esquerdo
  x: 10
  y: 10
Ponto inferior direito
  x: 50
  y: 30
Ponto
  x: 15
  y: 20
```

Saída

```
Retangulo criado:
(10; 10)
+-----+
|           |
|           |
|           |
+-----+
(50; 30)
Ponto criado: (15; 20)
O ponto esta contido no retangulo? Sim
```

Entrada

```
Ponto superior esquerdo
```

```
  x: 10
```

```
  y: 10
```

```
Ponto inferior direito
```

```
  x: 50
```

```
  y: 30
```

```
Ponto
```

```
  x: 5
```

```
  y: 5
```

Saída

```
Retangulo criado:
```

```
(10; 10)
```

```
|=====|
```

```
|       |
```

```
|       |
```

```
|=====|
```

```
(50; 30)
```

```
Ponto criado: (5; 5)
```

```
O ponto esta contido no retangulo? Nao
```

Exercício 13.7:

Com base no exercício anterior, adicione o método público `intersepta`, apresentado abaixo, à classe `Retangulo`.

- `boolean intersepta(Retangulo outroRetangulo)`

Esse método será responsável verificar se um outro retângulo intersepta o retângulo chamador. Escreva a classe `TesteRetanguloEx7`, onde a classe `Retangulo` atualizada será testada de modo a mostrar suas funcionalidades. Considere que seu programa só processará valores não negativos (maiores ou iguais à zero) e que o eixo `y` cresce para baixo, ou seja, o inverso do plano cartesiano tradicional. Considere também que as coordenadas do ponto superior esquerdo sempre estarão à esquerda e acima das coordenadas do ponto inferior direito.

Diretório base: ex13\$7/

Arquivos com a solução:

- `Retangulo.java`
- `Ponto.java`
- `TesteRetanguloEx7.java` `main`

Entrada

```
Retangulo 1
Ponto superior esquerdo
  x: 10
  y: 10
Ponto inferior direito
  x: 30
  y: 30
Retangulo 2
Ponto superior esquerdo
  x: 20
  y: 20
Ponto inferior direito
  x: 40
  y: 40
```

Saída

```
Retangulo 1:
(10; 10)
| ===== |
|       |
|       |
| ===== |
(30; 30)
Retangulo 2:
(20; 20)
| ===== |
|       |
|       |
| ===== |
(40; 40)
Os retangulos se interceptam? Sim
```

Entrada

```
Retangulo 1
Ponto superior esquerdo
  x: 10
  y: 10
Ponto inferior direito
  x: 30
  y: 30
Retangulo 2
Ponto superior esquerdo
  x: 50
  y: 50
Ponto inferior direito
  x: 60
  y: 60
```

Saída

```
Retangulo 1:
(10; 10)
| ===== |
|       |
|       |
| ===== |
(30; 30)
Retangulo 2:
(50; 50)
| ===== |
|       |
|       |
| ===== |
(60; 60)
Os retangulos se interceptam? Nao
```

HERANÇA E POLIMORFISMO

“It is the user who should parameterize procedures, not their creators”.

(PERLIS, 1982)



OS Capítulos anteriores exploramos dois conceitos importantes relacionados ao paradigma Orientado a Objetos, sendo eles o Encapsulamento e a Composição. Neste Capítulo exploraremos outros dois conceitos que são a Herança e o Polimorfismo. Esses quatro conceitos formam o alicerce do paradigma Orientado a Objetos. Vamos lá!

14.1 Exemplos em Linguagem Java

Classe abstrata Forma

```
1  /*
2  * Arquivo: capitulo14/Forma.java
3  * Autor: Prof. Dr. David Buzatto
4  *
5  * A classe forma é uma classe abstrada. Classes abstratas não podem ser
6  * instanciadas, ou seja, não é permitido criar objetos dessa classe de
7  * forma direta. Uma classe abstrata deve ser entendida como uma classe
8  * que normalmente será usada como um modelo para outras classes que
9  * herdarão dela.
10 *
11 * Normalmente, classes abstratas terão um ou mais métodos abstratos que
12 * deverão ser implementados por suas subclasses.
13 *
14 * Os conceitos de subclasses (classes filhas) e superclasses (classes mães)
15 * estão diretamente associados ao conceito de Herança, outro dos pilares
16 * do paradigma orientado a objetos. A ideia da Herança é criar tipos mais
17 * abstratos que servirão de modelo para tipos mais especializados.
18 *
19 * O processo de se estender (herdar) classes as associa em uma hierarquia.
20 * Na UML a representação da Herança é feita usando-se uma ligação da
21 * classe filha para a classe mãe, sendo que na extremidade da classe mãe
22 * há uma seta grande branca.
23 *
24 * Outro conceito que é associado à Herança, mas não somente à ela, é o
25 * Polimorfismo. A ideia do Polimorfismo se baseia em criar construções
26 * nas classes que têm comportamento diferentes dependendo do contexto.
27 * Quando estamos falando de Polimorfismo de Subtipo ou Polimorfismo de
28 * Inclusão, significa que métodos sobrescritos nas classes filhas serão
29 * escolhidos durante a vinculação dinâmica (vinculação tardia) na hora da
30 * execução dos mesmos, ou seja, dada uma variável que referencia um objeto,
31 * a versão do método que será invocada é a versão do objeto, não do tipo
32 * da variável.
33 *
34 * Outro ponto importante é que em Java não é permitido que haja a Herança
35 * múltipla, ou seja, uma classe herdar de mais de uma classe. Em Java
36 * só é permitida a herança simples, ou seja, só pode existir uma
37 * superclasse direta de uma outra classe. A motivação para isso é diminuir
38 * o problema de conflito de nomes que linguagens que permitem Herança
39 * Múltipla (C++ por exemplo) têm. Apesar de não se poder herdar de mais de
40 * uma classe diretamente, podemos incluir uma ou mais implementações de
41 * interfaces (veremos isso no próximo Capítulo).
42 */
43
44 import java.awt.Color;
45 import java.awt.Graphics2D;
```

```
46
47 public abstract class Forma {
48
49     /*
50      * Membros protected (protegidos) tem acesso direto nas subclasses
51      * da classe mãe.
52      */
53     protected int xIni;
54     protected int yIni;
55     protected int xFim;
56     protected int yFim;
57     protected Color corContorno = Color.BLACK;
58     protected Color corPreenchimento = Color.WHITE;
59
60     /*
61      * Um método abstrato precisa ser obrigatoriamente sobreescrito/
62      * implementado nas classes concretas que herdarem de Forma.
63      *
64      * Se uma classe possui pelo menos um método abstrato, ela precisa
65      * ser obrigatoriamente abstrata. Note que métodos abstratos não possuem
66      * implementação (terminam com ;).
67      */
68     public abstract void desenhar( Graphics2D g2d );
69
70     /*
71      * Classes abstratas podem ter métodos concretos (não abstratos) e
72      * inclusive construtores que serão chamados de forma indireta nas
73      * subclasses, permitindo que seus atributos sejam inicializados em
74      * um estado válido.
75      */
76     public int getXIni() {
77         return xIni;
78     }
79
80     public int getYIni() {
81         return yIni;
82     }
83
84     public int getXFim() {
85         return xFim;
86     }
87
88     public int getYFim() {
89         return yFim;
90     }
91
92     public Color getCorContorno() {
93         return corContorno;
94     }
```

```
95
96     public Color getCorPreenchimento() {
97         return corPreenchimento;
98     }
99
100    public void setXIni( int xIni ) {
101        this.xIni = xIni;
102    }
103
104    public void setYIni( int yIni ) {
105        this.yIni = yIni;
106    }
107
108    public void setXFim( int xFim ) {
109        this.xFim = xFim;
110    }
111
112    public void setYFim( int yFim ) {
113        this.yFim = yFim;
114    }
115
116    public void setCorContorno( Color corContorno ) {
117        this.corContorno = corContorno;
118    }
119
120    public void setCorPreenchimento( Color corPreenchimento ) {
121        this.corPreenchimento = corPreenchimento;
122    }
123
124 }
```

Classe Linha

```
1  /*
2   * Arquivo: capitulo14/Linha.java
3   * Autor: Prof. Dr. David Buzatto
4   *
5   * Classe que representa uma Forma do tipo Linha.
6   * A classe Linha herda de Forma e implementa o método abstrato desenhar.
7   */
8
9  import java.awt.Graphics2D;
10
11 public class Linha extends Forma {
12
13     /*
14      * Na implementação do método abstrato herdado a subclasse fornece
15      * seu comportamento especializado para esse método. Na implementação
16      * do método desenhar da classe Linha uma linha será desenhada usando
17      * o contexto gráfico Graphics2D (do Java2D) e os atributos definidos
18      * na superclasse.
19      */
20
21     public void desenhar( Graphics2D g2d ) {
22
23         // cria um novo contexto gráfico com base no passado no parâmetro
24         g2d = (Graphics2D) g2d.create();
25
26         // se há cor de preenchimento
27         if ( corContorno != null ) {
28
29             // configura a cor do "pincel" contexto
30             g2d.setColor( corContorno );
31
32             // desenha uma linha
33             g2d.drawLine( xIni, yIni, xFim, yFim );
34
35         }
36
37         // libera o contexto gráfico criado
38         g2d.dispose();
39
40     }
41 }
```

Classe Retangulo

```
1  /*
2   * Arquivo: capitulo14/Retangulo.java
3   * Autor: Prof. Dr. David Buzatto
4   *
5   * Classe que representa uma Forma do tipo Retângulo.
6   */
7
8  import java.awt.Graphics2D;
9
10 public class Retangulo extends Forma {
11
12     public void desenhar( Graphics2D g2d ) {
13
14         // cria um novo contexto gráfico com base no passado no parâmetro
15         g2d = (Graphics2D) g2d.create();
16
17         // define valores apropriados para o desenho do retângulo
18         int xIniD = xIni < xFim ? xIni : xFim;
19         int yIniD = yIni < yFim ? yIni : yFim;
20         int largura = Math.abs( xFim - xIni );
21         int altura = Math.abs( yFim - yIni );
22
23         // se há cor de preenchimento
24         if ( corPreenchimento != null ) {
25
26             // configura a cor do "pincel" contexto
27             g2d.setColor( corPreenchimento );
28
29             // pinta um retângulo
30             g2d.fillRect( xIniD, yIniD, largura, altura );
31         }
32
33         // se há cor de contorno
34         if ( corContorno != null ) {
35
36             // configura a cor do "pincel" contexto
37             g2d.setColor( corContorno );
38
39             // desenha o contorno de um retângulo
40             g2d.drawRect( xIniD, yIniD, largura, altura );
41
42         }
43
44         // libera o contexto gráfico criado
45         g2d.dispose();
46
47
```

```
48      }
49
50 }
```

Classe Elipse

```
1  /*
2   * Arquivo: capitulo14/Elipse.java
3   * Autor: Prof. Dr. David Buzatto
4   *
5   * Classe que representa uma Forma do tipo Elipse.
6   */
7
8 import java.awt.Graphics2D;
9
10 public class Elipse extends Forma {
11
12     public void desenhar( Graphics2D g2d ) {
13
14         // cria um novo contexto gráfico com base no passado no parâmetro
15         g2d = (Graphics2D) g2d.create();
16
17         // define valores apropriados para o desenho da elipse
18         int xIniD = xIni < xFim ? xIni : xFim;
19         int yIniD = yIni < yFim ? yIni : yFim;
20         int largura = Math.abs( xFim - xIni );
21         int altura = Math.abs( yFim - yIni );
22
23         // se há cor de preenchimento
24         if ( corPreenchimento != null ) {
25
26             // configura a cor do "pincel" contexto
27             g2d.setColor( corPreenchimento );
28
29             // pinta uma elipse
30             g2d.fillOval( xIniD, yIniD, largura, altura );
31         }
32
33         // se há cor de contorno
34         if ( corContorno != null ) {
35
36             // configura a cor do "pincel" contexto
37             g2d.setColor( corContorno );
38
39             // desenha o contorno de uma elipse
40             g2d.drawOval( xIniD, yIniD, largura, altura );
41
42         }
43
44         // libera o contexto gráfico criado
45         g2d.dispose();
46
47
```

```
48      }
49
50 }
```

Testes das Formas

```
1  /*
2   * Arquivo: capitulo14/Retangulo.java
3   * Autor: Prof. Dr. David Buzatto
4   *
5   * Teste das formas usando classes de interface gráfica.
6   *
7   * TesteFormas gerdade JFrame, que por sua vez é uma janela.
8   */
9
10 import java.awt.Color;
11 import java.awt.Graphics;
12 import java.awt.Graphics2D;
13 import java.awt.RenderingHints;
14 import java.util.ArrayList;
15 import java.util.List;
16 import javax.swing.JFrame;
17
18 public class TesteFormas extends JFrame {
19
20     // uma lista de formas
21     private List<Forma> formas;
22
23     public TesteFormas() {
24
25         // configura o JFrame
26         setTitle( "Teste Formas" );
27         setSize( 800, 600 );
28         setDefaultCloseOperation( EXIT_ON_CLOSE );
29         setLocationRelativeTo( null );
30
31         // cria a lista de formas
32         formas = new ArrayList<>();
33
34         // popula a lista
35         criarFormasTeste();
36
37     }
38
39     private void criarFormasTeste() {
40
41         // cria uma Linha e a configura
42         Linha linha = new Linha();
43         linha.setXIni( 50 );
44         linha.setYIni( 50 );
45         linha.setXFim( 160 );
46         linha.setYFim( 160 );
47 }
```

```
48     // cria um Retângulo e o configura
49     Retangulo retangulo = new Retangulo();
50     retangulo.setXIni( 200 );
51     retangulo.setYIni( 200 );
52     retangulo.setXFim( 400 );
53     retangulo.setYFim( 350 );
54     retangulo.setCorContorno( Color.GREEN );
55     retangulo.setCorPreenchimento( Color.YELLOW );
56
57     // cria uma elipse e a configura
58     Elipse elipse = new Elipse();
59     elipse.setXIni( 400 );
60     elipse.setYIni( 350 );
61     elipse.setXFim( 600 );
62     elipse.setYFim( 550 );
63     elipse.setCorContorno( Color.RED );
64     elipse.setCorPreenchimento( Color.BLUE );
65
66     // insere os objetos criados na lista de formas
67     formas.add( linha );
68     formas.add( retangulo );
69     formas.add( elipse );
70
71 }
72
73 /*
74 * O método paint é responsável em fazer o "desenho" da janela na
75 * tela. Esse desenho é feito usando o contexto gráfico Graphics
76 * que internamente é na verdade do subtipo Graphics2D.
77 */
78 @Override
79 public void paint( Graphics g ) {
80
81     /*
82     * A chamada ao paint da superclasse é obrigatório para que não
83     * haja inconsistências na apresentação do componente.
84     */
85     super.paint( g );
86
87     // cria um novo contexto gráfico com base no original
88     Graphics2D g2d = (Graphics2D) g.create();
89
90     // ativa suavização
91     g2d.setRenderingHint(
92         RenderingHints.KEY_ANTIALIASING,
93         RenderingHints.VALUE_ANTIALIAS_ON );
94
95     // percorre a lista de formas, desenhando uma a uma
96     for ( Forma f : formas ) {
```

```
97
98     /*
99      * Aqui que a vinculação dinâmica/tardia entra em ação
100     * para que o polimorfismo atue, pois a versão do método
101     * desenhar que será invocada dependerá do tipo do objeto,
102     * não da referência.
103     */
104    f.desenhar( g2d );
105
106 }
107
108 // libera o contexto gráfico criado
109 g2d.dispose();
110
111 }
112
113 public static void main( String[] args ) {
114     new TesteFormas().setVisible( true );
115 }
116
117 }
```

14.2 Representação em UML

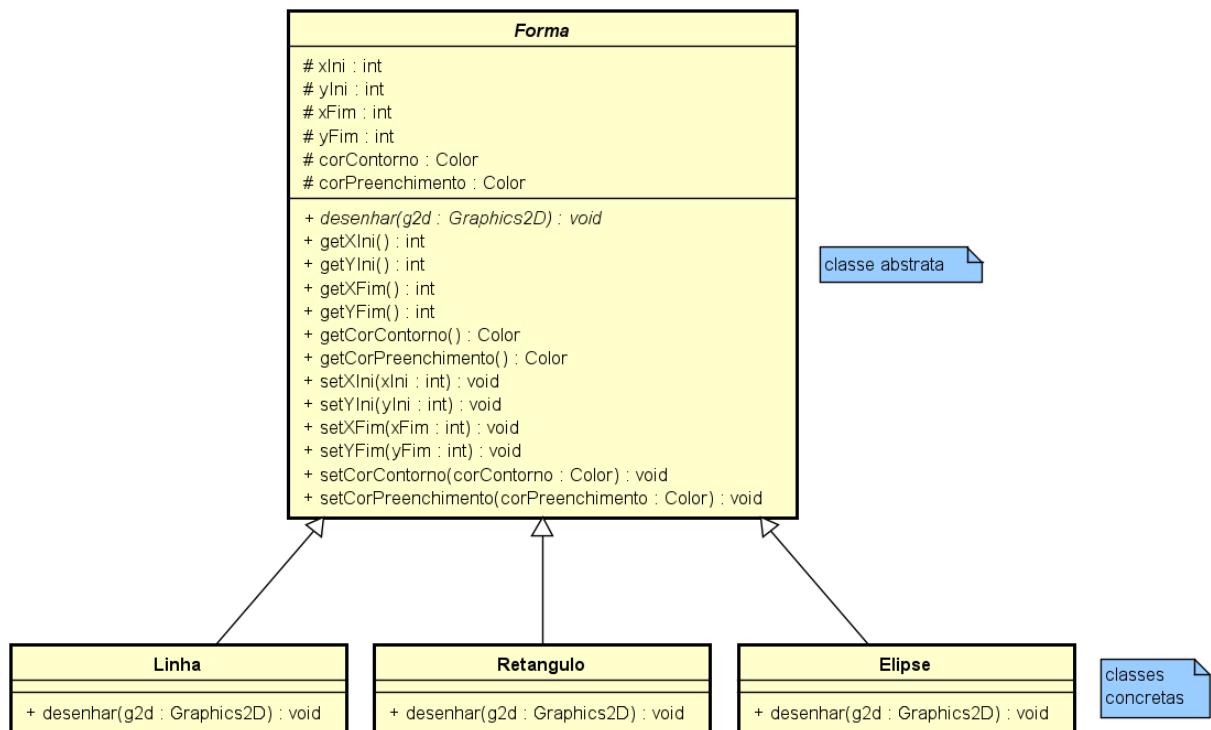


Figura 14.1: Representação de da Herança em UML

Neste Capítulo não há exercícios para serem feitos, pois implementaremos mais funcionalidades no programa de exemplo durante a aula!

INTERFACES

“Um contrato verbal não vale a tinta que é assinado.”

Samuel Goldwyn



S interfaces na linguagem Java devem ser enxergadas como contratos que devem ser seguidos. Há inúmeras interfaces prontas na própria plataforma para adicionarmos comportamentos às nossas classes, mas também podemos criar nossas próprias interfaces. Neste Capítulo iremos explorar essa funcionalidade da linguagem Java.

15.1 Exemplos em Linguagem Java

Classe Data Comparável

```
1  /*
2  * Arquivo: capitulo15/Data.java
3  * Autor: Prof. Dr. David Buzatto
4  *
5  * Classe que representa uma Data que é comparável.
6  *
7  * A classe Data implementa a interface Comparable.
8  *
9  * As interfaces representam "contratos" que devem ser seguidos pelas
10 * classes que as implementam. No caso da interface Comparable, o
11 * contrato é definido em como o método "compareTo" deve funcionar
12 * (veja abaixo).
13 *
14 * As classes que implementam Comparable e seguem o contrato têm a garantia
15 * que funcionarão apropriadamente em contextos que precisam que objetos
16 * dessa classe possam ser comparados, ou seja, se um objeto é "menor que"
17 * ("vem antes"), ou "maior que" ("vem depois") ou igual.
18 */
19 public class Data implements Comparable<Data> {
20
21     private int dia;
22     private int mes;
23     private int ano;
24
25     public Data() {
26         setDia( 1 );
27         setMes( 1 );
28         setAno( 1970 );
29     }
30
31     public Data( int dia, int mes, int ano ) {
32         setDia( dia );
33         setMes( mes );
34         setAno( ano );
35     }
36
37     public int getDia() {
38         return dia;
39     }
40
41     public int getMes() {
42         return mes;
43     }
44
45     public int getAno() {
```

```
46     return ano;
47 }
48
49     public void setDia( int dia ) {
50         this.dia = dia;
51     }
52
53     public void setMes( int mes ) {
54         this.mes = mes;
55     }
56
57     public void setAno( int ano ) {
58         this.ano = ano;
59     }
60
61 /**
62 * O método compareTo da interface Comparable retornará um valor
63 * inteiro.
64 *
65 * Esse valor representa se o objeto corrente é maior, menor ou igual
66 * ao objeto passado no parâmetro:
67 *
68 *      - Se o objeto corrente (o que invoca o método) for "menor que" o
69 *        objeto passado no parâmetro, o método deve retornar um valor
70 *        negativo;
71 *      - Se o objeto corrente (o que invoca o método) for "maior que" o
72 *        objeto passado no parâmetro, o método deve retornar um valor
73 *        positivo;
74 *      - Se o objeto corrente (o que invoca o método) for "igual ao"
75 *        objeto passado no parâmetro, o método deve retornar zero.
76 *
77 * O método compareTo, desde que implementado apropriadamente, definirá
78 * uma relação de ordem total para o tipo em questão, permitindo que
79 * esses objetos possam ser comparados e utilizados em métodos de
80 * ordenação, por exemplo.
81 */
82 @Override
83     public int compareTo( Data outra ) {
84
85         if ( this.ano < outra.ano ) {
86             return -1;
87         } else if ( this.ano > outra.ano ) {
88             return 1;
89         } else if ( this.mes < outra.mes ) {
90             return -1;
91         } else if ( this.mes > outra.mes ) {
92             return 1;
93         } else if ( this.dia < outra.dia ) {
94             return -1;
```

```
95     } else if ( this.dia > outra.dia ) {
96         return 1;
97     }
98
99     return 0;
100 }
101
102
103 @Override
104 public String toString() {
105     return String.format( "%02d/%02d/%04d", dia, mes, ano );
106 }
107
108 }
```

Testes da Data Comparável

```
1  /*
2   * Arquivo: capitulo15/TesteData.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5
6 import java.util.Arrays;
7
8 public class TesteData {
9
10    public static void main( String[] args ) {
11
12        Data d1 = new Data( 1, 2, 1990 );
13        Data d2 = new Data( 25, 2, 1985 );
14
15        System.out.println( "Comparando datas:" );
16
17        int comparacao = d1.compareTo( d2 );
18        if ( comparacao < 0 ) {
19            System.out.println( d1 + " < " + d2 );
20        } else if ( comparacao > 0 ) {
21            System.out.println( d1 + " > " + d2 );
22        } else { // comparacao == 0
23            System.out.println( d1 + " == " + d2 );
24        }
25        System.out.println();
26
27
28        System.out.println( "Ordenando datas:" );
29
30        Data[] datas = {
31            new Data( 2, 1, 1980 ),
32            new Data( 7, 7, 2023 ),
33            new Data( 18, 5, 1960 ),
34            new Data( 25, 7, 2015 )
35        };
36
37        System.out.println( "          Datas: " + Arrays.toString( datas ) );
38
39        // ordena
40        Arrays.sort( datas );
41
42        System.out.println( "Datas ordenadas: " + Arrays.toString( datas ) );
43
44    }
45
46 }
```

Interface Desenhavel

```
1  /*
2   * Arquivo: capitulo15/Desenhavel.java
3   * Autor: Prof. Dr. David Buzatto
4   *
5   * Uma interface que fornece aos seus implementadores um método
6   * de desenho.
7   *
8   * Já vimos que usualmente nomeamos nossas classes usando substantivos,
9   * pois representam coisas. As interfaces são normalmente nomeadas usando-se
10  * adjetivos, pois adicionarão características ou comportamentos às classes
11  * que as implementam.
12  *
13  * Toda classe que implementar a interface Desenhavel passará a ser uma
14  * classe cujos objetos podem ser desenhados ou que são desenháveis.
15  */
16
17 import java.awt.Graphics2D;
18
19 public interface Desenhavel {
20
21     public abstract void desenhar( Graphics2D g2d );
22
23 }
```

Classe Forma Desenhável

```
1  /*
2   * Arquivo: capitulo15/Forma.java
3   * Autor: Prof. Dr. David Buzatto
4   *
5   * Neste exemplo, separamos a ideia do desenho em uma interface chamada
6   * Desenhavel.
7   *
8   * Nessa interface é definido o método desenhar que a classe
9   * abstrata Forma deverá implementar ou deixar que as subclasses o
10  * implementem. Como forma é abstrata, ela não precisa implementar esse
11  * método e delegar essas responsabilidades às suas classes filhas.
12  *
13  * Caso uma classe seja concreta, ela obrigatoriamente deverá implementar
14  * o ou os métodos definidos nas interfaces implementadas.
15  *
16  * Na UML a representação de que uma classe implementa uma interface
17  * é feita por meio de uma realização, simbolizada de forma análoga à
18  * uma herança (linha com seta branca), só que a linha da seta é tracejada.
19 */
20
21 import java.awt.Color;
22
23 public abstract class Forma implements Desenhavel {
24
25     protected int xIni;
26     protected int yIni;
27     protected int xFim;
28     protected int yFim;
29     protected Color corContorno = Color.BLACK;
30     protected Color corPreenchimento = Color.WHITE;
31
32     public int getXIni() {
33         return xIni;
34     }
35
36     public int getYIni() {
37         return yIni;
38     }
39
40     public int getXFim() {
41         return xFim;
42     }
43
44     public int getYFim() {
45         return yFim;
46     }
47 }
```

```
48     public Color getCorContorno() {
49         return corContorno;
50     }
51
52     public Color getCorPreenchimento() {
53         return corPreenchimento;
54     }
55
56     public void setXIni( int xIni ) {
57         this.xIni = xIni;
58     }
59
60     public void setYIni( int yIni ) {
61         this.yIni = yIni;
62     }
63
64     public void setXFim( int xFim ) {
65         this.xFim = xFim;
66     }
67
68     public void setYFim( int yFim ) {
69         this.yFim = yFim;
70     }
71
72     public void setCorContorno( Color corContorno ) {
73         this.corContorno = corContorno;
74     }
75
76     public void setCorPreenchimento( Color corPreenchimento ) {
77         this.corPreenchimento = corPreenchimento;
78     }
79
80 }
```

15.2 Representação em UML

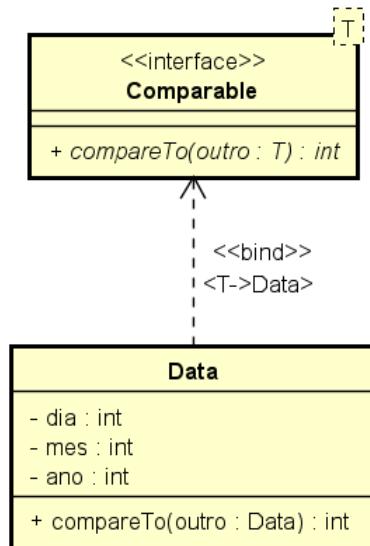


Figura 15.1: Representação da realização da interface Comparable pela classe Data

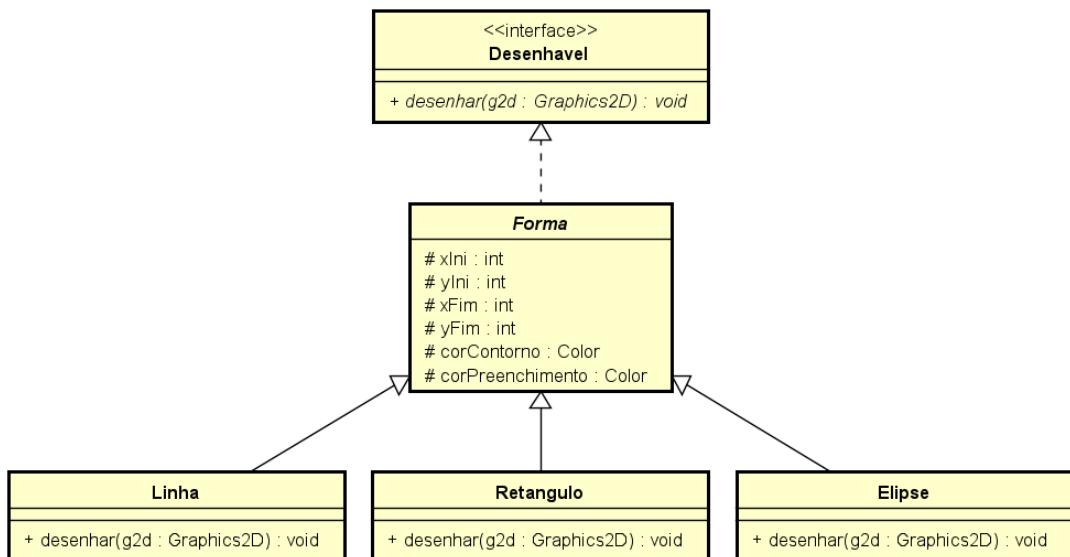


Figura 15.2: Representação da realização da interface Desenhavel pela classe Forma

ENUMERAÇÕES

“Educar verdadeiramente não é ensinar fatos novos ou enumerar fórmulas prontas, mas sim preparar a mente para pensar.”.

Albert Einstein



S enumerações servem para definir variáveis que terão um conjunto limitado de valores. Uma forma de alcançar tal comportamento seria a criação de uma série de constantes, mas as enumerações fornecem uma forma mais elegante de alcançar tal objetivo. Veja os exemplos a seguir.

16.1 Exemplos em Linguagem Java

Enumeração TipoUsuario

```
1  /*
2   * Arquivo: capitulo16/TipoUsuario.java
3   * Autor: Prof. Dr. David Buzatto
4   *
5   * Uma enumeração para o tipo de usuário.
6   *
7   * Uma enumeração define uma quantidade finita de "objetos" do tipo
8   * criado.
9   */
10 public enum TipoUsuario {
11
12     ADMINISTRADOR,
13     MODERADOR,
14     PADRAO;
15
16 }
```

Classe Usuario

```
1  /*
2  * Arquivo: capitulo16/Usuario.java
3  * Autor: Prof. Dr. David Buzatto
4  *
5  * Uma classe que representa um Usuário.
6  */
7  public class Usuario {
8
9      /*
10     * Um usuário é composto por duas Strings e uma enumeração.
11     */
12     private String email;
13     private String senha;
14     private TipoUsuario tipo;
15
16     public Usuario() {
17         setEmail( "aluno@ifsp.edu.br" );
18         setSenha( "123456" );
19         setTipo( TipoUsuario.PADRAO );
20     }
21
22     public Usuario( String email, String senha, TipoUsuario tipo ) {
23         setEmail( email );
24         setSenha( senha );
25         setTipo( tipo );
26     }
27
28     public String getEmail() {
29         return email;
30     }
31
32     public String getSenha() {
33         return senha;
34     }
35
36     public TipoUsuario getTipo() {
37         return tipo;
38     }
39
40     public void setEmail( String email ) {
41         this.email = email;
42     }
43
44     public void setSenha( String senha ) {
45         this.senha = senha;
46     }
47
```

```
48     public void setTipo( TipoUsuario tipo ) {  
49         this.tipo = tipo;  
50     }  
51  
52     @Override  
53     public String toString() {  
54         return email + " (" + tipo + ")";  
55     }  
56  
57 }
```

Teste da enumeração

```
1  /*
2   * Arquivo: capitulo16/TesteEnumeracao.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5  public class TesteEnumeracao {
6
7      public static void main( String[] args ) {
8
9          Usuario usuario = new Usuario(
10              "davidbuzatto@ifsp.edu.br", "12345", TipoUsuario.ADMINISTRADOR );
11
12         System.out.println( usuario );
13
14         if ( usuario.getTipo() == TipoUsuario.PADRAO ) {
15             System.out.println( "Usuário padrão (usando if)" );
16         }
17
18         switch ( usuario.getTipo() ) {
19
20             case ADMINISTRADOR:
21                 System.out.println( "Usuário administrador (usando switch)" );
22                 break;
23
24             case MODERADOR:
25                 System.out.println( "Usuário moderador (usando switch)" );
26                 break;
27
28             case PADRAO:
29                 System.out.println( "Usuário padrao (usando switch)" );
30                 break;
31
32         }
33
34     }
35
36 }
```

16.2 Representação em UML

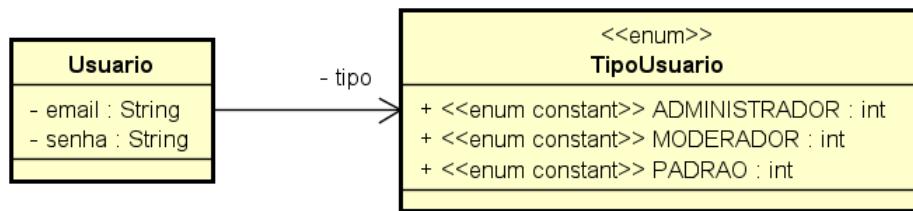


Figura 16.1: Representação de uma Enumeração na UML

RECORDS

“O cotidiano em si já é maravilhoso. Eu não faço nada mais do que registrá-lo”.

Franz Kafka



S *records*¹ (registros), nada mais são que um tipo especial de classe, sendo utilizadas apenas para manter dados que, por exemplo, serão transportados dentro da aplicação. Antes da existência dos mesmos, que foram introduzidos na versão 14 da linguagem Java como recurso experimental e adicionados, em sua forma final, na versão 16, quando havia a necessidade de se criar objetos imutáveis, o programador precisaria criar classes finais com uma série de restrições em seus atributos etc. Sendo assim, internamente, os *records* são classes imutáveis para dados e então, desde a versão 16 da linguagem Java, há esse "atalho" sintático para a criação de tal tipo de classe. O compilador é responsável em gerar uma série de construções de forma automática para nós:

- Métodos `equals`, `hashCode` e `toString`;
- Campos/atributos marcados automaticamente como `private` e `final`;
- Um construtor público para a inicialização de todos os campos;
- Métodos `get` que têm o mesmo nome dos atributos correspondentes.

Como analogia, os *records* podem ser enxergados como algo parecido com as *structs* (estruturas) da linguagem de programação C. Note, isso é apenas uma analogia, pois há diversas diferenças entre os *records* da linguagem Java e as *structs* da linguagem C. A ideia aqui é dizer que são construções que servem para representar TADs em que a preocupação está apenas nos dados e não nas operações. Em C++ a história também é diferente... Enfim, aqui o assunto é Java!

Vamos ao exemplo?

¹Neste livro, o termo será usado em inglês.

17.1 Exemplos em Linguagem Java

O record Pessoa

```
1  /*
2  * Arquivo: capitulo17/Pessoa.java
3  * Autor: Prof. Dr. David Buzatto
4  *
5  * Um record para Pessoas. Note que a sintaxe para a definição
6  * do record é um pouco diferente da que estamos acostumados.
7  *
8  * Basicamente trocamos a palavra chave class por record e,
9  * ao invés de criarmos os campos dentro da definição do
10 * record, nós os definimos seguindo o nome do record.
11 */
12 public record Pessoa( String nome, String sobrenome, int anoNascimento ) {
13
14     /*
15      * Como já informado, o compilador gerará os métodos equals,
16      * hashCode e toString para nós, além do construtor e de
17      * métodos get correspondentes aos atributos.
18      *
19      * O programador é livre para criar outros construtores, desde
20      * que nos construtores definidos todos os campos sejam inicializados,
21      * além de métodos e campos estáticos adicionais.
22      */
23
24 }
```

Teste do record Pessoa

```
1  /*
2   * Arquivo: capitulo17/TestePessoa.java
3   * Autor: Prof. Dr. David Buzatto
4   *
5   * Teste do record Pessoa.
6   */
7  public class TestePessoa {
8
9      public static void main( String[] args ) {
10
11         // instancia um record Pessoa (como um objeto comum)
12         Pessoa p1 = new Pessoa( "Joao", "da Silva", 1985 );
13
14         // instancia mais dois records do tipo Pessoa
15         Pessoa p2 = new Pessoa( "Joao", "da Silva", 1985 );
16         Pessoa p3 = new Pessoa( "Maria", "da Silva", 1990 );
17
18         // uma referencia chamada p4 para o record referenciado por p1
19         Pessoa p4 = p1;
20
21         // invoca o toString
22         System.out.println( p1 );
23
24         // get do nome
25         System.out.println( p1.nome() );
26
27         // get do sobrenome
28         System.out.println( p1.sobrenome() );
29
30         // get do ano de nascimento
31         System.out.println( p1.anoNascimento() );
32
33         System.out.println( "p1 igual a p2? " + p1.equals( p2 ) );
34         System.out.println( "p1 igual a p3? " + p1.equals( p3 ) );
35         System.out.println( "p1 igual a p4? " + p1.equals( p4 ) );
36         System.out.println( "p2 igual a p3? " + p2.equals( p3 ) );
37         System.out.println( "p1 == p2? " + ( p1 == p2 ) );
38         System.out.println( "p1 == p3? " + ( p1 == p3 ) );
39         System.out.println( "p1 == p4? " + ( p1 == p4 ) );
40         System.out.println( "p2 == p3? " + ( p2 == p3 ) );
41
42     }
43
44 }
```


Parte III

Tópicos Complementares

Java Collections Framework



JAVA Collections Framework (JCF) compreende a implementação padrão de diversos tipos de estruturas de dados básicas, como pilhas, filas, dequeus, listas, conjuntos e mapas. A implementação do JCF se encontra basicamente em dois pacotes: `java.util` e `java.util.concurrent`. Nos exemplos abaixo serão apresentadas as implementações contidas no pacote `java.util`, pois programação paralela/concorrente não será abordada neste livro. Em Java usa-se comumente o termo coleção para denotar estruturas de dados.

18.1 Pilhas, Filas e Deques

As pilhas, filas e as dequeus¹ (filas de fim duplo) são definidas na interface `Deque`², que fornece o contrato para a implementação dessas três estruturas de dados, pois uma deque é uma estrutura de dados linear que permite inserção e remoção de elementos em ambas extremidades. Exploraremos no exemplo a seguir duas implementações dessa interface: `ArrayDeque` e `LinkedList`. Vamos ao exemplo!

¹Double ended queue.

²Clique em mim para abrir a documentação ;)

Exemplo de Pilhas

```
1  /*
2   * Arquivo: capitulo18/ExemploPilhas.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5  import java.util.ArrayDeque;
6  import java.util.Deque;
7
8  public class ExemploPilhas {
9
10    public static void main( String[] args ) {
11
12      /*
13       * Uma deque que será usada como pilha. A implementação ArrayDeque
14       * é baseada em "redimensionamento" de arrays. Uma abordagem
15       * mais aprofundada das estruturas de dados não será dada
16       * neste livro.
17       */
18      Deque<Integer> pilha = new ArrayDeque<>();
19
20      /*
21       * void push( T elemento )
22       *
23       * Significado: empilhar.
24       *
25       * Insere um elemento no fim da deque ou no "topo da pilha".
26       *
27       * T é o tipo genérico utilizado para construir a deque.
28       */
29      pilha.push( 10 ); // empilha 10
30      pilha.push( 15 ); // empilha 15
31      pilha.push( 20 ); // empilha 20
32
33      // exibe a pilha (do topo à base)
34      System.out.println( pilha );
35
36      // itera pelos elementos da pilha (do topo à base)
37      for ( Integer elemento : pilha ) {
38        System.out.println( elemento );
39      }
40
41      /*
42       * T peek()
43       *
44       * Significado: "espiar"/consultar o topo.
45       *
46       * Consulta o elemento do fim da deque ou do "topo da pilha".
47       * Esse método não remove o elemento!
48     }
```

```
48      *
49      * T é o tipo genérico utilizado para construir a deque.
50      */
51     System.out.println( pilha.peek() ); // imprime 20
52
53     /*
54      * T pop()
55      *
56      * Significado: desempilhar.
57      *
58      * Remove o elemento do fim da deque ou do "topo da pilha".
59      *
60      * T é o tipo genérico utilizado para construir a deque.
61      */
62     System.out.println( pilha.pop() ); // imprime 20
63
64     // imprime [15, 10] (20 foi removido).
65     System.out.println( pilha );
66
67 }
68
69 }
```

Exemplo de Filas

```
1  /*
2   * Arquivo: capitulo18/ExemploFilas.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5  import java.util.ArrayDeque;
6  import java.util.Deque;
7
8  public class ExemploFilas {
9
10    public static void main( String[] args ) {
11
12      // Uma deque que será usada como fila, implementada como ArrayDeque.
13      Deque<Integer> fila = new ArrayDeque<>();
14
15      /*
16       * void offer( T elemento )
17       *
18       * Significado: enfileirar.
19       *
20       * Insere um elemento no fim da deque ou no "fim da fila".
21       *
22       * T é o tipo genérico utilizado para construir a deque.
23       */
24      fila.offer( 10 ); // enfileira 10
25      fila.offer( 15 ); // enfileira 15
26      fila.offer( 20 ); // enfileira 20
27
28      // exibe a fila (do início ao fim)
29      System.out.println( fila );
30
31      // itera pelos elementos da fila (do início ao fim)
32      for ( Integer elemento : fila ) {
33          System.out.println( elemento );
34      }
35
36      /*
37       * T peek()
38       *
39       * Significado: "espiar" o início/consultar o início.
40       *
41       * Consulta o primeiro elemento da deque ou do "início da fila".
42       * Esse método não remove o elemento!
43       *
44       * T é o tipo genérico utilizado para construir a deque.
45       */
46      System.out.println( fila.peek() ); // imprime 10
47
```

```
48     /*
49      * T poll()
50      *
51      * Significado: desenfileirar.
52      *
53      * Remove o primeiro elemento da deque ou do "início da fila".
54      *
55      * T é o tipo genérico utilizado para construir a deque.
56      */
57     System.out.println( fila.poll() ); // imprime 10
58
59     // imprime [15, 20] (10 foi removido)
60     System.out.println( fila );
61
62 }
63
64 }
```

Exemplo de Deques

```
1  /*
2   * Arquivo: capitulo18/ExemploDeques.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5  import java.util.Deque;
6  import java.util.LinkedList;
7
8  public class ExemploDeques {
9
10    public static void main( String[] args ) {
11
12      // Uma deque implementada como uma lista encadeada/ligada.
13      Deque<Integer> deque = new LinkedList<>();
14
15      /*
16       * void offer( T elemento )
17       * void offerLast( T elemento )
18       *
19       * Insere um elemento no fim da deque.
20       *
21       * T é o tipo genérico utilizado para construir a deque.
22       */
23      deque.offer( 10 );
24      System.out.println( deque );
25
26      deque.offerLast( 20 );
27      System.out.println( deque );
28
29      deque.offer( 30 );
30      System.out.println( deque );
31
32      /*
33       * void offerFirst( T elemento )
34       *
35       * Insere um elemento no inicio da deque.
36       *
37       * T é o tipo genérico utilizado para construir a deque.
38       */
39      deque.offerFirst( 40 );
40      System.out.println( deque );
41
42      // itera pelos elementos da deque (do inicio ao fim)
43      for ( Integer elemento : deque ) {
44          System.out.println( elemento );
45      }
46
47      /*

```

```
48     * T peek()
49     * T peekFirst()
50     *
51     * Consulta o elemento do início da deque.
52     * Esse método não remove o elemento!
53     *
54     * T é o tipo genérico utilizado para construir a deque.
55     */
56 System.out.println( deque.peek() );      // imprime 40
57 System.out.println( deque.peekFirst() ); // imprime 40
58
59 /**
60  * T peekLast()
61  *
62  * Consulta o elemento do fim da deque.
63  * Esse método não remove o elemento!
64  *
65  * T é o tipo genérico utilizado para construir a deque.
66  */
67 System.out.println( deque.peekLast() ); // imprime 30
68
69 System.out.println( deque );
70 /**
71  * T poll()
72  * T pollFirst()
73  *
74  * Remove o elemento do início da deque.
75  *
76  * T é o tipo genérico utilizado para construir a deque.
77  */
78 System.out.println( deque.poll() );      // imprime 40
79 System.out.println( deque.pollFirst() ); // imprime 10
80
81 /**
82  * T pollLast()
83  *
84  * Remove o elemento do fim da deque.
85  *
86  * T é o tipo genérico utilizado para construir a deque.
87  */
88 System.out.println( deque.pollLast() ); // imprime 30
89
90 }
91
92 }
```

Note que há vários métodos que não foram abordados nesses exemplos. Alguns desses métodos tem a mesma utilidade, mas se comportam de forma diferente. Por exemplo, `removeLast` e `pollLast` removem um elemento do fim da deque, entretanto, se a deque estiver vazia, `removeLast` lança

uma exceção, enquanto `pollLast` retorna `null`. Verifique a documentação para aprender mais! Caso não tenha percebido, os destaques das interfaces e classes são links para a documentação!

18.2 Listas

As listas são estruturas de dados lineares que permitem a inserção, alteração, consulta e remoção de elementos em qualquer parte da estrutura. No JCF a interface que define as operações básicas das listas é a `List`. Exploraremos no exemplo a seguir duas implementações dessa interface: `ArrayDeque` e `LinkedList`.

Exemplo de Listas

```
1  /*
2   * Arquivo: capitulo18/ExemploListas.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5  import java.util.ArrayList;
6  import java.util.LinkedList;
7  import java.util.List;
8
9 public class ExemploListas {
10
11     public static void main( String[] args ) {
12
13         /*
14          * Uma lista usando a implementação ArrayList, baseada em
15          * "redimensionamento" de arrays.
16          */
17         List<Integer> lista = new ArrayList<>();
18
19         /*
20          * Uma lista usando a implementação LinkedList, baseada em
21          * lista encadeada/ligada.
22          */
23         List<Integer> outraLista = new LinkedList<>();
24
25         /*
26          * void add( T elemento )
27          *
28          * Insere um elemento no fim da lista.
29          *
30          * T é o tipo genérico utilizado para construir a lista.
31          */
32         lista.add( 10 );
33         lista.add( 20 );
34         System.out.println( lista ); // imprime [10, 20]
35
36         outraLista.add( 30 );
37         outraLista.add( 40 );
38         System.out.println( outraLista ); // imprime [30, 40]
39
40         /*

```

```
41      * void addAll( Collection<? extends T> colecao )
42      *
43      * Insere todos os elementos da coleção passada como parâmetro
44      * na ordem que eles são iterados na coleção passada.
45      *
46      * T é o tipo genérico utilizado para construir a lista.
47      */
48 lista.addAll( outraLista );
49 System.out.println( lista ); // imprime [10, 20, 30, 40]
50
51 // itera pelos elementos da lista (do início ao fim)
52 for ( Integer elemento : lista ) {
53     System.out.println( elemento );
54 }
55
56 /*
57 * void add( int posicao, T elemento )
58 *
59 * Insere um elemento na posicao especificada.
60 *
61 * T é o tipo genérico utilizado para construir a lista.
62 */
63 lista.add( 3, 50 );
64 System.out.println( lista ); // imprime [10, 20, 30, 50, 40]
65
66 /*
67 * T get( int posicao )
68 *
69 * Obtém o elemento contido na posição especificada.
70 * Esse método não remove o elemento!
71 *
72 * T é o tipo genérico utilizado para construir a lista.
73 */
74 System.out.println( lista.get( 2 ) ); // imprime 30
75 System.out.println( lista ); // imprime [10, 20, 30, 50, 40]
76
77 /*
78 * T remove( int posicao )
79 *
80 * Remove o elemento contido na posição especificada.
81 *
82 * T é o tipo genérico utilizado para construir a lista.
83 */
84 System.out.println( lista.remove( 2 ) ); // imprime 30
85 System.out.println( lista ); // imprime [10, 20, 50, 40]
86
87 }
88 }
89 }
```

18.3 Conjuntos

O conjunto é uma estrutura de dados que têm como objetivo funcionar da mesma forma que um conjunto da matemática, ou seja, uma coleção de elementos não repetidos. A interface que define o contrato dos conjuntos é a `Set`. Exploraremos no exemplo a seguir três implementações dessa interface: `HashSet`, `TreeSet` e `LinkedHashSet`.

Algo importante em se levar em consideração é que os conjuntos precisam conseguir diferenciar os objetos que serão contidos por eles pois, afinal de contas, eles não devem conter objetos duplicados. A tarefa de executar essa diferenciação vem da implementação do método `equals`³, já discutido neste livro. Sendo assim, para que o conjunto funcione, a classe dos objetos que você inserirá no conjunto precisa do método `equals` implementado apropriadamente. Um outro detalhe importante é que os conjuntos são implementados basicamente de duas formas: ou como uma tabela de dispersão⁴ (`HashSet`) ou como uma árvore binária de busca balanceada⁵ (`TreeSet`).

Dependendo da implementação escolhida, é necessário que se implemente respectivamente ou o método `hashCode`⁶ ou a interface `Comparable`⁷, esta já discutida anteriormente. Isso se faz necessário, pois as tabelas de dispersão usarão esse código hash para conseguir espalhar os dados inseridos na estrutura, de modo a gerar a menor quantidade de colisões possível, e as árvores binárias de busca precisarão conseguir estabelecer a qual ordem natural⁸ dos elementos.

Exemplo de Conjuntos

```

1  /*
2   * Arquivo: capitulo18/ExemploConjuntos.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5  import java.util.HashSet;
6  import java.util.LinkedHashSet;
7  import java.util.Set;
8  import java.util.TreeSet;
9
10 public class ExemploConjuntos {
11
12     public static void main( String[] args ) {
13
14         // quatro strings para preencher os conjuntos
15         String[] nomes = { "David", "Aurora", "Fernanda", "Aurora" };
16
17         // Um conjunto implementado como uma tabela de dispersão
18         Set<String> conjunto1 = new HashSet<>();
19
20         // Um conjunto implementado como uma árvore binária de busca

```

³Para acessar a documentação do método `equals`, [clique aqui](#).

⁴Sinônimo de tabela hash, tabela de espalhamento ou tabela de escrutínio.

⁵Em Java, especificamente uma árvore vermelho e preto.

⁶Para acessar a documentação do método `hashCode`, [clique aqui](#).

⁷Para acessar a documentação da interface `Comparable`, [clique aqui](#).

⁸Relação de ordem total.

```
21     Set<String> conjunto2 = new TreeSet<>();  
22  
23     /*  
24      * Um conjunto implementado como uma tabela de dispersão, onde  
25      * a ordem de inserção dos elementos é mantida.  
26      */  
27     Set<String> conjunto3 = new LinkedHashSet<>();  
28  
29     /*  
30      * Insere as Strings no conjunto.  
31      *  
32      * Perceba que a String de conteúdo "Aurora" só será  
33      * inserida uma vez.  
34      */  
35     for ( String nome : nomes ) {  
36  
37         /*  
38          * void add( T elemento )  
39          *  
40          * Insere um elemento no conjunto.  
41          *  
42          * T é o tipo genérico utilizado para construir o conjunto.  
43          */  
44     conjunto1.add( nome );  
45     conjunto2.add( nome );  
46     conjunto3.add( nome );  
47 }  
48  
49 // imprime os conjuntos  
50  
51 // a ordem não é garantida  
52 System.out.println( conjunto1 );  
53  
54 // a ordem é garantida por causa da árvore  
55 System.out.println( conjunto2 );  
56  
57 // a ordem é garantida, mas baseada na ordem de inserção  
58 System.out.println( conjunto3 );  
59  
60 // iterando pelos elementos de um conjunto  
61 for ( String elemento : conjunto1 ) {  
62     System.out.println( elemento );  
63 }  
64  
65 }  
66 }  
67  
68 }
```

18.4 Mapas

Os mapas são estruturas de dados que permitem associar chaves com valores. A nomenclatura dessa estrutura de dados pode variar dependendo da linguagem de programação ou biblioteca utilizada, sendo que alguns nomes comuns são Tabelas de Símbolos, Dicionários ou Arrays Associativos. A interface que define o contrato dos mapas é a `Map`. Exploraremos no exemplo a seguir três implementações dessa interface: `HashMap`, `TreeMap` e `LinkedHashMap`.

Note que, assim como os conjuntos, as estruturas de dados subjacentes que permitem o funcionamento apropriado dos mapas são, novamente, as tabelas de dispersão e as árvores binárias de busca.

Exemplo de Mapas

```
1  /*
2   * Arquivo: capitulo18/ExemploMapas.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5 import java.util.HashMap;
6 import java.util.LinkedHashMap;
7 import java.util.Map;
8 import java.util TreeMap;
9
10 public class ExemploMapas {
11
12     public static void main( String[] args ) {
13
14         /*
15          * Nesse exemplo será implementado uma situação
16          * clássica de uso de mapas, principalmente do ponto
17          * de vista didático: Como contar a quantidade de cada
18          * caractere que aparece em um texto?
19          */
20         String texto = "o rato roeu a roupa do rei de roma";
21
22         // Um mapa implementado como uma tabela de dispersão
23         Map<Character, Integer> contador1 = new HashMap<>();
24
25         // Um mapa implementado como uma árvore binária de busca
26         Map<Character, Integer> contador2 = new TreeMap<>();
27
28         /*
29          * Um mapa implementado como uma tabela de dispersão, onde
30          * a ordem de inserção dos elementos é mantida.
31          */
32         Map<Character, Integer> contador3 = new LinkedHashMap<>();
33
34         // processa os mapas, fazendo a contagem dos caracteres os exibindo.
35         processar( contador1, texto );
```

```
36     processar( contador2, texto );
37     processar( contador3, texto );
38
39 }
40
41 /**
42 * Processa os mapas, contando os caracteres e os exibindo.
43 *
44 * @param contador O mapa que contém as contagens.
45 * @param texto O texto a ser processado.
46 */
47 private static void processar(
48     Map<Character, Integer> contador,
49     String texto ) {
50
51     // processa todos os caracteres do texto, ignorando os espaços
52     for ( char c : texto.toCharArray() ) {
53         if ( c != ' ' ) {
54             contar( contador, c );
55         }
56     }
57
58     // exibe o mapa
59     System.out.println( contador );
60
61     /*
62      * Set<K> keySet()
63      *
64      * Retorna um conjunto com todas as chaves do mapa. O tipo do
65      * conjunto depende do tipo do mapa.
66      *
67      * K é o tipo da chave (obrigatoriamente um Comparable) usado
68      * para construir o mapa.
69      */
70     System.out.println( contador.keySet() );
71
72     /*
73      * Collection<V> values()
74      *
75      * Retorna uma coleção com todos os valores do mapa. A ordem dos
76      * valores depende do tipo do mapa.
77      *
78      * V é o tipo do valor usado para construir o mapa.
79      */
80     System.out.println( contador.values() );
81
82     System.out.println( "Contém 'a'? " + contador.containsKey( 'a' ) );
83     System.out.println( "Contém 'b'? " + contador.containsKey( 'b' ) );
84
```

```
85     imprimir( contador );
86
87 }
88
89 /**
90 * Conta as ocorrências de um caractere.
91 *
92 * @param contador O mapa que conterá a contagem.
93 * @param c O caractere a ser contado.
94 */
95 private static void contar( Map<Character, Integer> contador, char c ) {
96
97     /*
98      * boolean containsKey( K chave )
99      *
100     * Verifica se existe uma determinada chave no mapa.
101     *
102     * K é o tipo da chave (obrigatoriamente um Comparable) usado
103     * para construir o mapa.
104     */
105    if ( !contador.containsKey( c ) ) {
106
107        /*
108         * void put( K chave, V valor )
109         *
110         * Associa uma chave a um valor no mapa. Caso a chave não
111         * exista, ela é inserida. Caso já exista, o valor anterior
112         * é sobrescrito.
113         *
114         * K é o tipo da chave (obrigatoriamente um Comparable) usado
115         * para construir o mapa.
116         * V é o tipo do valor usado para construir o mapa.
117         */
118        contador.put( c, 0 );
119
120    }
121
122    /*
123     * V get( K chave )
124     *
125     * Retorna o valor associado à chave. Caso a chave não exista,
126     * retorna null.
127     *
128     * K é o tipo da chave (obrigatoriamente um Comparable) usado
129     * para construir o mapa.
130     * V é o tipo do valor usado para construir o mapa.
131     */
132    contador.put( c, contador.get( c ) + 1 );
133}
```

```
134     }
135
136     /**
137      * Processa o mapa passado como parâmetro.
138      * A ordem dos elementos visitados dependerá da implementação.
139      *
140      * HashMap: ordem não garantida;
141      * TreeMap: ordem garantida pela árvore;
142      * LinkedHashMap: ordem de inserção.
143      *
144      * @param mapa O mapa a ser processado.
145      */
146     private static void imprimir( Map<Character, Integer> mapa ) {
147
148         /*
149          * Set<Map.Entry<K, V>> entrySet()
150          *
151          * Retorna um conjunto com todos as associações entre chaves
152          * e valores no mapa.
153          *
154          * K é o tipo da chave (obrigatoriamente um Comparable) usado
155          * para construir o mapa.
156          * V é o tipo do valor usado para construir o mapa.
157          */
158         for ( Map.Entry<Character, Integer> e : mapa.entrySet() ) {
159             System.out.printf( "%c -> %d\n", e.getKey(), e.getValue() );
160         }
161     }
162
163 }
164 }
```

Caso você queira explorar a documentação do JCF o link para a versão 17 da linguagem Java é esse aqui: <<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/doc-files/coll-overview.html>>.

EXPRESSÕES LAMBDA E STREAMS

“A função da oração não é influenciar Deus, mas especialmente mudar a natureza daquele que ora”.

Søren Kierkegaard



ESTE Capítulo exploraremos superficialmente duas funcionalidades adicionadas na versão 8 da linguagem Java que são as expressões lambda e os streams (fluxos). Em relação às expressões lambda, a ideia é ter uma sintaxe simplificada para a implementação de interfaces funcionais, ou seja, interfaces que contém apenas um método que será tratado como uma função. Já os streams, são uma forma de facilitar a forma com que o programador pode expressar consultas em estruturas de dados complexas. A seguir, você verá dois exemplos muito simples sobre esses dois assuntos. Recomendo que, para ter uma melhor noção de tais funcionalidades, você busque na Web sobre o assunto. Há diversas fontes de pesquisa com exemplos úteis e didáticos.

19.1 Exemplos em Linguagem Java

Exemplo de Expressões Lambda

```

1  /*
2   * Arquivo: capitulo19/ExemploLambdas.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5  import java.util.Arrays;
6  import java.util.Comparator;
7
8  public class ExemploLambdas {
9
10    // um record do tipo Pessoa para o exemplo
11    private record Pessoa( String nome, int idade ) {
12    }
13
14    public static void main( String[] args ) {
15
16        Pessoa[] pessoas = new Pessoa[]{
17            new Pessoa( "Joao", 35 ),
18            new Pessoa( "Mariana", 15 ),
19            new Pessoa( "Maria", 40 ),
20            new Pessoa( "Jose", 49 )
21        };
22
23        System.out.println( Arrays.toString( pessoas ) );
24
25        // ordena as pessoas pela idade, passando um comparator
26        Arrays.sort( pessoas, new Comparator<Pessoa>() {
27            public int compare( Pessoa p1, Pessoa p2 ) {
28                return p1.idade() - p2.idade();
29            }
30        });
31        System.out.println( Arrays.toString( pessoas ) ); // ordenado
32
33        embaralhar( pessoas );
34        System.out.println( Arrays.toString( pessoas ) ); // embaralhado
35
36        // ordena as pessoas pela idade, usando uma expressão lambda
37        Arrays.sort( pessoas, ( p1, p2 ) -> {
38            return p1.idade() - p2.idade();
39        });
40        System.out.println( Arrays.toString( pessoas ) ); // ordenado
41
42    }
43
44    private static void embaralhar( Pessoa[] pessoas ) {
45        for ( int i = 0; i < pessoas.length; i++ ) {

```

```
46         int p = (int) (Math.random() * pessoas.length);
47         Pessoa t = pessoas[p];
48         pessoas[p] = pessoas[i];
49         pessoas[i] = t;
50     }
51 }
52
53 }
```

Exemplo de Streams

```
1  /*
2   * Arquivo: capitulo19/ExemploStreams.java
3   * Autor: Prof. Dr. David Buzatto
4   */
5  import java.util.Arrays;
6  import java.util.List;
7  import java.util.stream.Collectors;
8
9 public class ExemploStreams {
10
11    // um record do tipo Pessoa para o exemplo
12    private record Pessoa( String nome, int idade ) {
13    }
14
15    public static void main( String[] args ) {
16
17        List<Pessoa> pessoas = Arrays.asList( new Pessoa[]{
18            new Pessoa( "Joao", 35 ),
19            new Pessoa( "Mariana", 15 ),
20            new Pessoa( "Maria", 40 ),
21            new Pessoa( "Jose", 49 )
22        });
23        System.out.println( pessoas );
24
25        // filtrando as pessoas menores de idade e coletando
26        // em uma nova lista
27        List<Pessoa> menoresDeIdade = pessoas.stream()
28            .filter( p -> p.idade < 18 )
29            .collect( Collectors.toList() );
30        System.out.println( menoresDeIdade );
31
32        // somando a idade de todas as pessoas
33        int somaIdades = pessoas.stream()
34            .mapToInt( p -> p.idade() )
35            .sum();
36        System.out.println( somaIdades );
37
38        // somando a idade de todas as pessoas com nomes
39        // iniciados em M
40        int somaIdadesM = pessoas.stream()
41            .filter( p -> p.nome().startsWith( "M" ) )
42            .mapToInt( p -> p.idade() )
43            .sum();
44        System.out.println( somaIdadesM );
45
46    }
47}
```

48 }

Parte IV

Finalizando...

CONCLUSÃO

“O que vale na vida não é o ponto de partida e sim a caminhada. Caminhando e semeando, no fim terás o que colher”.

Cora Coralina



SSIM finalizamos nosso livro! Vale a pena salientar que o que vimos durante o mesmo corresponde a uma pequena parte das funcionalidades disponíveis na linguagem de programação Java, sendo que, para que você possa esgotar o assunto, será necessário o estudo de outras fontes, além de anos de prática com a linguagem em um ambiente de desenvolvimento real.

Caso queira se especializar mais na linguagem, recomendo as obras:

- DEITEL, P. M.; DEITEL, H. M. **Java: como programar.** 10. ed. São Paulo: Pearson, 2017. 934 p.
- SIERRA, K. et al. **Use a Cabeça: Java.** 3. ed. Rio de Janeiro: Alta Books, 2024. 702 p.
- HORSTMANN, C. **Core Java: Fundamentals, Volume 1.** 13. ed. New York: Oracle Press, 2024. 840 p.
- HORSTMANN, C. **Core Java: Advanced Features, Volume 2.** 13. ed. New York: Oracle Press, 2024. 992 p.

Espero que este livro tenha sido útil!

Um grande abraço a todos!

Até mais!

BIBLIOGRAFIA

BEECROWD. 2025. Disponível em: <<https://www.beecrowd.com.br/>>. Acesso em: 06 de janeiro de 2025.

DEITEL, P. M.; DEITEL, H. M. **Java: como programar**. 10. ed. São Paulo: Pearson, 2017. 934 p.

HORSTMANN, C. **Core Java: Advanced Features, Volume 2**. 13. ed. New York: Oracle Press, 2024. 992 p.

HORSTMANN, C. **Core Java: Fundamentals, Volume 1**. 13. ed. New York: Oracle Press, 2024. 840 p.

PERLIS, A. J. Special feature: Epigrams on programming. **SIGPLAN Not.**, ACM, New York, NY, USA, v. 17, n. 9, p. 7–13, set. 1982. ISSN 0362-1340. Disponível em: <<http://doi.acm.org/10.1145/947955.1083808>>.

SIERRA, K. et al. **Use a Cabeça: Java**. 3. ed. Rio de Janeiro: Alta Books, 2024. 702 p.

ISBN: 978-65-01-28801-7



A standard linear barcode representing the ISBN number 978-65-01-28801-7.

9
786501
288017