Primeira Lista de Exercícios

Julho de 2013

Sumário

Introdução

Este documento contem a primeira lista de exercícios para a segunda edição do livro **Introdução à Programação Orientada a Objetos usando Java**. Cada capítulo deste documento corresponde a um capítulo do livro, frequentemente fazendo referência a figuras, listagens e exemplos apresentados no livro.

Os exercícios apresentados para cada capítulo são divididos em cinco categorias de dificuldade, marcadas com um número correspondente de estrelas. Esta lista contém um total de **330** exercícios.

Muitos exercícios parecem similares, o que é proposital: o estudante pode escolher quais dos exercícios aparentemente similares ele vai resolver, e deduzir a solução dos outros. Leitores que estudam em grupos podem também dividir tarefas e comparar resultados usando exercícios similares. A seleção ampla também facilita a um instrutor passar trabalhos, dividir tarefas, etc.

Outras listas de exercícios, projetos de programação e material sobre o livro podem ser encontrados em `http://www.elsevier.com.br/rafaelsantos`. Não existe uma lista de exercícios resolvidos: muitos aceitam mais de uma solução para implementação, e podem ser implementados para verificar se a resposta está de acordo com o esperado.

Capítulo 1

Introdução à programação orientada a objetos

Este capítulo apresenta o conceito de programa de computador e o conceito de modelo, que agrega atributos e operações pertinentes a um problema que se deseja representar e processar por programas de computadores. O capítulo apresenta alguns exemplos que ilustram como *encapsular* os atributos e operações de forma coerente, ou seja, contendo somente os atributos e operações realmente relevantes para o problema.

Exemplos de modelos são apresentados como diagramas UML simplificados e como pseudocódigo, com vários comentários que esclarecem a implementação do pseudocódigo e um exemplo simples de reuso de modelos. O capítulo também apresenta os conceitos básicos de orientação a objetos e discute a necessidade de seu uso.

1.1 Exercícios do Capítulo 1

Exercício 1.1 ★

Explique, com exemplos, por que seria complicado usar um "supermodelo" que representaria **todos** os atributos de uma pessoa.

Exercício 1.2 ★

Descreva, com suas próprias palavras, a operação `calculaConta` do modelo que representa o Restaurante Caseiro Hipotético.

Exercício 1.3 ★

Imagine que o Restaurante Caseiro Hipotético facilite aos seus clientes a divisão do valor total da conta pelo número de clientes. Que atributos adicionais deveriam ser representados pelo modelo? Quais operações deveriam ser criadas e/ou modificadas?

Exercício 1.4 ★

Escreva um modelo para representar uma lâmpada que está à venda em um supermercado. Que atributos devem ser representados por este modelo?

Exercício 1.5 ★

Imagine uma lâmpada que possa ter três estados: apagada, acesa e meia-luz. Usando o modelo `Lampada` (listagem 1.1 no livro) como base, escreva o modelo `LampadaTresEstados`.

Exercício 1.6 ★

Generalize o modelo `LampadaTresEstados` (exercício 1.5) para que ele possa representar uma lâmpada onde a luminosidade pode ser ajustada com qualquer valor entre 0% (apagada) e 100% (acesa). *Dica:* em vez de operações para possibilitar o ajuste para cada um dos estados, descreva uma operação que receba um valor de ajuste.

Exercício 1.7 ★

Inclua, no modelo `Lampada` (listagem 1.1 no livro), uma operação `estáLigada` que retorne verdadeiro se a lâmpada estiver ligada e falso, caso contrário.

Exercício 1.8 ★

A operação `abreConta` do modelo `ContaBancariaSimplificada` (Listagem 1.2 no livro) permite que alguém crie uma conta bancária passando como argumento um valor negativo, criando uma conta já em débito. Modifique a operação `abreConta` para que, se alguém passar um saldo inicial negativo, que este seja considerado como zero.

Exercício 1.9 ★

Modifique a operação `mostraDados` do modelo `ContaBancariaSimplificada` (Listagem 1.2 no livro) para que, caso o saldo esteja negativo, uma mensagem de alerta seja impressa. *Dica:* O saldo só poderá ser negativo se a conta for especial.

Exercício 1.10 ★

Baseado no modelo `Data` (Listagem 1.3 no livro) crie o modelo `HoraAproximada`, que represente uma hora qualquer (usando valores para representar horas e minutos). Que atributos e operações este modelo deve ter?

Exercício 1.11 ★

Baseado no modelo `Data` (Listagem 1.3 no livro) e considerando o exercício 1.10, crie o modelo `HoraPrecisa`, que represente uma hora qualquer (usando valores para representar horas, minutos, segundos e centésimos de segundos). Que atributos e operações este modelo deve ter? Que atributos e operações poderiam ser copiados do modelo `HoraAproximada`, do exercício 1.10?

Exercício 1.12 ★

Crie um modelo `DataHora` que represente simultaneamente uma data e uma hora aproximada. *Dica:* O modelo pode conter instâncias dos modelo `Data` e `HoraAproximada`.

Exercício 1.13 ★

O modelo `Data` (Listagem 1.3 no livro) pode conter datas não-válidas, com os valores de dia, mês e ano iguais a zero, que podem ser criadas quando a operação `inicializaData` for chamado com valores incorretos. Modifique a operação `mostraData` para que, se o dia, mês ou ano forem inválidos (isto é, iguais a zero), uma mensagem "Data Inválida" seja impressa em vez dos valores de dia, mês e ano.

Exercício 1.14 ★

Crie um modelo para representar um professor de uma disciplina qualquer. *Dica:* Use, para orientação, o modelo `RegistroAcademico` (Listagem 1.4 no livro).

Exercício 1.15 ★

Crie um modelo para representar um time de um esporte qualquer em um campeonato desse esporte. Que atributos e operações esse modelo deve ter?

Exercício 1.16 ★

Escreva um modelo `Empregado` que represente um empregado de uma empresa qualquer. Considere que os atributos `nome`, `departamento`, `horasTrabalhadasNoMês` e `salárioPorHora` devam ser representados, e que ao menos as operações `mostraDados` e `calculaSalárioMensal` sejam implementadas.

Exercício 1.17 ★

Crie um modelo `Musica` para representar uma música, para uso em uma coleção ou banco de dados de músicas. Que atributos e operações esse modelo deve ter?

Exercício 1.18 ★

Crie um modelo `Ponto2D` para representar um ponto no espaço cartesiano de duas dimensões. Que dados e operações esse modelo deve ter? *Dica:* Imagine um gráfico no qual você tenha que desenhar pontos, baseados nesse modelo.

Exercício 1.19 ★

Crie um modelo `Livro` que represente os atributos básicos de um livro, sem se preocupar com a sua finalidade.

Exercício 1.20 ★

Usando o resultado do exercício 1.19 como base, crie um modelo `LivroDeLivraria` que represente os atributos básicos de um livro que está à venda em uma livraria. Veja também o exercício 1.21.

Exercício 1.21 ★

Usando o resultado do exercício 1.19 como base, crie um modelo `LivroDeBiblioteca` que represente os atributos básicos de um livro de uma biblioteca, que pode ser emprestado a leitores. Veja também o exercício 1.20.

Exercício 1.22 ★

Usando o resultado do exercício 1.19 como base, crie um modelo `DicionarioBilingue` que represente os atributos básicos de um dicionário de línguas (por exemplo, português-inglês, latim-aramaico etc.).

Exercício 1.23 ★ ★

Crie um modelo para representar um retângulo, cujos pontos opostos sejam instâncias do modelo `Ponto2D` (exercício 1.18). Veja também o exercício 1.30.

Exercício 1.24 ★ ★

Escreva um modelo que represente um polígono regular de até dez lados. Que atributos e operações este modelo deve conter? Descreva, para esse modelo, uma operação que retorne o nome do polígono baseado no seu número de lados.

Exercício 1.25 ★ ★

Imagine que o Restaurante Caseiro Hipotético deva ser representado para fins de inclusão em guias turísticos. Quais atributos e operações devem ser representados pelo modelo?

Exercício 1.26 ★ ★

Imagine que os empregados de uma empresa tenham dois valores de salário para horas trabalhadas, diferenciados entre horas normais e horas extras. Modifique o modelo `Empregado` (veja o exercício 1.16) para que dois valores de horas trabalhadas e dois valores de salário-hora sejam usados.

Exercício 1.27 ★ ★

Modifique a operação `calculaSalárioMensal` no modelo `Empregado` (veja o exercício 1.16) para que todos os empregados do departamento Diretoria tenham 10% de bônus salarial.

Exercício 1.28 ★ ★

Imagine que o Restaurante Caseiro Hipotético deva ser representado para fins de cálculo de impostos. Quais atributos e ações devem ser representados pelo modelo?

Exercício 1.29 ★ ★

Modifique a operação `mostraData` no modelo `Data` (Listagem 1.3 no livro) para que o mês seja mostrado por extenso. *Dica:* Veja o exercício 1.13.

Exercício 1.30 ★ ★

Crie um modelo para representar uma linha, unida por dois pontos no espaço cartesiano de duas dimensões, usando o modelo criado no exercício 1.18. Que atributos e operações esse modelo deve ter?

Exercício 1.31 ★ ★ ★

Considere o modelo `Lampada` mostrado em pseudocódigo na listagem 1.1 no livro. Imagine que uma lâmpada representada por esse modelo possa ter um outro dado, `queimada`, além do dado `estado`. Que operações deveriam ser modificados no modelo `Lampada`? Que outras operações deveriam ser adicionadas?

Exercício 1.32 ★ ★ ★

Crie um modelo `CDDeMusicas` que contenha várias instâncias do modelo `Musica` (exercício 1.17). Como você acha que podemos fazer para representar, em um `CDDeMusicas`, um número variável de instâncias de `Musica`?

Exercício 1.33 ★ ★ ★

Crie um modelo `EquacaoSegundoGrau` que contenha somente uma operação, a que calcula as raízes da equação. Considere que os valores de a, b e c serão passados para uma operação desse modelo. Qual a complexidade adicional de se criar esse modelo, quando comparado com um algoritmo simples? Quais as vantagens esperadas?

Exercício 1.34 ★ ★ ★ ★

A operação `inicializaData` do modelo `Data` (Listagem 1.3 no livro) tem uma abordagem simplista demais para verificar se o dia sendo usado é válido ou não: nessa operação ainda seria possível passar a data 31/02/2000 e a operação iria considerar os valores passados como sendo válidos. Modifique a operação `dataÉVálida` para que esta considere o valor máximo que pode ser aceito como válido, dependendo do mês, de forma que, para meses com 30 dias, o valor 31 para o dia seja considerado incorreto, e que para fevereiro o valor máximo seja calculado em função de o ano ser bissexto ou não. *Dica:* Anos bissextos (tendo 29 dias em fevereiro) são divisíveis por quatro, a não ser que sejam divisíveis por 100. Anos que podem ser divididos por 400 também são bissextos. Dessa forma, 1964 e 2000 são bissextos, mas 1900 não é bissexto. A operação de divisibilidade pode ser implementada pela função módulo, representada pelo sinal `%`, e comparada com zero: a expressão `(1966 % 4) == 0` é verdadeira, enquanto a expressão `(1967 % 4) == 0` é falsa.

Capítulo 2

Criando classes em Java

Neste capítulo vemos como modelos, descritos e exemplificados no capítulo 2, podem ser implementados na linguagem Java. Vemos também as regras básicas de sintaxe da linguagem, nomes válidos e não válidos para classes, atributos e métodos, conceitos de pacotes e classes e como comentar código para facilitar a sua leitura por outros programadores, assim como declarar atributos e métodos em classes em Java.

Neste capítulo também são apresentados os conceitos de escopo e de modificadores de acesso para atributos e métodos em classes, que permitem o encapsulamento efetivo destes atributos e métodos. É feita uma breve apresentação dos diferentes modificadores de acesso e seus efeitos.

2.1 Exercícios do Capítulo 2

Exercício 2.1 ★

Quais dos identificadores abaixo podem ser usados como nomes de classes, atributos, métodos e variáveis em Java? Quais não podem, e por quê?

A. `four`
B. `for`
C. `from`
D. `4`
E. `FOR`

Exercício 2.2 ★

Quais dos identificadores abaixo podem ser usados como nomes de classes, atributos, métodos e variáveis em Java? Quais não podem, e por quê?

A. `dia&noite`
B. `diaENoite`
C. `dia & noite`
D. `dia E noite`
E. `dia_e_noite`

Exercício 2.3 ★

Quais dos identificadores abaixo podem ser usados como nomes de classes, atributos, métodos e variáveis em Java? Quais não podem, e por quê?

A. `contador`
B. `1contador`
C. `contador de linhas`
D. `Contador`
E. `count`

Exercício 2.4 ★

Considerando a tabela 2.2 no livro, escolha o tipo de dado ou classe mais adequada para representar:

- O número de municípios de um estado do Brasil.
- O nome de um estado do Brasil.
- A população de um estado do Brasil.
- A área do Brasil em quilômetros quadrados.
- A população total do mundo.
- O CEP de um endereço no Brasil.
- O nome de uma rua em um endereço no Brasil.

Exercício 2.5 ★

Considerando a tabela 2.2, escolha o tipo de dado ou classe mais adequada para representar:

- A altura de uma pessoa em metros.
- O peso de uma pessoa em quilos.
- A temperatura corporal de uma pessoa.
- O sexo de uma pessoa.
- A altura de uma pessoa em milímetros.

Exercício 2.6 ★

Responda verdadeiro ou falso para cada uma das afirmações abaixo, explicando ou justificando a sua resposta.

A. Um valor do tipo `boolean` pode receber o valor numérico zero.
B. Um valor do tipo `float` pode armazenar valores maiores do que os que podem ser armazenados por um valor do tipo `long`.
C. Podemos ter caracteres cujos valores sejam negativos.
D. O número de bytes ocupados por uma variável do tipo `float` depende do computador e do sistema operacional sendo usado.
E. O tipo `char` pode ser usado para representar pares de caracteres, uma vez que variáveis desse tipo ocupam dois bytes na memória.
F. Os tipos de dados `double` e `long` não são equivalentes, apesar de variáveis desses tipos ocuparem o mesmo espaço na memória.

Exercício 2.7 ★

Escreva a classe `Lampada` correspondente ao modelo da listagem 1.1 no livro. Que tipo de dado pode ser usado para representar o atributo `estado`?

Exercício 2.8 ★

Escreva na classe `Lampada` (veja o exercício 2.7) o método correspondente à resposta do exercício 1.7.

Exercício 2.9 ★

Modifique a resposta do exercício 2.7 para que a classe represente também o número de watts da lâmpada. Escreva um método `éEconômica` que retorne o valor booleano `true` se a lâmpada consumir menos de 40 watts e `false` caso contrário. *Dica:* A expressão `(a > b)` retorna `true` se `a` for maior do que `b` e `false` caso contrário.

Exercício 2.10 ★

Identifique e explique o(s) erro(s) na classe abaixo.

```
1  class Registro De Eleitor
2    {
3    /**
4     * Declaração dos atributos desta classe
5     */
6    int tituloDeEleitor; // número do título do eleitor
7    String nome; // nome do eleitor
8    short zonaEleitoral; // número da zona eleitoral
9    } // fim da classe
```

Exercício 2.11 ★

Identifique e explique o(s) erro(s) na classe abaixo.

```
1  class DoisValores
2    {
3    /**
4     * Declaração dos atributos desta classe
5     */
6    int valor1,valor2;
7    /**
8     * Declaração dos métodos desta classe
9     */
10   int maior()
11     {
12     if (valor1 > valor2)
13       return true;
14     else return false;
15     }
16   void menor()
17     {
18     if (valor1 < valor2)
19       return valor1;
20     else return valor2;
21     }
22   } // fim da classe
```

Exercício 2.12 ★

Identifique e explique o(s) erro(s) na classe abaixo.

```
class FaceDoDado
  {
  /**
   * Declaração dos atributos desta classe
   */
  int 1,2,3,4,5,6;
  /**
   * Declaração dos métodos desta classe
   */
  void 1()
    {
    System.out.println(1);
    }
  void 2()
    {
    System.out.println(2);
    }
  void 3()
    {
    System.out.println(3);
    }
  void 4()
    {
    System.out.println(4);
    }
  void 5()
    {
    System.out.println(5);
    }
  void 6()
    {
    System.out.println(6);
    }
  } // fim da classe
```

Exercício 2.13 ★

Identifique e explique o(s) erro(s) na classe abaixo.

```
class NumeroComplexo
  {
  /**
   * Declaração dos atributos desta classe
   */
  float real,imaginário;
  /**
   * Declaração dos métodos desta classe
   */
  float valor()
    {
    return real,imaginário;
    }
  } // fim da classe
```

Exercício 2.14 ★

Identifique e explique o(s) erro(s) na classe abaixo.

```
1  class Amplitude
2    {
3    /**
4     * Declaração dos atributos desta classe
5     */
6    double val1,val2,val3;
7    /**
8     * Declaração dos métodos desta classe
9     */
10   double amplitude()
11     {
12     double amplitude2()
13       {
14       return val1-val2;
15       }
16     return amplitude2()-val3;
17     }
18   } // fim da classe
```

Exercício 2.15 ★

Identifique e explique o(s) erro(s) na classe abaixo.

```
1  class Registro De Eleitor
2    {
3    /**
4     * Declaração dos atributos desta classe
5     */
6    int tituloDeEleitor; // número do título do eleitor
7    String nome; // nome do eleitor
8    short zonaEleitoral; // número da zona eleitoral
9    } // fim da classe
```

Exercício 2.16 ★

Escreva uma classe `HoraAproximada` que represente o modelo do exercício 1.10.

Exercício 2.17 ★

Usando o exercício 2.16 como referência, escreva uma classe `HoraPrecisa` que represente o modelo do exercício 1.11.

Exercício 2.18 ★

Escreva uma classe `CadernoDeEnderecos` que represente os dados de uma pessoa, como nome, telefone, e-mail e endereço. Que atributos e métodos essa classe deve ter?

Exercício 2.19 ★

Escreva a classe `Ponto2D`, correspondente ao modelo da resposta do exercício 1.18.

Exercício 2.20 ★

Escreva uma classe `Livro` que represente o modelo do exercício 1.19.

Exercício 2.21 ★

Escreva uma classe `LivroLivraria` que represente o modelo do exercício 1.20.

Exercício 2.22 ★

Escreva uma classe `LivroBiblioteca` que represente o modelo do exercício 1.21.

Exercício 2.23 ★ ★

Escreva a classe `Contador` que encapsule um valor usado para contagem de itens ou eventos. Essa classe deve esconder o valor encapsulado de programadores-usuários, fazendo com que o acesso ao valor seja feito somente através de métodos que devem zerar, incrementar e imprimir o valor do contador.

Exercício 2.24 ★ ★

Escreva, na classe `Data`, um método `duplicaData` que receba como argumento uma outra instância da classe `Data`, e duplique os valores dos campos da instância passada como argumento para os atributos encapsulados.

Exercício 2.25 ★ ★

Escreva a classe `PoligonoRegular`, correspondente à resposta do exercício 1.24.

Exercício 2.26 ★ ★

Crie uma classe `Linha2D` para representar uma linha, unida por dois pontos no espaço cartesiano de duas dimensões, usando duas instâncias da classe `Ponto2D`, criada no exercício 2.19. Veja também o exercício 1.30.

Exercício 2.27 ★ ★

Crie uma classe `Retangulo` para representar um retângulo cujos pontos opostos sejam duas instâncias da classe `Ponto2D`, que deve ter sido criada no exercício 2.19. Veja também o exercício 1.23.

Exercício 2.28 ★ ★

Modifique a classe `Lampada` para que esta contenha também um campo que indique quantas vezes a lâmpada foi acesa. Tente usar uma instância da classe `Contador` (veja o exercício 2.23). Em que método esse atributo deve ser modificado, e como?

Exercício 2.29 ★ ★ ★

Escreva uma classe `ContaBancariaSimplificada` que corresponda ao modelo na listagem 1.2 do livro. Considere que modificadores de acesso devam ser usados para os métodos e campos da classe.

Exercício 2.30 ★ ★ ★

Implemente a lógica correta de cálculo de anos bissextos e dias nos meses mostrada no exercício 1.34 na classe `Data`.

Exercício 2.31 ★ ★ ★

Se os métodos `abreConta`, `deposita` e `retira` que devem ter sido criados no exercício 2.29 forem criados como o modelo da listagem 1.2 do livro sugere, alguns erros poderão ocorrer, como abrir uma conta com valor negativo, ou depositar ou retirar valores negativos. Modifique os métodos citados para que somente valores positivos sejam considerados pelos métodos.

Exercício 2.32 ★ ★ ★

Uma das operações que podemos efetuar com datas é a comparação para ver se uma data ocorre antes de outra. O algoritmo para comparação é muito simples, e seus passos estão abaixo. Nesse algoritmo, consideramos que `dia1`, `mês1` e `ano1` são os dados da primeira data, e que `dia2`, `mês2` e `ano2` são os dados da segunda data.

1. Se `ano1 < ano2` a primeira data vem antes da segunda.
2. Se `ano1 > ano2` a primeira data vem depois da segunda.
3. Se `ano1 == ano2` e `mês1 < mês2` a primeira data vem antes da segunda.
4. Se `ano1 == ano2` e `mês1 > mês2` a primeira data vem depois da segunda.
5. Se `ano1 == ano2` e `mês1 == mês2` e `dia1 < dia2` a primeira data vem antes da segunda.
6. Se `ano1 == ano2` e `mês1 == mês2` e `dia1 > dia2` a primeira data vem depois da segunda.
7. Se nenhum desses casos ocorrer, as datas são exatamente iguais.

Escreva um método `vemAntes` na classe `Data` que receba como argumento outra instância da classe `Data` e implemente o algoritmo acima, retornando `true` se a data encapsulada vier antes da passada como argumento e `false` caso contrário. Se as datas forem exatamente iguais, o método deve retornar `true`.

Exercício 2.33 ★ ★ ★ ★

Escreva em Java uma classe `RestauranteCaseiro` que implemente o modelo descrito na figura 1.1 da seção 1.2 no livro. Para isso, crie também uma classe `MesaDeRestaurante` que represente uma mesa de restaurante conforme mostrado na figura 1.1 do livro. Algumas sugestões sobre a criação dessas classes são:

- A classe `MesaDeRestaurante` deve ter atributos para representar a quantidade de cada pedido feito, um método `adicionaAoPedido` que incrementa a quantidade de pedidos feitos, o método `zeraPedidos` que cancela todos os pedidos feitos, isto é, faz com que a quantidade de pedidos seja zero para cada item, e o método `calculaTotal`, que calcula o total a ser pago por aquela mesa. Como modelar cada item da comanda separadamente?
- A classe `RestauranteCaseiro` deve ter várias atributos que são instâncias da classe `MesaDeRestaurante`, para representar suas mesas separadamente.
- A classe `RestauranteCaseiro` também deve ter um método `adicionaAoPedido` que adicionará uma quantidade a um item de uma mesa. Esse método deverá chamar o método `adicionaAoPedido` da mesa à qual o pedido está sendo adicionado.

A solução deste exercício requer a criação de um número predeterminado e imutável de instâncias de `MesaDeRestaurante` em `RestauranteCaseiro`. Comente sobre as vantagens e desvantagens de criar classes desta forma.

Exercício 2.34 ★ ★ ★ ★

Escreva em Java a classe `NumeroComplexo` que represente um número complexo. A classe deverá ter os seguintes métodos:

- `inicializaNúmero`, que recebe dois valores como argumentos para inicializar os campos da classe (parte real e imaginária);
- `imprimeNúmero`, que deve imprimir o número complexo encapsulado usando a notação $a + bi$ onde a é a parte real e b a imaginária;
- `éIgual`, que recebe outra instância da classe `NumeroComplexo` e retorna `true` se os valores dos campos encapsulados forem iguais aos da instância passada como argumento;
- `soma`, que recebe outra instância da classe `NumeroComplexo` e soma este número complexo com o encapsulado usando a fórmula $(a + bi) + (c + di) = (a + c) + (b + d)i$;
- `subtrai`, que recebe outra instância da classe `NumeroComplexo` e subtrai o argumento do número complexo encapsulado usando a fórmula $(a + bi) - (c + di) = (a - c) + (b - d)i$;
- `multiplica`, que recebe outra instância da classe `NumeroComplexo` e multiplica este número complexo com o encapsulado usando a fórmula $(a + bi) * (c + di) = (ac - bd) + (ad + bc)i$;
- `divide`, que recebe outra instância da classe `NumeroComplexo` e divide o número encapsulado pelo passado como argumento usando a fórmula $\frac{(a+bi)}{(c+di)} = \frac{ac+bd}{c^2+d^2} + \frac{bc-ad}{c^2+d^2}i$;

Capítulo 3

Criando aplicações em Java

Este capítulo mostra como criar aplicações simples em Java. Aplicações são classes que não tem como objetivo representar modelos, mas sim criar instâncias das classes que representam os modelos. O capítulo apresenta o método especial `main` que permite que a classe possa ser executada como uma aplicação.

O capítulo também apresenta a palavra chave `new`, que permite a criação de instâncias de classes, e demonstra a criação de instâncias de classes vistas neste capítulo e no anterior. O capítulo também descreve com detalhes a relação entre instâncias e referências a estas instâncias, incluindo informações sobre instâncias sem referências, referências sem instâncias e o uso da palavra chave `null`.

3.1 Exercícios do Capítulo 3

Exercício 3.1 ★

Escreva um programa em Java que imprima o seu nome.

Exercício 3.2 ★

Escreva um programa em Java que leia o seu nome do teclado e imprima-o com uma mensagem qualquer. Veja o apêndice A para exemplos.

Exercício 3.3 ★

Explique, com suas palavras, por que uma classe como a `Ponto2D` (listagem 3.2 no livro) não pode ser executada.

Exercício 3.4 ★

Escreva um programa em Java que use várias instâncias da classe `Lampada` (veja o exercício 2.7).

Exercício 3.5 ★

Escreva uma aplicação em Java que demonstre o uso de instâncias da classe `Contador`, que deve ter sido criada como resposta ao exercício 2.23.

Exercício 3.6 ★

Escreva uma aplicação em Java que demonstre o uso de instâncias da classe `PoligonoRegular`, que deve ter sido criada como resposta ao exercício 2.25.

Exercício 3.7 ★

Identifique e explique o(s) erro(s) na classe abaixo.

```
1  public class DemoImpressao
2    {
3    main(String[] args)
4      {
5      System.out.println("7+2="+(7+2));
6      System.out.println("7-2="+(7-2));
7      System.out.println("7*2="+(7*2));
8      System.out.println("7/2="+(7/2));
9      return true;
10     }
11   } // fim da classe
```

Exercício 3.8 ★

Identifique e explique o(s) erro(s) na classe abaixo.

```
1  public static void main(String[] args)
2    {
3    Data hoje = new Data();
4    hoje.inicializaData(1,7,2013);
5    hoje.imprimeData();
6    }
```

Exercício 3.9 ★

Identifique e explique o(s) erro(s) na classe abaixo.

```
1  public class Atribuicoes
2    {
3    public static void main(String[] args)
4      {
5      Data a;
6      Data b = new Data();
7      b = null;
8      b = a;
9      }
10   } // fim da classe
```

Exercício 3.10 ★

Escreva uma aplicação que demonstra o uso de instâncias da classe que deve ter sido criada como resposta ao exercício 2.9. Demonstre o método `éEconômica`.

Exercício 3.11 ★ ★

Escreva uma aplicação que demonstre o uso de instâncias da classe `ContaBancariaSimplificada` que deve ter sido criada como resposta ao exercício 2.29. Demonstre como a transferência de valores de uma instância da classe para outra pode ser feita através de chamadas aos métodos `deposita` e `retira`. Tente fazer com que os dados que serão usados nas classes sejam lidos do teclado (veja o apêndice A no livro).

Exercício 3.12 ★ ★

Uma instância da classe `Ponto2D` foi criada na linha 46 da listagem 3.3 (no livro). Os atributos encapsulados nessa instância podem ser modificados? Por quê?

Exercício 3.13 ★ ★

Demonstre, em uma aplicação, o método `duplicaData` da classe `Data`, que deve ter sido criado como resposta ao exercício 2.24.

Exercício 3.14 ★ ★

Escreva uma aplicação que demonstre o uso de instâncias da classe `PoligonoRegular`, correspondente à resposta do exercício 2.25.

Exercício 3.15 ★ ★

Escreva uma aplicação em Java que demonstre o uso de instâncias das classes `Livro`, `LivroLivraria` e `LivroBiblioteca` (veja os exercícios 2.20, 2.21 e 2.22).

Exercício 3.16 ★ ★

Escreva uma aplicação em Java que demonstre o uso de instâncias da classe `Lampada` que incorpore um contador de quantas vezes foi acesa (veja o exercício 2.28).

Exercício 3.17 ★ ★ ★

A classe abaixo pode ser compilada sem erros. Quando for executado, o programa imprimirá que o resultado da comparação na linha 11 é `true` mas o resultado da comparação na linha 12 é `false`. Explique por quê.

```
1  public class DemoDataCopiada
2    {
3    public static void main(String[] argumentos)
4      {
5      Data lançamentoDaAtlantis18 = new Data();
6      Data inícioDeOperaçãoDoHAL = new Data();
7      Data morteDeCharlesHuggins;
8      lançamentoDaAtlantis18.inicializaData((byte)12,(byte)1,(short)1997);
9      inícioDeOperaçãoDoHAL.inicializaData ((byte)12,(byte)1,(short)1997);
10     morteDeCharlesHuggins = lançamentoDaAtlantis18;
11     System.out.println(lançamentoDaAtlantis18 == morteDeCharlesHuggins);
12     System.out.println(lançamentoDaAtlantis18 == inícioDeOperaçãoDoHAL);
13     }
14   }
```

Exercício 3.18 ★ ★ ★

Escreva uma aplicação que demonstre o uso de instâncias da classe `NumeroComplexo` que deve ter sido criada como resposta ao exercício 2.34. Demonstre o uso de todas as operações.

Capítulo 4

Construtores e sobrecarga

Neste capítulo vemos construtores, que são métodos especiais que são executados quando criamos instâncias de classes, e que podem ser usados para garantir inicialização dos atributos de instâncias. Vemos também quais são os valores *default* para diversos tipos de dados em Java, e as diferenças básicas entre construtores e outros métodos de classes.

O capítulo também apresenta o conceito de sobrecarga, que possibilita a criação de vários métodos com o mesmo nome mas diferenciados pelos tipos de argumentos passados. Sobrecarga permite a escrita de classes com métodos mais concisos, de forma a também reduzir a necessidade de reescrita de código caso alguns métodos sejam modificados.

4.1 Exercícios do Capítulo 4

Exercício 4.1 ★

Escreva um construtor para a classe `Data` que receba os valores correspondentes ao dia, mês e ano, e inicialize os campos da classe, verificando antes se a data é válida.

Exercício 4.2 ★

Escreva um construtor para a classe `Lampada` de forma que instâncias desta só possam ser criadas se um estado inicial for passado para o construtor. Esse estado pode ser o valor booleano que indica se a lâmpada está acesa (`true`) ou apagada (`false`).

Exercício 4.3 ★

Considere a classe `Lampada` que também representa o número de watts da lâmpada (veja o exercício 2.9). Escreva dois construtores para a classe: um que recebe como argumentos o número de watts da lâmpada, e outro, sem argumentos, que considera que a lâmpada tem 60 watts por *default*.

Exercício 4.4 ★

Escreva dois construtores para a classe `Contador` (exercício 2.23), um que não receba argumentos e considere que o contador começa a contar a partir do zero, e outro que aceita um valor inicial para contagem.

Exercício 4.5 ★

Identifique e explique o(s) erro(s) na classe abaixo.

```
1  public class Data
2    {
3    private byte dia,mês;
4    private short ano;
5    private Data(byte dd,byte mm,short aa)
6      {
7      dia = dd; mês = mm; ano = aa;
8      }
9    }
```

Exercício 4.6 ★

Identifique e explique o(s) erro(s) na classe abaixo.

```
1  public class Ponto2D
2    {
3    private double x,y;
4    public Ponto2D(double _x,double _y)
5      {
6      x = _x; y = _y;
7      }
8    public Ponto2D(double coord1,double coord2)
9      {
10     x = coord1; y = coord2;
11     }
12   }
```

Exercício 4.7 ★

Identifique e explique o(s) erro(s) na classe abaixo.

```
1  public class DemoConstrutor
2    {
3    private int a,b;
4    public DemoConstrutor()
5      {
6      System.out.println("No construtor sem argumentos...");
7      DemoConstrutor(0,0);
8      }
9    public DemoConstrutor(int xa,int xb)
10     {
11     System.out.println("No construtor com argumentos...");
12     a = xa; b = xb;
13     }
14   }
```

Exercício 4.8 ★

Escreva um construtor para a classe `PoligonoRegular` (exercício 2.25), que receba um valor inteiro correspondente ao número de lados do polígono.

Exercício 4.9 ★

Identifique e explique o(s) erro(s) na classe abaixo.

```
1  public class Media
2    {
3    public int Media(int a,int b)
4      {
5      return (a+b)/2;
6      }
7    public double Media(int a,int b)
8      {
9      return (a+b)/2;
10     }
11   }
```

Exercício 4.10 ★

Considerando a classe `RoboSimples` (listagem 4.5 no livro), quais das chamadas ao método `move` abaixo podem ser usadas? Explique.

- `move();`
- `move(1);`
- `move('A');`
- `move("A");`
- `move(1/3);`
- `move(2,3,5);`
- `move(9,false);`
- `move("17");`
- `move((long)3);`
- `move((char)65);`

Exercício 4.11 ★

Liste as assinaturas dos construtores e métodos na classe `RoboSimples` (listagem 4.5 no livro).

Exercício 4.12 ★ ★

Escreva outro construtor para a classe `Data` que receba uma instância da própria classe `Data` e use os dados desta para inicializar os campos. Veja também o exercício 4.1.

Exercício 4.13 ★ ★

Escreva três construtores para a classe `NumeroComplexo` (exercício 2.34). Um construtor deverá receber os dois valores (real e imaginário) como argumentos, o outro somente o valor real, considerando o imaginário como sendo zero, e o terceiro construtor não recebe argumentos, considerando as partes real e imaginária do número complexo como sendo iguais a zero.

Exercício 4.14 ★ ★

Considerando as classes abaixo, para cada chamada ao método `doisValores` identifique que forma do método será chamada.

```
1  public class Soma
2    {
3    public int doisValores(int a,int b) // soma dois inteiros
4      {
5      return a+b;
6      }
7    public double doisValores(double a,int b) // soma um double e um inteiro
8      {
9      return a+b;
10     }
11   public double doisValores(double a,double b) // soma dois doubles
12     {
13     return a+b;
14     }
15   }
16
17 public class TesteSoma
18   {
19   public static void main(String[] args)
20     {
21     Soma soma = new Soma(); // cria instância da classe Soma
22     // Declara várias variáveis
23     byte b = 20;
24     short s = 99;
25     int i = 1000;
26     long l = 1234L;
27     float f = 3.1416f;
28     double d = 2000;
29     // Chama vários métodos da classe Soma
30     System.out.println(soma.doisValores(b,s));
31     System.out.println(soma.doisValores(i,s));
32     System.out.println(soma.doisValores(i,i));
33     System.out.println(soma.doisValores(l,b));
34     System.out.println(soma.doisValores(f,s));
35     System.out.println(soma.doisValores(d,b));
36     System.out.println(soma.doisValores(b,d));
37     System.out.println(soma.doisValores(i,l));
38     System.out.println(soma.doisValores(l,l));
39     System.out.println(soma.doisValores(d,f));
40     }
41   }
```

Exercício 4.15 ★ ★

Escreva dois construtores para a classe `Ponto2D` (listagem 3.2 no livro): um sem argumentos que considere que o ponto está na origem, ou seja, com coordenadas $(0, 0)$, e um que receba dois argumentos do tipo `double` e que os use para inicializar os campos da classe.

Exercício 4.16 ★ ★

O que aconteceria se no construtor da classe `EventoAcademico` as instâncias internas da classe `Data` fossem simplesmente igualadas às instâncias passadas como argumentos (por exemplo, se escrevêssemos `inícioDoEvento = i; fimDoEvento = f;` em vez do trecho entre as linhas 37 e 40 da listagem 4.3 no livro)? Explique, usando a aplicação na listagem 4.4 como exemplo.

Exercício 4.17 ★ ★

Escreva dois construtores para a classe `ContaBancariaSimplificada` (exercício 2.29), um que inicialize todos os campos da classe e outro que considere que o saldo inicial será zero e a conta não será especial.

Exercício 4.18 ★ ★

Suponha que os robôs modelados pela classe `RoboSimples` (listagem 4.5 no livro) possam se movimentar para a frente e para trás. Escreva na classe dois métodos `moveParaTrás`, um que mova os robôs uma unidade e outro que aceite um valor como argumento (número de unidades a mover). *Dica:* Mover um robô n unidades para trás é a mesma coisa que movê-lo $-n$ unidades para a frente, então podemos chamar o método `move` de dentro do método `moveParaTrás`, trocando o sinal do valor do movimento.

Exercício 4.19 ★ ★ ★

Escreva quatro construtores para a classe `Linha` (exercício 2.26): um sem argumentos que considere que a linha comece e termine no ponto $(0, 0)$; um que receba um argumento do tipo `Ponto2D` e que considere que a linha comece na origem e termine no ponto passado como argumento; um que receba duas instâncias da classe `Ponto2D` como argumentos e um que receba quatro valores de ponto flutuante, correspondentes às duas coordenadas.

Exercício 4.20 ★ ★ ★

Escreva quatro construtores para a classe `Retangulo` (exercício 2.27): um sem argumentos que considere que os dois pontos extremos do retângulo tenham coordenadas iguais a $(0, 0)$; um que receba um argumento do tipo `Ponto2D` e que considere que um dos pontos extremos do retângulo está na origem do sistema de coordenadas e que o outro seja o ponto passado como argumento; um que receba duas instâncias da classe `Ponto2D` como argumentos e as considere como pontos extremos do retângulo; e um que receba quatro valores de ponto flutuante, correspondentes às duas coordenadas dos pontos extremos.

Capítulo 5

Atributos e métodos estáticos

Neste capítulo vemos atributos e métodos estáticos, ou seja, que são invariantes para todas as instâncias de determinada classe. Atributos estáticos podem ser usados para compartilhar valores entre todas as instâncias de uma classe e para definir valores constantes que podem ser usados em qualquer classe. Métodos estáticos facilitam o desenvolvimento de aplicações, podendo ser considerados equivalentes a subrotinas em programação procedural.

O capítulo também introduz o conceito de fábricas de instâncias, que permitem a criação de instâncias de classes de forma mais controlável; e enumeradores, uma forma mais elegante e flexível de definir valores constantes em Java.

5.1 Exercícios do Capítulo 5

Exercício 5.1 ★

Explique, com suas palavras, por que os atributos na classe `ConstantesMatematicas` (listagem 5.6) não devem ser declarados com o modificador `private`.

Exercício 5.2 ★

Considerando a existência de métodos-fábrica, é justificável a criação de construtores privados? Dê um exemplo.

Exercício 5.3 ★

Escreva, para a classe `DataComFabrica` (listagem 5.11 no livro), um método `seteDeSetembro` que se comporte como uma fábrica de instâncias. Que argumentos este método deve receber?

Exercício 5.4 ★

Escreva, para a classe `DataComFabrica` (listagem 5.11 no livro), um método `primeiroDoMês` que se comporte como uma fábrica de instâncias. Que argumentos este método deve receber?

Exercício 5.5 ★

Podemos ter várias versões do método `main` em uma classe, usando a sobrecarga de métodos? Explique.

Exercício 5.6 ★

A distância média da Terra à Lua é de aproximadamente 384.400 quilômetros. Usando a classe `ConversaoDeUnidadesDeComprimento` (listagem 5.8 no livro), escreva um programa em Java que mostre qual é a distância média da Terra à Lua em milhas e pés. Escreva métodos adicionais para a classe `ConversaoDeUnidadesDeComprimento`, se necessário.

Exercício 5.7 ★ ★

O método `main` pode ser chamado a partir de outro método estático da mesma classe. Se isso for feito, que problemas potenciais podem ocorrer na aplicação?

Exercício 5.8 ★ ★

O que aconteceria se não inicializássemos os atributos da classe `ConstantesMatematicas` listagem 5.6 no livro? Explique.

Exercício 5.9 ★ ★

Crie um enumerador em Java para representar as cores do arco-íris.

Exercício 5.10 ★ ★

Escreva a classe `ConversaoDeUnidadesDeArea` com métodos estáticos para conversão das unidades de área segundo a lista abaixo.

- 1 metro quadrado = 10.76 pés quadrados
- 1 pé quadrado = 929 centímetros quadrados
- 1 milha quadrada = 640 acres
- 1 acre = 43.560 pés quadrados

Exercício 5.11 ★ ★

A área de um campo de futebol é de 8.250 metros quadrados. Usando a classe `ConversaoDeUnidadesDeArea` (exercício 5.10), escreva um programa em Java que mostre qual é a área de um campo de futebol em pés quadrados, acres e centímetros quadrados. Escreva métodos adicionais para a classe `ConversaoDeUnidadesDeArea`, se necessário.

Exercício 5.12 ★ ★

Escreva a classe `ConversaoDeUnidadesDeVolume` com métodos estáticos para conversão das unidades de volume segundo a lista abaixo.

- 1 litro = 1000 centímetros cúbicos
- 1 metro cúbico = 1000 litros
- 1 metro cúbico = 35.32 pés cúbicos
- 1 galão americano = 231 polegadas cúbicas
- 1 galão americano = 3.785 litros

Exercício 5.13 ★ ★

O volume de uma piscina olímpica é de 1.890 metros cúbicos. Usando a classe `ConversaoDeUnidadesDeVolume` (exercício 5.12), escreva um programa em Java que mostre qual é o volume de uma piscina olímpica em litros, pés cúbicos e centímetros cúbicos. Escreva métodos adicionais para a classe `ConversaoDeUnidadesDeVolume`, se necessário.

Exercício 5.14 ★ ★

Escreva a classe `ConversaoDeUnidadesDeTempo` com métodos estáticos para conversão **aproximada** das unidades de velocidade segundo a lista abaixo.

- 1 minuto = 60 segundos
- 1 hora = 60 minutos
- 1 dia = 24 horas
- 1 semana = 7 dias
- 1 mês = 30 dias
- 1 ano = 365.25 dias

Exercício 5.15 ★ ★

O tempo de gestação de um elefante indiano é de aproximadamente 624 dias. Usando a classe `ConversaoDeUnidadesDeTempo` (exercício 5.14), escreva um programa em Java que mostre qual é o tempo de gestação de um elefante indiano em dias, horas, minutos e segundos. Escreva métodos adicionais para a classe `ConversaoDeUnidadesDeTempo`, se necessário.

Exercício 5.16 ★ ★

Escreva uma classe `ConversaoDeUnidadesDeTemperatura` que contenha métodos estáticos para calcular a conversão entre diferentes escalas de temperatura. Considere as fórmulas de conversão abaixo:

- De graus Celsius (C) para graus Fahrenheit (F): $F = (9 \times C/5) + 32)$
- De graus Fahrenheit (F) para graus Celsius (C): $C = (F - 32) \times +5/9$
- De graus Celsius (C) para graus Kelvin (K): $K = C + 273.15$
- De graus Kelvin (K) para graus Celsius (C): $C = K - 273.15$
- De graus Celsius (C) para graus Réaumur (Re): $Re = C * 4/5$
- De graus Réaumur (Re) para graus Celsius (C): $C = Re * 5/4$
- De graus Kelvin (K) para graus Rankine (R): $R = K * 1.8$
- De graus Rankine (R) para graus Kelvin (K): $K = R/1.8$

Veja que já que existem cinco sistemas de medidas de temperatura, devem haver 20 diferentes métodos de conversão de temperatura. Alguns podem ser escritos indiretamente, por exemplo, para converter de Celsius para Rankine, podemos converter de Celsius para Kelvin e converter esse resultado para Rankine.

Exercício 5.17 ★ ★ ★

Escreva uma classe `SerieLimitada`, que encapsula um valor inteiro sequencial como os usados em notas e séries de gravuras. Essa classe deve permitir que um programa crie um número limitado de instâncias dela, cada uma numerada com um valor sequencial. O número total de instâncias é controlado pelo campo `máximoDeInstâncias`, declarado como `static final`, e o de instâncias já criadas é controlado pelo campo `contador` declarado como `static`. Escreva também uma aplicação que crie algumas instâncias da classe para demonstrar seu funcionamento.

Exercício 5.18 ★ ★ ★

Escreva uma versão da classe `ContaBancariaSimplificada` (exercício 2.29) que contenha um campo `númeroDaConta` declarado como `static`, e que incremente o valor desse campo cada vez que uma instância da classe for criada. Escreva também uma aplicação que crie algumas instâncias da classe para demonstrar seu funcionamento.

Exercício 5.19 ★ ★ ★

Escreva uma classe que contenha métodos estáticos para retornar o maior e o menor de dois, três, quatro e cinco valores (com um total de oito métodos), considerando que os argumentos e retorno dos métodos podem ser dos tipos `int` e `double`. *Dica:* Os métodos podem ser chamados em cascata: para calcular o maior de três valores a, b e c, pode-se calcular o maior valor de a e b, e comparar esse resultado com c.

Exercício 5.20 ★ ★ ★

Escreva uma classe `VeiculoAVenda` que represente informações básicas sobre um veículo genérico que esteja à venda, como `tipo`, `ano` e `preçoDeVenda`, com um construtor para inicializar estes valores e um método `toString` adequado. Escreva também um enumerador para os diversos tipos de veículos (automóveis, caminhões, motocicletas, etc.).

Exercício 5.21 ★ ★ ★

Crie um enumerador que represente as moedas em uso atualmente no Brasil. Associe a símbolos representando as moedas os seus valores.

Exercício 5.22 ★ ★ ★

Crie um enumerador que represente os dias da semana. Associe aos símbolos dos dias da semana valores booleanos que indicam se o dia é fim-de-semana ou não.

Exercício 5.23 ★ ★ ★

Crie um enumerador que represente os meses em um ano. Associe aos símbolos dos meses valores inteiros que indicam quantos dias temos no mês se o ano for bissexto **e** não bissexto.

Exercício 5.24 ★ ★ ★

Crie um enumerador que represente diferentes tipos de tortas em uma lanchonete. Associe aos símbolos dos tipos de tortas valores inteiros correspondentes às calorias da torta e valores de ponto flutuante correspondentes aos preços das tortas.

Exercício 5.25 ★ ★ ★

Crie um enumerador que represente os países da América do Sul. Associe aos símbolos dos países valores para representar sua área, população e capital.

Capítulo 6

Estruturas de decisão e controle – Condicionais

Uma característica importante de computadores e dos programas que os controla, que lhes dá uma enorme flexibilidade de aplicações, é a capacidade de tomar decisões que afetam que trechos do programa serão executados sob certas condições. Estruturas condicionais afetam o fluxo de execução do código dependendo do resultado da avaliação de um operador.

Este capítulo apresenta as três estruturas de controle em Java: blocos de instruções `if-else` que podem ser usados para determinar a execução de um trecho de código baseado em um teste booleano; o operador condicional `?` que se comporta como uma versão simplificada de `if-else`; e blocos de instruções `switch` que executam trechos de código baseado em comparações de igualdade.

6.1 Exercícios do Capítulo 6

Exercício 6.1 ★

Escreva para a classe `Comparavel` (listagem 6.1) o método `éIgualAQualquerUmDe` que aceite dois valores como argumentos e retorne `true` se o valor encapsulado for igual a qualquer um dos passados como argumentos.

Exercício 6.2 ★

Escreva versões do método `éIgualAQualquerUmDe` (veja o exercício 6.1) que aceitem três, quatro e cinco valores do tipo `double` como argumentos, e retorne `true` se o valor encapsulado for igual a qualquer um dos valores passados como argumentos.

Exercício 6.3 ★

O método `calculaPreço` na classe `EntradaDeCinema` (listagem 6.4 no livro) verifica primeiro se o dia da semana é dia de desconto, para depois verificar a idade do cliente. Modifique esse método para que primeiro a idade seja verificada, para depois verificar o dia da semana, de forma que o resultado final seja o mesmo.

Exercício 6.4 ★

Modifique o método `calculaPreço` da classe `EntradaDeCinema` (listagem 6.4 no livro) para que este também considere que horas são, e retorne o preço de meia entrada para antes de quatro horas.

Exercício 6.5 ★

O que aconteceria se todos os `else` fossem retirados do método `toString` da classe `DataIf` (listagem 6.7 no livro)? Existe alguma vantagem ou desvantagem em fazer isso?

Exercício 6.6 ★

O método `mudaDireção` da classe `RoboSimples` (listagem 4.5 no livro) não verifica se a direção passada como argumento é uma das direções válidas (`'N'`, `'S'`, `'E'` ou `'O'`). Modifique o método de forma que, se um caracter diferente dos aceitos como direções válidas for passado, o método considere a direção como sendo `'N'`.

Exercício 6.7 ★

Modifique o método `diasNoMês` da classe `DataSwitch` (listagem 6.9 no livro) para que ele use comandos `if` em vez de `switch`. Existe alguma vantagem em fazer isto neste caso?

Exercício 6.8 ★

Explique e exemplifique o que aconteceria com o método `diasNoMês` da classe `DataSwitch` (listagem 6.9 no livro) se os comandos `break` fossem retirados das instruções `case` do método.

Exercício 6.9 ★ ★

Escreva uma classe que encapsule uma carta de baralho, com um valor que represente o valor da carta, de um (ás) a treze (rei), e outro valor correspondente ao naipe (1 = ouros, 2 = paus, 3 = copas e 4 = espadas). Escreva nessa classe um método `toString` que retorne o nome da carta por extenso, usando a instrução `switch`.

Exercício 6.10 ★ ★

Escreva, para a classe `Ponto2D` (listagem 3.2 no livro), os métodos `estáAcimaDe`, `estáAbaixoDe`, `estáÀEsquerdaDe` e `estáÀDireitaDe` que recebem como argumento uma outra instância da classe `Ponto2D` e retornam `true` se o ponto encapsulado estiver, respectivamente, acima, abaixo, à esquerda e à direita do ponto passado como argumento.

Exercício 6.11 ★ ★

Escreva para a classe `Comparavel` (listagem 6.1 no livro) os métodos `éMaiorOuIgual`, `éMenorOuIgual` e `éDiferenteDe` que recebam um valor do tipo `double` como argumento e retorne `true` se o valor encapsulado for, respectivamente, maior ou igual, menor ou igual, ou diferente do passado como argumento. *Dica:* Este problema pode também ser resolvido usando-se os métodos `éIgualA`, `éMenorQue` e `éMaiorQue`, já existentes na classe, e as operações booleanas "ou" e "e".

Exercício 6.12 ★ ★

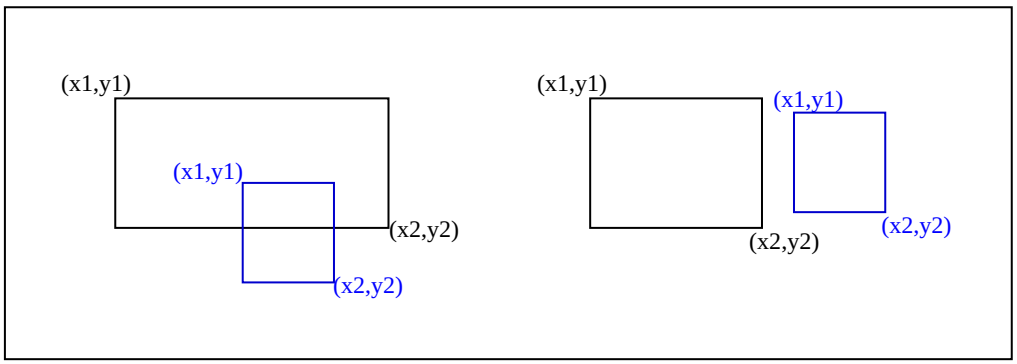
Escreva versões do método `éDiferenteDe` (veja o exercício 6.11) que aceitem três, quatro e cinco valores do tipo `double` como argumentos, e retorne `true` se o valor encapsulado for diferente de todos os valores passados como argumentos.

Exercício 6.13 ★ ★ ★

Escreva uma aplicação em Java que simule uma calculadora bem simples. Esse programa deve ler dois valores de ponto flutuante do teclado e um caracter, correspondente a uma das operações básicas (`+`, `-`, `*` ou `/`), calcular a operação e imprimir o resultado. A aplicação deve considerar divisões por zero como sendo erros, e imprimir uma mensagem adequada. A aplicação também deve considerar a possibilidade de o usuário entrar um valor diferente de `+`, `-`, `*` ou `/` como operador, e imprimir uma mensagem de erro.

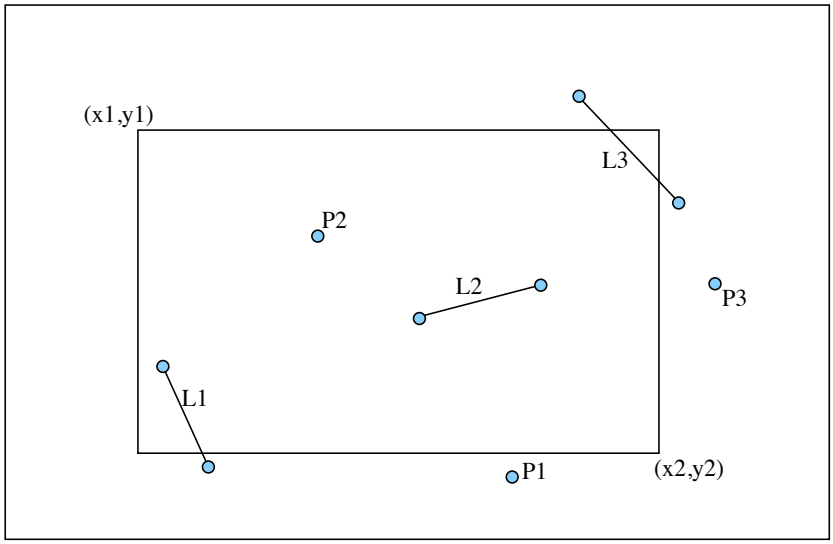
Exercício 6.14 ★ ★ ★ ★

Modifique a classe `Retangulo` (exercício 2.27) para que esta contenha um método `calculaIntersecção`, que recebe como argumento uma outra instância da própria classe `Retangulo` e que calcula as coordenadas de um retângulo que é a intersecção do retângulo encapsulado com o passado como argumento, retornando uma **nova** instância da classe `Retangulo` correspondente à intersecção. *Dicas:* Os pontos do novo retângulo podem ser calculados com regras simples, implementadas através de `if`s encadeados. Nem sempre existe intersecção entre dois retângulos. Considere a figura abaixo: no lado esquerdo existem dois retângulos (mostrados em cores diferentes) que têm intersecção, e, no lado direito, dois que não têm. No caso de não existir intersecção, o método deve retornar `null`. Veja também o exercício 6.15.



Exercício 6.15 ★ ★ ★ ★

Modifique a classe `Retangulo` (exercício 2.27) para que esta contenha um método para verificar se um ponto, passado como argumento, está localizado dentro de um retângulo. O ponto deve ser representado por uma instância da classe `Ponto2D` (exercício 2.19). O método deverá retornar `true` se o ponto estiver contido no retângulo, e `false` se não estiver. *Dica:* Para verificar se um ponto está dentro do retângulo, verifique se as coordenadas do ponto estão dentro das coordenadas do retângulo. Considerando a figura abaixo, onde $(x1, y1)$ e $(x2, y2)$ são as coordenadas que definem o retângulo, o ponto $P1$ estaria fora do retângulo, uma vez que a sua coordenada y é menor do que a menor coordenada y do retângulo. O ponto $P2$ estaria dentro do retângulo, e o ponto $P3$ também estaria fora do retângulo.



Exercício 6.16 ★ ★ ★ ★

Modifique a classe `Retangulo` (exercício 2.27) para que esta contenha um método para verificar se uma linha, passada como argumento, está localizada inteiramente dentro de um retângulo. A linha deve ser representado por uma instância da classe `Linha2D` (exercício 2.26). O método deverá retornar `true` se a linha estiver contida no retângulo, e `false` se não estiver. *Dica:* Para verificar se uma linha está dentro do retângulo, verifique se as coordenadas dos dois pontos que definem a linha estão dentro das coordenadas do retângulo. Somente se os dois pontos estiverem dentro do retângulo, a linha também estará: na figura mostrada no exercício 6.15, a linha $L2$ está dentro do retângulo, as linhas $L1$ e $L3$, não.

Capítulo 7

Estruturas de decisão e controle – Repetição

Outra característica importante de programas de computadores é a que permite repetir de forma controlada a execução de comandos, o que é essencial para a grande maioria das aplicações sérias. Com esta possibilidade podemos executar individualmente operações sobre elementos em listas de qualquer natureza e efetuar cálculos matemáticos repetitivos.

Neste capítulo vemos os conceitos pertinentes a laços, as diferentes formas de implementá-los em Java e o uso correto de variáveis de controle e contadores. Vemos também uma breve introdução à recursividade, técnica usada para resolver um problema pela resolução de partes similares mas mais simples do mesmo problema.

7.1 Exercícios do Capítulo 7

Exercício 7.1 ★

Após a execução do trecho de código `int a,b,c; a=2; b=3; c= a++ + b++;`, quais serão os valores das variáveis?

A. a=3, b=4 e c=7.
B. a=3, b=4 e c=5.
C. a=2, b=3 e c=7.
D. a=2, b=3 e c=5.

Exercício 7.2 ★

Após a execução do trecho de código `long x,y,z; x=0; y=12; z= ++x + ++y;`, quais serão os valores das variáveis?

A. x=0, y=12 e z=12.
B. x=0, y=12 e z=14.
C. x=1, y=13 e z=12.
D. x=1, y=13 e z=14.

Exercício 7.3 ★

Por que um bloco `while` iniciado por `while(true)` pode ser útil, enquanto um bloco iniciado por `while(false)` certamente será inútil? Explique.

Exercício 7.4 ★

Escreva um programa em Java que, usando a classe `ConversaoDeUnidades-DeComprimento` (listagem 5.8 no livro), imprima as distâncias em milhas e pés correspondentes às distâncias em metros de zero a cem metros.

Exercício 7.5 ★

Escreva um programa em Java que, usando a classe `ConversaoDeUnidadesDeTempo` (exercício 5.14), imprima o tempo em segundos, horas, semanas e anos correspondentes ao tempo em dias, de um a trinta e um dias.

Exercício 7.6 ★

Escreva um programa em Java que, usando a classe `ConversaoDeTemperatura` (exercício 5.16), imprima as temperaturas em graus Kelvin e Fahrenheit correspondentes aos graus Celsius entre zero e cem graus.

Exercício 7.7 ★

Identifique e explique o(s) erro(s) na classe abaixo.

```
1  public class ContadorComFor
2    {
3    public static void main(String[] argumentos)
4      {
5      double a, b = 1;
6      for (a = 0; a < 1000; b++)
7        {
8        System.out.println(a + " " + b);
9        }
10     }
11   }
```

Exercício 7.8 ★

Identifique e explique o(s) erro(s) na classe abaixo.

```
1  public class ContadorComWhile1
2    {
3    public static void main(String[] argumentos)
4      {
5      int contador = 0;
6      while (contador != 100)
7        {
8        contador = contador + 3;
9        }
10     }
11   }
```

Exercício 7.9 ★

Identifique e explique o(s) erro(s) na classe abaixo.

```
1  public class ContadorComWhile2
2    {
3    public static void main(String[] argumentos)
4      {
5      double valor = 100;
6      while (valor < 100)
7        {
8        valor /= 2;
9        }
10     }
11   }
```

Exercício 7.10 ★

Explique por que o uso de recursão deve ser implementado através de métodos.

Exercício 7.11 ★

Modifique as classes `JogoDeAdivinhacao` e `UsaJogoDeAdivinhacao` (listagens 7.5 e 7.6 no livro) para que o número de tentativas seja também variável, isto é, que não seja fixo na classe `JogoDeAdivinhacao`.

Exercício 7.12 ★

O valor de x^y pode ser calculado como sendo x multiplicado por si mesmo y vezes (se y for inteiro). Escreva uma classe `SeriesMatematicas` que contenha o método estático `elevadoA` que receba como argumentos os valores x e y e calcule e retorne x^y.

Exercício 7.13 ★ ★

Considere duas variáveis `X` e `Y` que possam assumir valores entre -100 e 100. Escreva um programa em Java que imprima todos os valores de `X` e `Y` para os quais a soma `X + Y` seja igual a 100 ou igual a -100.

Exercício 7.14 ★ ★

Escreva uma classe `Loteria` que tenha métodos estáticos para imprimir versões aproximadas dos cartões da Mega-Sena e LotoMania (somente com os números, respeitando o número de linhas e a distribuição dos números nas linhas).

Exercício 7.15 ★ ★

Escreva um programa em Java que imprima a série de Fibonacci até o N-ésimo elemento, sem usar recursão. O número de elementos N pode ser lido do teclado.

Exercício 7.16 ★ ★

Usando a classe `JogoDeAdivinhacao` (listagem 7.5 no livro) como base, escreva uma classe `SenhaDeCaixaEletronico` com um método que peça ao usuário para entrar com uma senha numérica, e, se em três tentativas o usuário não acertar a senha, imprima uma mensagem de erro. Escreva também uma aplicação que demonstre a classe.

Exercício 7.17 ★ ★

Reescreva o método `tenta` da classe `JogoDeAdivinhacao` (listagem 7.5 no livro) de forma que o `if` com `break` seja eliminado, e que as duas condições de término do laço (número correto ou fim do número de tentativas) sejam tratadas pela instrução `while` do bloco `do-while`.

Exercício 7.18 ★ ★ ★

Escreva uma versão recursiva do método `elevadoA` (exercício 7.12) para a classe `SeriesMatematicas`.

Exercício 7.19 ★ ★ ★

Considerando o exercício 6.13 como base, modifique a aplicação de forma que o usuário não possa entrar um caractere diferente de `+`, `-`, `*` ou `/` para a operação – a aplicação deverá repetir a pergunta até que um dos operadores válidos seja entrado.

Exercício 7.20 ★ ★ ★

Considerando os exercícios 6.13 e 7.19 como base, modifique a aplicação para que, caso o segundo valor entrado seja igual a zero, o programa não permita a escolha da operação divisão.

Exercício 7.21 ★ ★ ★

Escreva uma classe `Serie` que encapsule o mecanismo de geração de séries numéricas como as usadas em testes de raciocínio, onde uma pessoa deve deduzir a regra que gerou os números e acertar os próximos números da série. A série deve ser gerada usando três valores: *inicial*, *multiplicador* e *adicional*, de forma que o primeiro número da série será igual a *inicial*, o segundo será calculado como (*inicial* + *adicional*) * *multiplicador*, o terceiro como (*segundo* + *adicional*) * *multiplicador*, e assim sucessivamente. Os valores devem ser passados como argumentos para o construtor da classe e usados por um método `imprime`, que recebe como argumento o número de termos que serão impressos. Por exemplo, uma aplicação poderia criar três instâncias da classe `Serie` e imprimir, respectivamente, os primeiros 10, 12 e 14 termos, com o trecho de código abaixo:

```
1 Serie s1 = new Serie(0,-2,2);
2 s1.imprime(10);
3 Serie s2 = new Serie(1,2,0);
4 s2.imprime(12);
5 Serie s3 = new Serie(1,1,2);
6 s3.imprime(14);
```

O resultado da execução do trecho de código acima é mostrado abaixo.

```
1 0 -4 4 -12 20 -44 84 -172 340 -684
2 1 2 4 8 16 32 64 128 256 512 1024 2048
3 1 3 5 7 9 11 13 15 17 19 21 23 25 27
```

Exercício 7.22 ★ ★ ★

Modifique a classe `Serie` (exercício 7.21) para que o método `imprime` receba um argumento inicial do tipo `int` que indica qual dos valores da série deve ser impresso como um asterisco – dessa forma fica mais fácil simular testes de raciocínio. Por exemplo, uma aplicação poderia criar uma série como `Serie novaSerie = new Serie(-20,-2,-3);` `novaSerie.imprime(10,5);` , e o resultado da execução desse trecho de código seria `-20 46 -86 178 * 706 -1406 2818 -5630 11266` (o quinto elemento da série está oculto).

Exercício 7.23 ★ ★ ★

Escreva uma aplicação em Java que calcule o máximo divisor comum de dois números. O algoritmo de Euclides para o cálculo do máximo divisor comum entre dois números positivos M e N calcula $MDC(M, N)$ como:

- Se $N > M$, retorne $MDC(N, M)$.
- Se $N = 0$, retorne M.
- Senão, retorne $MDC(N, M\%N)$ (onde % é o operador módulo, que retorna o resto da divisão).

Exercício 7.24 ★ ★ ★

A raiz quadrada de um número pode ser encontrada usando-se um algoritmo recursivo, que usa como entrada três valores: N que é o número do qual queremos calcular a raiz quadrada; A que é uma aproximação inicial da raiz quadrada; e E que é o máximo de erro que pode ser admitido no cálculo. Os passos do algoritmo são:

- Se o valor absoluto de $A^2 - N$ for menor do que E, retorne A.
- Senão, faça $A = (A^2 + N)/(2A)$ e execute novamente o algoritmo.

O valor absoluto de um valor Y pode ser calculado com o método `Math.abs(Y)`.

Exercício 7.25 ★ ★ ★

Escreva um programa em Java que calcule a função de Ackermann. A função de Ackermann (A) é calculada como:

- Se $x == 0$, $A(x, y) = y + 1$.
- Se $y == 0$, $A(x, y) = A(x - 1, 1)$.
- Senão, $A(x, y) = A(x - 1, A(x, y - 1))$.

Evite tentar calcular o valor dessa função para valores grandes de x e/ou y – o cálculo de $A(3, 4)$ gera mais de 10.000 chamadas recursivas.

Exercício 7.26 ★ ★ ★

Modifique as classes `JogoDeAdivinhacao` e `UsaJogoDeAdivinhacao` (listagens 7.5 e 7.6 no livro) para que o usuário tenha que adivinhar um de dois números escolhidos. Dois números devem ser passados para o construtor, e o método `tenta` deve dizer, quando o usuário entrar um valor pelo teclado, se esse valor é maior ou menor do que cada um dos dois valores secretos. Se o usuário acertar um dos dois valores, ele vence o jogo.

Exercício 7.27 ★ ★ ★

Escreva um programa em Java que calcule a série $\frac{1}{1\times3}+\frac{1}{2\times4}+\frac{1}{3\times5}+\frac{1}{4\times6}+\cdots$ com N termos, sendo que o valor de N pode ser entrado via teclado.
O resultado da série, se calculada infinitamente, será igual a 3/4. Evidentemente a série não poderá ser calculada infinitamente, devendo parar depois de N termos, sendo que o valor de N deve ser fornecido como argumento ao método. Um recurso interessante é mostrar os valores calculados nos diferentes passos da série para ver se ela está convergindo.

Exercício 7.28 ★ ★ ★

Escreva um programa em Java que calcule a série $\frac{1}{1\times3}+\frac{1}{3\times5}+\frac{1}{5\times7}+\frac{1}{7\times9}+\cdots$ com N termos, sendo que o valor de N pode ser entrado via teclado. O resultado da série, se calculada infinitamente, será igual a 1/2. Veja também o exercício 7.27.

Exercício 7.29 ★ ★ ★ ★

Escreva uma versão recursiva do programa que calcula a sequência que converge para 1/2 (exercício 7.28).

Exercício 7.30 ★ ★ ★ ★

Escreva uma versão recursiva do programa que calcula a sequência que converge para 3/4 (exercício 7.27).

Exercício 7.31 ★ ★ ★ ★

Implemente uma solução recursiva para a geração de séries de números do exercício 7.21, de forma que o método `imprime` chame um outro método, que por sua vez será chamado recursivamente, passando para esse outro método os argumentos necessários para a geração da série. Como será feito o controle de parada da recursão?

Exercício 7.32 ★ ★ ★ ★ ★

A base dos logaritmos naturais (e) pode ser calculada por qualquer uma das séries ou sequências abaixo:

$$e = \frac{2}{1}\left(\frac{4}{3}\right)^{\frac{1}{2}}\left(\frac{6\times8}{5\times7}\right)^{\frac{1}{4}}\left(\frac{10\times12\times14\times16}{9\times11\times13\times15}\right)^{\frac{1}{8}}\left(\frac{18\times20\times22\times24\times26\times28\times30\times32}{17\times19\times21\times23\times25\times27\times29\times31}\right)^{\frac{1}{16}}\cdots$$

$$e = \left(\frac{2}{1}\right)^{\frac{1}{2}}\left(\frac{2\times4}{3\times3}\right)^{\frac{1}{4}}\left(\frac{4\times6\times6\times8}{5\times5\times7\times7}\right)^{\frac{1}{8}}\left(\frac{8\times10\times10\times12\times12\times14\times14\times16}{9\times9\times11\times11\times13\times13\times15\times15}\right)^{\frac{1}{16}}\cdots$$

Escreva um programa em Java que, usando as séries acima, calcule o valor de e (base dos logaritmos naturais). Tente avaliar qual dos métodos é o mais eficiente: sabendo que e é aproximadamente igual a 2.718281828459045, execute os métodos até que a diferença entre o valor calculado e o valor constante seja muito pequena, e indique o número de iterações que cada um dos métodos demorou para atingir essa marca.

Capítulo 8

Reutilização de classes

Um dos conceitos principais de orientação a objetos é o de reuso de classes – comportamento e atributos já definidos em uma classes podem ser usados em outras, evitando assim que código tenha que ser reescrito, minimizando a possibilidade de erro e facilitando a manutenção do código.

Este capítulo apresenta os dois conceitos de reuso de classe implementados em Java: delegação ou composição (uso de classes já existentes como atributos em novas classes) e herança (derivação de classes novas a partir de existentes). A influência dos diverentes modificadores de acesso sobre classes herdeiras também é apresentada e exemplificada.

8.1 Exercícios do Capítulo 8

Exercício 8.1 ★

É possível evitar completamente a necessidade de sobreposição de métodos criando métodos em classes descendentes que tenham assinaturas diferentes. Por exemplo, a classe `Pessoa` poderia ter o método `imprimePessoa` para imprimir seus atributos, e a classe `Aluno` que estende a classe `Pessoa` poderia ter o método `imprimeAluno` para imprimir os atributos de `Aluno`. Que vantagens e desvantagens essa abordagem teria sobre a sobreposição de métodos?

Exercício 8.2 ★

Identifique e explique o(s) erro(s) na classe abaixo.

```
1  public class DataHora extends Data,Hora
2    {
3    public DataHora(byte d, byte m, short a, byte hor, byte min, byte seg)
4      {
5      super(d, m, a);
6      super(hor, min, seg);
7      }
8
9    public String toString()
10     {
11     return super.toString() + " " + super.toString();
12     }
13   }
```

Exercício 8.3 ★

Identifique e explique o(s) erro(s) nas classes abaixo.

```
1  public class Ponto2D_V2
2    {
3    private double x, y;
4
5    public Ponto2D_V2(double _x, double _y)
6      {
7      x = _x;
8      y = _y;
9      }
10   }
```

```
1  public class Ponto3D_V2 extends Ponto2D_V2
2    {
3    private double z;
4
5    public Ponto3D_V2(double _x, double _y, double _z)
6      {
7      x = _x;
8      y = _y;
9      z = _z;
10     }
11   }
```

Exercício 8.4 ★

Explique, com suas palavras, por que construtores de superclasses não são herdados por subclasses.

Exercício 8.5 ★

Explique com suas palavras o que aconteceria se removêssemos a palavra-chave `super` da linha 35 da classe `Funcionario` (listagem 8.9 no livro).

Exercício 8.6 ★

Para cada uma das cinco instâncias das classes criadas no método `main` da classe `EmprestimoBancario` (listagem 8.22 no livro), verifique se as expressões `instanceof Pessoa`, `instanceof Funcionario` e `instanceof ChefeDeDepartamento` retornariam verdadeiro ou falso.

Exercício 8.7 ★ ★

Considere a classe `DataHora` (Listagem 8.1 no livro) e as classes `Data` e `Hora` cujas instâncias são usadas na sua composição. Escreva, se ainda não existir na classe `Data`, um método `éIgual` que receba como argumento uma instância da própria classe `Data` e retorne o valor booleano `true` se a data representada for igual à data passada. Faça o mesmo para a classe `Hora`. Escreva também na classe `DataHora` um método `éIgual` que receba outra instância da própria classe `DataHora` como argumento e que seja executado delegando a comparação aos métodos das classes `Data` e `Hora`.

Exercício 8.8 ★ ★

Escreva a classe `LampadaFluorescente` como sendo herdeira da classe `Lampada` (exercício 2.7). A classe `LampadaFluorescente` deve ter um campo que represente o comprimento da lâmpada em centímetros. Crie nessa classe um construtor para inicializar os seus atributos.

Exercício 8.9 ★ ★

Escreva duas versões da classe `DataHora` (Listagem 8.1 no livro): uma que usa o mecanismo de herança e que herda da classe `Data` e contém um campo que é uma instância da classe `Hora` e outra versão que herda da classe `Hora` e contém um campo que é uma instância da classe `Data`. Existem diferenças ou vantagens claras entre as duas abordagens classes?

Exercício 8.10 ★ ★

Modifique os métodos da classe `EmprestimoBancario` (listagem 8.22) para que o empréstimo calculado para uma instância que herde da classe `Aluno` seja constante e igual a 500 reais.

Exercício 8.11 ★ ★

Escreva uma classe `EventoDelegacao` que seja baseada na classe `DataHora` e que contenha um campo para indicar qual o evento que ela representa (use uma string para isso). Use o mecanismo de delegação para criar a classe `EventoDelegacao`.

Exercício 8.12 ★ ★

Escreva uma classe `EventoHeranca` que seja baseada na classe `DataHora` e que contenha um campo para indicar qual evento ela representa (use uma string para isso). Use o mecanismo de herança para criar a classe `EventoHeranca`. Veja também o exercício 8.11.

Exercício 8.13 ★ ★

Escreva outro construtor para a classe `ChefeDeDepartamento` (listagem 8.10 no livro) que em vez de receber e repassar os dados separados de um funcionário (`nome`, `identidade`, `admissão` e `nascimento`), receba uma instância da classe `Funcionario` já construída. *Dica:* é possível passar para o construtor de uma classe ancestral uma instância da própria classe: neste exemplo, é possível passar para o construtor de `Funcionario` uma instância de `Funcionario`.

Exercício 8.14 ★ ★

Escreva uma classe `Diretor` que herde da classe `ChefeDeDepartamento` e que contenha campos adicionais para representar a data de promoção ao cargo.

Exercício 8.15 ★ ★

Escreva a classe `PacienteDeClinica`, descrita na figura 8.1.

Exercício 8.16 ★ ★

Identifique e explique o(s) erro(s) nas classes abaixo.

```
1  public class DataParaAgenda
2    {
3    private byte dia, mês;
4    private short ano;
5
6    public DataParaAgenda(byte d, byte m, short a)
7      {
8      dia = d;
9      mês = m;
10     ano = a;
11     }
12
13   public String toString()
14     {
15     return dia + "/" + mês + "/" + ano;
16     }
17   }
```

```
1  public class DataHoraParaAgenda extends DataParaAgenda
2    {
3    private Hora hora;
4
5    public DataHora(byte d, byte m, short a, byte hor, byte min, byte seg)
6      {
7      super(d, m, a);
8      hora = new Hora(hor, min, seg);
9      }
10
11   public String toString()
12     {
13     return super.toString() + " " + hora.toString();
14     }
15   }
```

```
1  public class EntradaNaAgenda extends DataHora
2    {
3
4    private String evento;
5
6    public EntradaNaAgenda(byte d, byte m, short a, byte hor, byte min, byte seg,
7                           String ev)
8      {
9      super.
10     super(d, m, a);
11     super(d, m, a, hor, min, seg);
12     evento = ev;
13     }
14
15   public String toString()
16     {
17     return super.super.toString() + ":" + super.toString() + " -> " + ev;
18     }
19   }
```

Exercício 8.17 ★ ★

Modifique o método `calculaEmprestimo` da classe `EmprestimoBancario` (listagem 8.23 no livro) de forma que as instâncias sejam verificadas na ordem inversa da presente, isto é, verificando primeiro se `p` é uma instância de `Pessoa`, depois de `Funcionario` e depois de `ChefeDeDepartamento`. Compile e execute o programa, e explique o resultado da modificação.

Exercício 8.18 ★ ★ ★

Usando o exercício 8.7 como base, crie na classe `Data` os métodos `éAntesDe` e `éDepoisDe` que retornam `true` se a data passada como argumento for respectivamente posterior e anterior à data representada. Escreva também esses métodos na classe `Hora`. Escreva também na classe `DataHora` os métodos `éAntesDe` e `éDepoisDe` que recebem uma instância da própria classe `DataHora` como argumento e que sejam executados delegando a comparação aos métodos das classes `Data` e `Hora`.

Exercício 8.19 ★ ★ ★

Escreva as classes `LivroLivraria` e `LivroBiblioteca` que herdam da classe `Livro`. Quais as diferenças entre as duas classes, e que campos elas têm em comum? *Dica:* Os campos em comum devem ser preferencialmente representados pela classe ancestral. Veja também os exercícios 2.20, 2.21 e 2.22.

Capítulo 9

Classes abstratas e interfaces

O mecanismo de herança permite a organização de classes em estruturas hierárquicas onde uma classe herdeira *é-um-tipo-de* classe ancestral, mais genérica. Uma forma de generalizar ainda mais classes ancestrais é declará-las como abstratas – assim estas classes genéricas não podem ser instanciadas, mas definem um conjunto de métodos (contrato) que devem ser obrigatoriamente implementado por classes herdeiras, mais específicas, para que estas possam ser instanciadas.

Este capítulo apresenta os conceitos de classes abstratas e interfaces, outro mecanismo de criação de contratos. Implicações do uso destas técnicas nos mecanismos de herança serão apresentadas e exemplificadas.

9.1 Exercícios do Capítulo 9

Exercício 9.1 ★
Explique com suas palavras por que uma classe abstrata não pode ser instanciada.

Exercício 9.2 ★
Explique com suas palavras por que uma interface não pode ter métodos estáticos.

Exercício 9.3 ★
Explique, com suas palavras, por que interfaces não podem ter construtores.

Exercício 9.4 ★
Explique com suas palavras por que não podemos ter construtores declarados com a palavra-chave `abstract`.

Exercício 9.5 ★
O que ocorrerá se declararmos a classe `ContaSimples` (listagem 9.2 no livro) como sendo abstrata?

Exercício 9.6 ★

O que aconteceria se tentássemos chamar o método `exibeTodosOsDados` da classe `USaObjetosGeometricos` (listagem 9.9 no livro) passando para ele, como argumento, uma string? Explique.

Exercício 9.7 ★

Identifique e explique o(s) erro(s) na classe abaixo.

```
1  public class Produto
2    {
3    private String identificação;
4    private double custoDeFabricação;
5
6    Produto(String i, double c)
7      {
8      identificação = i;
9      custoDeFabricação = c;
10     }
11
12   abstract public String toString();
13   abstract public void novoCusto(double nc);
14   }
```

Exercício 9.8 ★

Identifique e explique o(s) erro(s) na classe abaixo.

```
1  public abstract class Dispositivo
2    {
3
4    private String nome;
5    private long capacidadeEmBytes;
6
7    public Dispositivo(String n, long c)
8      {
9      nome = n;
10     capacidadeEmBytes = c;
11     }
12
13   abstract public String toString();
14   abstract public double capacidadeEmMegabytes();
15   }
```

```
1  public class DiscoOtico extends Dispositivo
2    {
3
4    public DiscoOtico(long c)
5      {
6      super("Disco Ótico", 241172480L);
7      }
8
9    public String toString()
10     {
11     return "Dispositivo:" + nome + " Capacidade:" + c;
12     }
13   }
```

Exercício 9.9 ★

Por que campos em interfaces devem ser inicializados em sua declaração? Explique.

Exercício 9.10 ★

Na classe `UsaObjetosGeometricosComPolimorfismo` (listagem 9.10 no livro) declaramos quatro referências: `o1`, `o2`, `o3` e `o4`. Podemos associar estas refereências a instâncias de `ObjetoGeoetrico`? Por quê?

Exercício 9.11 ★

A classe `ContaBancaria` (listagem 9.1 no livro) é declarada como abstrata. A classe `ContaEspecial` (listagem 9.3 no livro) que herda de `ContaBancaria` não é abstrata. Se criarmos uma classe `ContaSuperEspecial` que herda de `ContaBancaria`, seremos obrigados a escrever algum método em `ContaSuperEspecial` para atender aos requisitos de herança de classes abstratas?

Exercício 9.12 ★ ★

Identifique e explique o(s) erro(s) na classe abaixo.

```
1  public interface Resetavel
2    {
3    public void reseta();
4    }
```

```
1  public interface Modificavel
2    {
3    public void reseta(int origem);
4    public void modifica(int tamanho);
5    }
```

```
1  public class Contador implements Resetavel, Modificavel
2    {
3    int valor;
4
5    public void reseta()
6      {
7      valor = 1;
8      }
9
10   public void modifica(int tam)
11     {
12     valor = tam;
13     }
14   }
```

Exercício 9.13 ★ ★

Reescreva a classe `ConstantesMatematicas` (mostrada na listagem 5.6) como uma interface. Demonstre seu uso através de uma classe que implementa essa interface e tem acesso aos seus campos.

Exercício 9.14 ★ ★

Escreva a classe `Quadrado`, que implementa a interface `ObjetoGeometrico`. Use como referência as classes `Circulo` e `Retangulo` (listagens 9.7 e 9.8 no livro).

Exercício 9.15 ★ ★

Escreva outra versão da classe `Quadrado` como herdeira da classe `Retangulo` (listagem 9.8 no livro). Quais métodos devem ser sobrescritos?

Exercício 9.16 ★ ★

Escreva a classe `Trapezio`, que implementa a interface `ObjetoGeometrico`. Use como referência a classe `Retangulo` (listagem 9.8 no livro).

Exercício 9.17 ★ ★

Inclua, na interface `ObjetoGeometrico` (listagem 9.5 no livro), a declaração do método `public String toString`, que fará com que todos as classes que implementarem a interface `ObjetoGeometrico` sejam obrigadas a implementar o método `toString`. Deliberadamente, deixe de implementar esse método em uma classe que implemente a interface (pode-se usar como exemplo uma das classes `Circulo` ou `Retangulo`, retirando-se delas o método `toString`). Escreva uma aplicação que use uma dessas classes. O que acontece se o método `toString` não for implementado apesar de estar declarado na interface? Explique.

Exercício 9.18 ★ ★

Podemos ter na interface `Escalavel` (listagem 9.11 no livro) um método `reduz` que chame o método `amplia` com valores entre zero e um? Explique.

Exercício 9.19 ★ ★ ★

Reescreva a classe `Lampada` como sendo uma interface. Escreva as classes `LampadaIncandescente` e `LampadaDeNatal` que implementam a interface `Lampada`. Faça com que a classe `LampadaDeNatal` contenha um campo `cor` como sendo de um enumerador `Cor`, que contém constantes representando diversas cores padrão. Veja também o exercício 2.7.

Exercício 9.20 ★ ★ ★

Escreva para a interface `ObjetoGeometrico` (listagem 9.5 no livro), a declaração do método `clona`, que não recebe argumentos e retorna uma instância de `ObjetoGeometrico`. Ao criar essa declaração, todas as classes que implementam `ObjetoGeometrico` deverão implementar esse método. Crie esse método nas classes que implementam a interface. *Dica:* O método deverá ter o mesmo tipo de retorno do declarado na interface. Por exemplo, a classe `Circulo` deverá ter um método `public ObjetoGeometrico clona`, que deve criar e retornar um clone do círculo sendo encapsulado. Não existe problema em declarar um método como devendo retornar `ObjetoGeometrico` e retornando `Circulo`, já que, por causa das regras de polimorfismo, `Circulo` *é-um-tipo-de* `ObjetoGeometrico`.

Capítulo 10

Arrays em Java

Arrays são estruturas de dados simples que permitem a representação de várias variáveis ou referências em uma única estrutura, usando índices para referenciar as variáveis ou instâncias. Arrays podem ter mais de uma dimensão, podendo ser usados para representar uma ampla variedade de abstrações.

Este capítulo apresenta como declarar, usar e encapsular arrays em classes e aplicações em Java. O capítulo também apresenta uma forma específica de declaração de laços para percorrer todos os elementos de um array e informações sobre os argumentos para o método `main` que podem ser inicializados para uso em aplicações de linha de comando.

10.1 Exercícios do Capítulo 10

Exercício 10.1 ★

Após a execução da linha `double[] inversos = new double[100]; inversos[40] = 1./40.;` em um método qualquer, quais das opções abaixo serão verdadeiras?

A. O array `inversos` tem 99 posições.
B. `inversos[0]` é igual a `Double.NaN`.
C. `inversos[40]` é igual a zero.
D. Existem 99 valores no array iguais a zero.
E. `inversos[100]` é igual a `null`.

Exercício 10.2 ★

Após a execução da linha `char[] alfabeto = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I'};` em um método qualquer, quais das opções abaixo serão verdadeiras?

A. O array `alfabeto` tem nove posições.
B. O quinto elemento do array é o caracter `'F'`.
C. O décimo elemento do array é `null`.
D. O décimo elemento do array é o caracter espaço.
E. O valor de `alfabeto.length` é 8.

Exercício 10.3 ★

Considerando a declaração `float[] sequência = new float[25];`, quais das declarações abaixo serão corretas (isto é, poderão ser compiladas e executadas sem problemas)?

A. `sequência[0] = 0;`
B. `sequência[1] = 1;`
C. `sequência[1.5] = 1.5;`
D. `sequência[-1] = -1;`
E. `sequência[23] = "23";`
F. `sequência[24] = 24;`
G. `sequência[25] = 25;`

Exercício 10.4 ★

Qual será o conteúdo dos arrays declarados na aplicação abaixo ao término da execução do método `main`?

```
1  public class Arrays1
2    {
3    public static void main(String[] argumentos)
4      {
5      double[] valores = {1,2,3,4,5,6};
6      double[] primeiraCópia = valores;
7      double[] segundaCópia = valores;
8      primeiraCópia[1] = 1;
9      segundaCópia[2] = valores[0]+primeiraCópia[1];
10     primeiraCópia[3] = valores[1]+segundaCópia[2];
11     valores[4] = primeiraCópia[2]+segundaCópia[3];
12     valores[5] = segundaCópia[3]+primeiraCópia[4];
13     }
14   }
```

Exercício 10.5 ★

Qual será o conteúdo dos arrays declarados na aplicação abaixo ao término da execução do método `main`?

```
1  public class Arrays2
2    {
3    public static void main(String[] argumentos)
4      {
5      float[] constantes = {100f,10f,1f,0.1f,0.01f,0.001f};
6      float[] duplicata = constantes;
7      resetaArray(duplicata);
8      }
9    private static void resetaArray(float[] array)
10     {
11     for(int índice=0;índice<array.length;índice++)
12       array[índice] = 0f;
13     }
14   }
```

Exercício 10.6 ★

O método abaixo pode ser um método da classe `ArrayDeFloats` (listagem 10.5 no livro)? Explique.

```
1 public void mudaTamanho(int novoTamanho)
2   {
3   array.length = novoTamanho;
4   }
```

Exercício 10.7 ★

Escreva uma aplicação em Java que declare e inicialize um vetor de booleanos (lendo-os do teclado), e calcule quantos elementos são iguais a `true`.

Exercício 10.8 ★

Escreva uma classe em Java que encapsule um array de 12 bytes, onde cada elemento do array contém o número de dias no mês correspondente, desconsiderando se o ano é bissexto ou não. Por exemplo, o elemento 0 do array (correspondente a janeiro) deve valer 31. Escreva um método que retorne o valor encapsulado para determinado mês.

Exercício 10.9 ★

Escreva um programa que declare um array bidimensional chamado `tabuada` de 10×10 posições e preencha os elementos do array com os valores da tabuada da soma para aquele elemento, de forma que, por exemplo, o elemento `tabuada[7][9]` valha 16. Use o tipo de dado mais adequado para este array.

Exercício 10.10 ★

Escreva para a classe `MatrizDeDoubles` (listagem 10.12 no livro) métodos que permitam o acesso e a modificação dos valores individuais da matriz encapsulada. *Dica:* Veja a classe `ArrayDeFloats` (listagem 10.5 no livro).

Exercício 10.11 ★

Modifique a classe `ArrayDeFloats` (listagem 10.5 no livro), criando outro construtor que receba como argumentos o tamanho do array a ser criado e um valor constante, e inicialize os elementos do array com essa constante.

Exercício 10.12 ★ ★

Escreva um programa que declare um array de cinco dimensões chamado `tabuada` de $10 \times 10 \times 10 \times 10 \times 10$ posições e preencha os elementos do array com os valores da tabuada da multiplicação para aquele elemento, de forma que, por exemplo, o elemento `tabuada[6][5][2][1][4]` valha 240. *Dica:* Use o exercício 10.9 como base.

Exercício 10.13 ★ ★

Crie na classe `ArrayDeFloats` (listagem 10.5 no livro) um método `éIgual` que receba outra instância da classe como argumento e retorne `true` se os tamanhos e valores do array encapsulado e do passado como argumento forem iguais.

Exercício 10.14 ★ ★

Crie na classe `ArrayDeFloats` (listagem 10.5 no livro) um método `soma` que some (acumule) a cada um dos elementos do array encapsulado uma constante do tipo `float` que será passada como argumento.

Exercício 10.15 ★ ★

Crie na classe `ArrayDeFloats` (listagem 10.5 no livro) um método `total` que some todos os valores do array, retornando o resultado dessa somatória.

Exercício 10.16 ★ ★

Crie na classe `MatrizDeDoubles` (listagem 10.12 no livro) um método `éIgual` que receba outra instância da classe como argumento e retorne `true` se os tamanhos e valores da matriz encapsulada e da passada como argumento forem iguais.

Exercício 10.17 ★ ★

Crie na classe `ArrayDeFloats` o método `existe`, que recebe um valor do tipo `float` como argumento e retorna o booleano `true` se o valor passado como argumento existir no array encapsulado.

Exercício 10.18 ★ ★

Escreva uma classe que encapsule uma matriz de tamanho 2×2 de valores do tipo float ou double, usando um array de duas dimensões. Nessa classe, escreva um método que calcule o determinante da matriz encapsulada e um método que permita a impressão formatada dos seus valores (duas linhas com dois valores cada).
Dica: Se a matriz M é dada por

$$\begin{pmatrix} x_{00} & x_{01} \\ x_{10} & x_{11} \end{pmatrix}$$

então o determinante é calculado como $(x_{00} \times x_{11}) - (x_{01} \times x_{10})$.

Exercício 10.19 ★ ★

Usando o exercício 10.18 como base, escreva uma classe que encapsule uma matriz 3×3 usando um array de duas dimensões. Nessa classe, escreva um método que calcule o determinante da matriz encapsulada e um método que permita a impressão formatada dos seus valores (três linhas com três valores cada).
Dica: Se a matriz M é dada por

$$\begin{pmatrix} x_{00} & x_{01} & x_{02} \\ x_{10} & x_{11} & x_{12} \\ x_{20} & x_{21} & x_{22} \end{pmatrix}$$

então o determinante é calculado como $(x_{00} \times x_{11} \times x_{12}) + (x_{01} \times x_{12} \times x_{20}) + (x_{02} \times x_{10} \times x_{21}) - (x_{00} \times x_{12} \times x_{21}) - (x_{01} \times x_{10} \times x_{22}) - (x_{02} \times x_{11} \times x_{20})$.

Exercício 10.20 ★ ★ ★

Crie uma classe que implementa um *array de escrita única* de valores de um tipo numérico qualquer. Um array de escrita única é um array cujos elementos só possam ser modificados uma única vez. Para a implementação, devemos ter, para cada elemento do array, um valor booleano associado que diz se o elemento pode ser modificado ou não. Quando instâncias dessa classe forem criadas, todos os elementos do array poderão ser modificados, mas assim que um elemento for modificado pela primeira vez o seu valor booleano associado será modificado de forma que da próxima vez que o elemento for modificado nada ocorrerá. Crie também uma aplicação que demonstre o uso desta classe.

Exercício 10.21 ★ ★ ★

Usando o exercício 10.20 como base, crie uma classe que implemente um array cujos elementos só podem ser modificados um certo número limitado de vezes. Quando instâncias dessa classe forem criadas, elas devem receber um valor que diz quantas vezes um dado elemento do array encapsulado pode ser modificado. *Dica:* O valor deve ser o mesmo para todos os elementos do array, mas cada elemento individual do array poderá ser modificado o número de vezes que for especificado.

Exercício 10.22 ★ ★ ★

Usando a classe `CalculadoraDeLinhaDeComando` (listagem 10.17 no livro) como base, escreva uma aplicação em Java que processe a linha de comando, recebendo três ou mais argumentos, de forma que o primeiro deva ser o operador `'+'` ou `'*'`, e os argumentos do segundo em diante devam ser valores numéricos. A aplicação deve efetuar a soma ou multiplicação de todos os argumentos passados e mostrar o resultado. Se, por exemplo, os argumentos `+ 2 4 1 5` forem passados, a aplicação deverá imprimir o resultado `12`. Se os argumentos `* 2 4 1 5` forem passados, a aplicação deverá imprimir o resultado `40`.

Exercício 10.23 ★ ★ ★

Escreva uma classe `MatrizDeNumerosComplexos`, com a mesma funcionalidade da classe `MatrizDeDoubles` (listagem 10.12 no livro), que encapsule uma matriz bidimensional de instâncias da classe `NumeroComplexo` (veja os exercícios 4.13 e 10.10).

Exercício 10.24 ★ ★ ★

Escreva na classe `ArrayDeFloats` o método `reverte`, que reverte a ordem dos elementos do array encapsulado, de forma que o primeiro passe a ser o último e vice-versa. Por exemplo, se o array encapsulado for $\{9, 9, 2, 7, 0, 5\}$, depois da execução do método ele será $\{5, 0, 7, 2, 9, 9\}$. *Dica:* Existem duas abordagens para a solução desse problema, uma que modifica os valores do array encapsulado e outra que cria uma nova instância; considere implementar as duas.

Exercício 10.25 ★ ★ ★

Escreva para a classe `ArrayDeFloats` o método `éPalíndromo`, que retorna `true` se o array encapsulado for palíndromo. Um array palíndromo é aquele que pode ser lido do início para o fim e do fim para o início, da mesma forma. Por exemplo, o array $(2, -4, 9, 0, 9, -4, 2)$ é palíndromo. *Dica:* Existem ao menos duas soluções para o problema, uma que verifica os dados do array, comparando-os, e outra que usa a solução do exercício 10.24.

Exercício 10.26 ★ ★ ★

Considere um bilhete de loteria instantânea (raspadinha), que contém seis valores numéricos. Se três desses valores forem iguais, o jogador receberá o valor que apareceu repetido, caso contrário receberá zero. Escreva uma classe `Raspadinha` em Java que simule os valores da raspadinha com um array de inteiros, e calcule o prêmio para o vencedor. Por exemplo, se o array encapsulado for `(1,5,10,500,5,5)`, o vencedor deverá receber cinco reais, e se o array encapsulado for `(10,5,10,100,1,5)` o vencedor não deverá receber nada. *Dica:* existe mais de uma maneira de implementar o algoritmo que verifica o valor a ser pago – uma coleção enorme de `ifs` **não** é a mais eficiente!

Exercício 10.27 ★ ★ ★ ★

Crie uma classe que represente um jogo da velha, usando uma matriz de duas dimensões para representar as posições do jogo. A matriz deve ser alocada no construtor da classe, ter o tamanho 3×3 e ser de um tipo que suporte três estados possíveis: vazio, preenchido com `'O'` e preenchido com `'X'` (use um enumerador). A classe deve poder ser usada para jogos com dois jogadores.
Dica: A classe deve ter os seguintes métodos:

- `jogaO`, que aceita dois valores que são as coordenadas onde um `'O'` será jogado, e marca na matriz a posição **somente** se esta estiver livre.
- `jogaX`, que aceita dois valores que são as coordenadas onde um `'X'` será jogado, e marca na matriz a posição **somente** se esta estiver livre.
- `verifica`, que verifica a matriz para ver se existe algum ganhador (esse método deve verificar se existem três marcas iguais que não sejam vazias em uma horizontal, vertical ou diagonal da matriz).
- `toString`, que retornará uma string com a representação gráfica do jogo com as posições atuais.

Escreva também um programa que use a classe. Este programa deve executar um laço no qual fica perguntando as posições para os jogadores alternadamente, enquanto não houver vitória, desistência ou acabarem as posições vazias da matriz.

Exercício 10.28 ★ ★ ★ ★

Um *quadrado mágico* é uma matriz quadrada de valores inteiros onde a soma dos valores em cada linha, coluna ou diagonal principal é a mesma, e não existem elementos repetidos. Por exemplo, a matriz

$$\begin{matrix} 2 & 7 & 6 \\ 9 & 5 & 1 \\ 4 & 3 & 8 \end{matrix}$$

representa um quadrado mágico. Escreva uma classe `QuadradoMagico` que tenha o método estático `éQuadradoMágico` que retorne `true` caso a matriz, passada como argumento para o método, represente um quadrado mágico.

Exercício 10.29 ★ ★ ★ ★

O *crivo de Eratóstenes* é um algoritmo usado para identificar números primos. O algoritmo (apresentado aqui da maneira mais simples) primeiro declara um array de N posições de valores booleanos, todos iguais a `true` (considerando que em princípio qualquer número pode ser primo). O algoritmo, em seguida, marca todos os elementos do array cujos índices são múltiplos de 2 e maiores que o próprio 2 como `false`, indicando que nenhum múltiplo de dois pode ser primo. O algoritmo repete esse último procedimento para todos os valores múltiplos de 3 e maiores que 3, depois para todos os valores múltiplos de 4 e maiores que 4, e assim sucessivamente, até chegar até $N/2$. Ao final, os índices dos elementos do array que valerem `false` serão valores não-primos, e os que ainda valerem `true` depois da execução do algoritmo serão primos. *Dica:* Para entender melhor o algoritmo, rode uma simulação em papel antes. Esse algoritmo pode ser consideravelmente otimizado, tente fazer isso.

Exercício 10.30 ★ ★ ★ ★

O algoritmo de ordenação de bolha (*bubblesort*) serve para colocar os elementos de uma lista em ordem crescente ou decrescente. O algoritmo é como segue:

1. Percorra o array da primeira posição até a penúltima, chamando o elemento do array nessa posição de P;
2. Para cada P percorra o mesmo array da posição seguinte de P até a última posição do array, chamando o elemento nessa posição de Q.
3. Se $P > Q$, troque os valores de P e Q.

Uma simulação simples do algoritmo é mostrada abaixo. Consideremos o array $\{7, 5, 1, 3\}$ que deve ser ordenado de forma crescente. Os passos da simulação seriam:

1. O array vale, inicialmente, $\{7, 5, 1, 3\}$.
2. A posição de P no array é 0, e seu valor é 7. A posição de Q no array é 1, e seu valor é 5. Como $P > Q$, os valores são trocados. Agora o array é $\{5, 7, 1, 3\}$.
3. A posição de P no array ainda é 0, e seu valor é 5. A posição de Q no array é 2, e seu valor é 1. Como $P > Q$, os valores são trocados. Agora o array é $\{1, 7, 5, 3\}$.
4. A posição de P no array ainda é 0, e seu valor é 1. A posição de Q no array é 3, e seu valor é 3. Como P não é maior que Q, os valores não são trocados. O array permanece sendo $\{1, 7, 5, 3\}$.
5. A posição de P no array agora é 1 (próxima iteração do laço externo), e seu valor é 7. A posição de Q no array é 2, e seu valor é 5. Como $P > Q$, os valores são trocados. O array agora é $\{1, 5, 7, 3\}$.
6. A posição de P no array ainda é 1, e seu valor é 5. A posição de Q no array é 3, e seu valor é 3. Como $P > Q$, os valores são trocados. Agora o array é $\{1, 3, 7, 5\}$.
7. A posição de P no array agora é 2 (próxima iteração do laço externo), e seu valor é 7. A posição de Q no array é 3, e seu valor é 5. Como $P > Q$, os valores são trocados. Agora o array é $\{1, 3, 5, 7\}$.
8. O array vale, finalmente, $\{1, 3, 5, 7\}$

Implemente na classe `ArrayDeFloats` (listagem 10.5 no livro) o método `ordenaCrescentePorBolha` que implemente o algoritmo descrito. Crie também o método `ordenaDecrescentePorBolha` que implemente um algoritmo que ordene de forma decrescente: a única diferença do algoritmo que ordena de forma crescente é que P e Q deverão ser trocados se $P < Q$. De que outra forma podemos resolver este segundo problema?

Exercício 10.31 ★ ★ ★ ★

Um outro algoritmo de ordenação bastante conhecido é o algoritmo de ordenação por seleção (*selection sort*). Esse algoritmo funciona de maneira simples: primeiro, cria um array do mesmo tamanho do array a ser ordenado, depois percorre sucessivamente o array original procurando o menor valor. Esse menor valor é inserido no array de destino, e o valor é removido do array original. O algoritmo repete esses passos até que todos os elementos do array original tenham sido removidos. Como arrays em Java têm tamanho imutável, não é simples remover um elemento de um array de forma que o tamanho do array encolha. Para facilitar a implementação deste algoritmo, usamos o conceito de marcadores (*flags*) – valores que são colocados no lugar do valor removido e que indicam a remoção.

Uma simulação desse algoritmo que ordena o array $\{41, -72, -25, 25, -3, -85, 19, -63, 44\}$ é mostrada abaixo (onde os caracteres *** indicam que o elemento já foi ordenado e não deve ser considerado pelo passo, e os caracteres ### indicam o elemento que foi retirado por último).

Passo	Array original									Array ordenado
0	41	-72	-25	25	-3	-85	19	-63	44	{} (vazio)
1	41	-72	-25	25	-3	###	19	-63	44	$\{-85\}$
2	41	###	-25	25	-3	***	19	-63	44	$\{-85, -72\}$
3	41	***	-25	25	-3	***	19	###	44	$\{-85, -72, -63\}$
4	41	***	###	25	-3	***	19	***	44	$\{-85, -72, -63, -25\}$
5	41	***	***	25	###	***	19	***	44	$\{-85, -72, -63, -25, -3\}$
6	41	***	***	25	***	***	###	***	44	$\{-85, -72, -63, -25, -3, 19\}$
7	41	***	***	###	***	***	***	***	44	$\{-85, -72, -63, -25, -3, 19, 25\}$
8	###	***	***	***	***	***	***	***	44	$\{-85, -72, -63, -25, -3, 19, 25, 41\}$
9	***	***	***	***	***	***	***	***	###	$\{-85, -72, -63, -25, -3, 19, 25, 41, 44\}$

Não podemos simplesmente usar um valor numérico qualquer como marcador, precisando usar um valor que seja claramente um valor não-válido. Se o array for de valores do tipo `float` poderemos usar, então, o valor `Float.NaN`, que não pode ser comparado com nenhum outro valor. Alternativamente podemos usar um array auxiliar de valores do tipo `boolean` para marcar os valores do array original.

Crie, para a classe `ArrayDeFloats` (listagem 10.5 no livro), o método `ordenaCrescentePorSeleção` que implemente o algoritmo descrito. Escreva também o método `ordenaDecrescentePorSeleção` que implemente um algoritmo que ordene de forma decrescente: a única diferença do algoritmo que ordena de forma crescente é que em vez de procurar o menor valor no array original, deveremos procurar o maior valor. De que outra forma podemos resolver este segundo problema?

Exercício 10.32 ★ ★ ★ ★

O jogo japonês *Gomoku* é jogado por duas pessoas em um tabuleiro quadrado de tamanho 19×19. Cada pessoa recebe um conjunto de peças pretas e brancas que devem ser colocadas alternadamente no tabuleiro, na posição que o jogador desejar. Ganha o jogo o primeiro jogador que conseguir colocar cinco de suas peças em uma linha reta horizontal, vertical ou diagonal.
Crie uma classe em Java que represente um jogo de *Gomoku*, usando uma matriz de duas dimensões para representar as posições do jogo. A matriz deve ser alocada no construtor da classe, ter o tamanho 19×19 e ser de um tipo que suporte três estados possíveis: vazio, preenchido com peça preta e preenchido com peça branca (use um enumerador para isto!). A classe deve poder ser usada para jogos com dois jogadores.
A classe deve ter os seguintes métodos:

- `jogaPreta`, que aceita dois valores que são as coordenadas onde uma peça preta será jogada, e marca na matriz a posição **somente** se esta estiver livre.
- `jogaBranca`, que aceita dois valores que são as coordenadas onde uma peça branca será jogada, e marca na matriz a posição **somente** se esta estiver livre.
- `verifica`, que verifica a matriz para ver se existe algum ganhador (este método deve verificar se existem cinco peças iguais que não sejam vazias em uma horizontal, vertical ou diagonal da matriz, depois de **cada** jogada feita).
- `toString`, que retornará uma string com a representação gráfica do jogo com as posições atuais.

Escreva também um programa que use a classe. Esse programa deve executar um laço no qual fica perguntando as posições para os jogadores alternadamente, enquanto não houver vitória, desistência ou acabarem as posições vazias da matriz.
Dica: O algoritmo do jogo não é tão diferente do jogo da velha (exercício 10.27), exceto pelo método `verifica`. Esse método pode, para cada posição do array bidimensional, ver se existem linhas de cinco peças iguais contadas a partir da posição sendo procurada. O único cuidado adicional é garantir que o algoritmo não procurará peças fora do tabuleiro.

Exercício 10.33 ★ ★ ★ ★

Considere o método `existe`, pedido como solução do exercício 10.17. Esse método é potencialmente ineficiente, pois se o elemento que desejamos encontrar estiver próximo do fim do array, o método deverá percorrer grande parte do array. Se o array estiver ordenado (veja o exercício 10.30) existe um algoritmo muito mais eficiente de busca, chamado *busca binária*, que funciona através da divisão do array em áreas nas quais a busca pode ser bem-sucedida. Esse algoritmo retorna o índice do elemento procurado ou `-1` se o elemento não existir no array, e funciona com os seguintes passos:

1. Iniciamos o algoritmo estabelecendo a área de pesquisa, fazendo com que `primeiro` seja igual a `0` e `último` seja igual ao último índice válido do array.
2. Verificamos se o elemento procurado é igual a `array[primeiro]`. Se for, retornamos `primeiro`.
3. Verificamos se o elemento procurado é igual a `array[último]`. Se for, retornamos `último`.
4. Verificamos a diferença entre `último` e `primeiro` – se for igual a 1, significa que o array sendo procurado somente tem duas posições, mas como o valor procurado não é nenhum dos dois extremos, ele não existe no array, e o algoritmo retorna zero.
5. Calculamos a posição central do array como `primeiro+(último-primeiro)/2` e a armazenamos em `índiceCentral`. Esse cálculo deve ser feito usando valores inteiros, pois o índice de um array é sempre inteiro.
6. Se o valor procurado for maior do que `array[índiceCentral]`, chamamos o algoritmo recursivamente, fazendo com que `primeiro` seja igual a `índiceCentral`.
7. Se o valor procurado não for maior do que `array[índiceCentral]`, chamamos o algoritmo recursivamente, fazendo com que `último` seja igual a `índiceCentral`.

Uma simulação do algoritmo, usando o array $\{1, 7, 8, 10, 11, 13, 21, 39, 41, 46, 47, 50, 51, 54, 58, 61, 62, 67, 80, 90, 96, 97, 99, 100\}$, e procurando neste o valor `80`, é mostrada abaixo:

1. O algoritmo inicia a busca entre os valores 1 e 100, com `primeiro` valendo 0 e `último` valendo 23.
2. Como o valor procurado não é igual a 1 nem igual a 100, calculamos `índiceCentral` como sendo 11. O valor de `array[índiceCentral]` é 50. Como $80 > 50$, chamamos o algoritmo recursivamente com `primeiro` valendo 11 e `último` valendo 23.
3. Como o valor procurado não é igual a 50 nem igual a 100, calculamos `índiceCentral` como sendo 17. O valor de `array[índiceCentral]` é 67. Como $80 > 67$, chamamos o algoritmo recursivamente com `primeiro` valendo 17 e `último` valendo 23.
4. Como o valor procurado não é igual a 67 nem igual a 100, calculamos `índiceCentral` como sendo 20. O valor de `array[índiceCentral]` é 96. Como 80 não é maior que 96, chamamos o algoritmo recursivamente com `primeiro` valendo 17 e `último` valendo 20.
5. Como o valor procurado não é igual a 67 nem igual a 96, calculamos `índiceCentral` como sendo 18. O valor de `array[índiceCentral]` é 80. Como 80 não é maior que 80, chamamos o algoritmo recursivamente com `primeiro` valendo 17 e `último` valendo 18.
6. Como o valor procurado é igual a `80`, retornamos `último`, que é igual a 18.

Implemente esse algoritmo de busca para a classe `ArrayDeFloats`. *Dica:* Garanta que o array estará ordenado antes de fazer a busca.

Exercício 10.34 ★ ★ ★ ★

Escreva para a classe `MatrizDeDoubles` (listagem 10.12) um método `multiplica`, que aceite outra instância da classe `MatrizDeDoubles` como argumento e multiplique a matriz encapsulada pela passada como argumento, retornando ma nova matriz.

A multiplicação de uma matriz A de dimensões A_lxA_c por uma matriz B de dimensões B_lxB_c só pode ser efetuada se os valores A_c e B_l forem iguais, ou seja, se o número de colunas da matriz A for igual ao número de linhas da matriz B, caso contrário o método deverá retornar `null`. O resultado será uma matriz C de dimensões A_lxB_c. O esquema gráfico da multiplicação é mostrado abaixo:

$$\begin{bmatrix} A_{(0,0)} & A_{(0,1)} & A_{(0,2)} & A_{(0,3)} \\ A_{(1,0)} & A_{(1,1)} & A_{(1,2)} & A_{(1,3)} \end{bmatrix} \begin{bmatrix} B_{(0,0)} & B_{(0,1)} & B_{(0,2)} \\ B_{(1,0)} & B_{(1,1)} & B_{(1,2)} \\ B_{(2,0)} & B_{(2,1)} & B_{(2,2)} \\ B_{(3,0)} & B_{(3,1)} & B_{(3,2)} \end{bmatrix} \begin{bmatrix} C_{(0,0)} & C_{(0,1)} & C_{(0,2)} \\ C_{(1,0)} & C_{(1,1)} & C_{(1,2)} \end{bmatrix}$$

onde o valor de $C_{(l,c)}$ será calculado como

$$C_{(l,c)} = \sum_{x=0}^{x<A_c} (A_{(l,x)} \times B_{(x,c)})$$

Exercício 10.35 ★ ★ ★ ★ ★

Considere os quadrados mágicos descritos no exercício 10.28. Uma aplicação poderia tentar criar quadrados mágicos de 3×3 posições com valores inteiros, não-repetidos e sequenciais como o mostrado. Um algoritmo de força bruta poderia testar todas as variações dos valores de 1 a 9 colocados em cada uma das nove posições da matriz e usar o método `éQuadradoMágico` da classe `QuadradoMagico` para verificar quais das combinações correspondem a um quadrado mágico, mas o número total de variações é $9^9 = 387420489$ – se dez mil combinações fossem testadas por segundo, este algoritmo demoraria mais de dez horas para testar todas!

Um algoritmo mais eficiente tentaria calcular as variações levando em conta que os valores não devem ser repetidos: se o valor 1 for colocado na posição superior esquerda da matriz, ele não deverá ser usado nas outras posições e assim em diante – dessa forma, o número de quadrados mágicos a serem testados seria $9! = 362880$ – menos do que um milésimo do valor anterior.

Um algoritmo ainda mais eficiente consideraria que nem todas as combinações de valores devem ser testadas – por exemplo, se o valor 1 está na posição superior esquerda da matriz, sabemos que a soma dos dois valores nas outras colunas da primeira linha deve ser 14, já que a soma da linha deve ser 15. Possivelmente uma solução recursiva poderia ser usada nesse algoritmo.

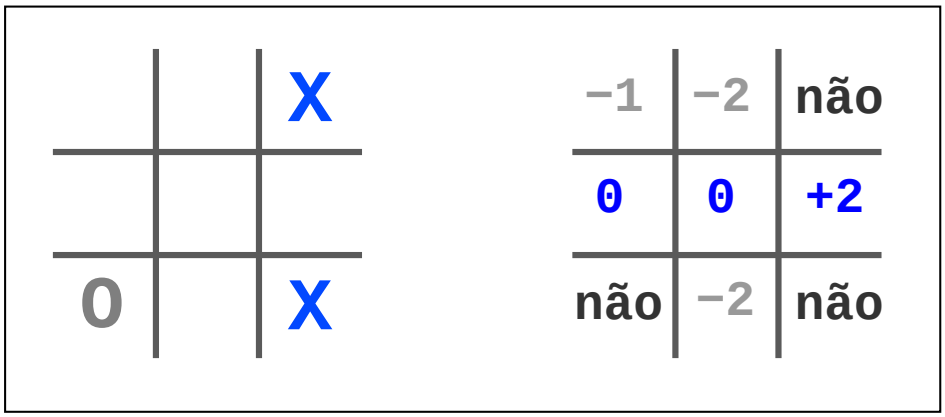
Crie, na classe `QuadradoMagico`, um método `calculaTodos` que calcule todos os quadrados mágicos de tamanho 3×3 cujos valores sejam consecutivos e não-repetidos entre 1 e 9 (com soma das linhas, colunas e diagonais igual a 15).

Exercício 10.36 ★ ★ ★ ★ ★

É possível embutir um pouco de Inteligência Artificial no jogo-da-velha do exercício 10.27, fazendo com que um jogador jogue contra o computador. Quando for a vez do computador jogar, as coordenadas onde este colocará sua peça não serão entradas via teclado: a própria classe pode escolher a melhor posição vazia para jogar sua peça com base na seguinte *heurística* (série de passos que podem levar à solução de um problema): para cada posição desocupada no tabuleiro, some:

- Mais dois pontos se a posição for a central,
- Mais um ponto se a posição for nos quatro cantos da matriz,
- Menos dois pontos se já houver uma ou mais peças do adversário na linha, coluna ou diagonal onde a posição se encontra,
- Mais quatro pontos se a posição impedir a vitória do adversário,
- Mais quatro pontos se a posição levar a uma vitória,
- Ao final do cálculo, escolher a posição que teve maior número de pontos.

Para exemplificar, considere a figura abaixo, que representa um jogo em andamento, onde o computador joga com as peças `'O'`. No exemplo mostrado, a melhor posição para o computador jogar seria aquela cujo valor é `+2`. As posições marcadas com `não` já estariam ocupadas.
Usando o exercício 10.27 como base, escreva um método `jogaComputador` que calcule a melhor posição para jogo e efetue a jogada. Outras partes da classe deverão ser reescritas, por exemplo, para permitir que o usuário decida se vai jogar com os `'X'` ou `'O'` e quem será o primeiro a jogar.
Dica: A classe pode conter outra matriz de valores inteiros, do mesmo tamanho do tabuleiro do jogo-da-velha, que será reinicializada e calculada com o algoritmo acima a cada jogada do computador.

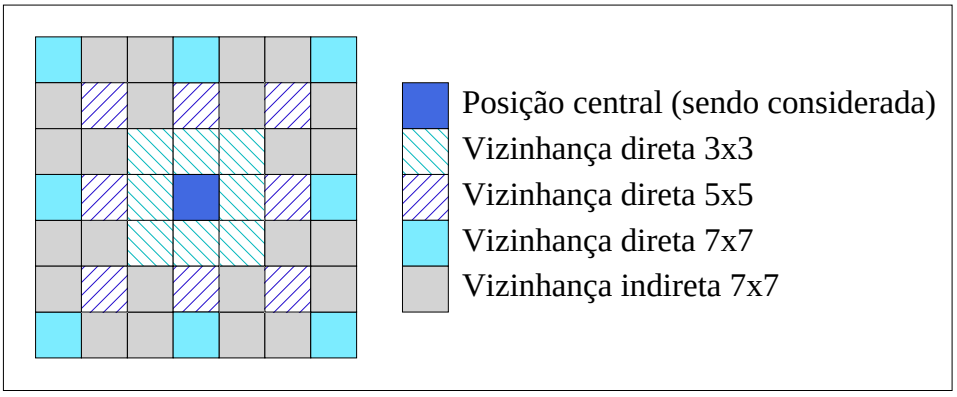


Exercício 10.37 ★ ★ ★ ★ ★

Também é possível adaptar o jogo de *Gomoku* (exercício 10.32) para que o computador possa jogar com o usuário, usando alguma inteligência para decidir onde posicionar suas peças. A heurística para que o computador decida qual posição é melhor para jogar uma peça pode ser:

- Menos dois pontos para cada peça do adversário que estiver na vizinhança direta 7×7 e mais dois pontos para cada peça do computador que estiver nesta vizinhança,
- Menos quatro pontos para cada peça do adversário que estiver na vizinhança direta 5×5 e mais quatro pontos para cada peça do computador que estiver nesta vizinhança,
- Menos oito pontos para cada peça do adversário que estiver na vizinhança direta 3×3 e mais oito pontos para cada peça do computador que estiver nesta vizinhança,
- Menos um ponto para cada peça do adversário que estiver na vizinhança indireta 7×7 e mais um ponto para cada peça do computador que estiver nesta vizinhança.

A figura abaixo mostra, para uma determinada posição, quais são as posições que correspondem às vizinhanças.
Usando o exercício 10.32 como base, escreva um método `jogaComputador` que calcule a melhor posição para jogo e efetue a jogada. Outras partes da classe deverão ser reescritas, por exemplo, para permitir que o usuário decida se vai jogar com as peças pretas ou brancas e quem será o primeiro a jogar.
Dica: Veja também o enunciado do exercício 10.36.



Capítulo 11

Classes para manipulação de strings

Strings ou cadeias de caracteres são estruturas bastante usadas em classes e programas de computador – sempre que for necessário representar informações textuais podemos usar strings. Em muitas aplicações é necessário processar estas strings para separá-las em caracteres individuais, identificar padrões, extrair trechos das strings, concatenar e formatar, etc.

Strings são representadas por uma classe específica em Java (`String`). Este capítulo apresenta esta classe, suas características e principais métodos para processamento da string, dos caracteres que a compõem e de trechos da string. O capítulo apresenta também a classe `StringBuilder` que pode ser usada para representar strings modificáveis de forma mais eficiente.

11.1 Exercícios do Capítulo 11

Exercício 11.1 ★

Identifique e explique o(s) erro(s) na classe abaixo.

```
1 public class MudaString
2   {
3   public static void main(String[] argumentos)
4     {
5     String nome = "Dan Gusfield";
6     nome.charAt(3) = '+';
7     System.out.println(nome);
8     }
9   }
```

Exercício 11.2 ★

Quais serão os valores retornados pelo método `length` quando aplicado às strings "`cadeia de caracteres`", "`\n\n`" e "`"`"?

A. 20, 2 e zero, respectivamente.
B. 19, 2 e um, respectivamente.
C. 18, 2 e zero, respectivamente.
D. 19, 4 e `null`, respectivamente.
E. 18, 4 e zero, respectivamente.

Exercício 11.3 ★

Quais das seguintes operações com strings abaixo retornarão o valor booleano `true`?

A. `"Tremblay".startsWith("T")`
B. `"Tremblay".endsWith("Y")`
C. `"Tremblay".toLowerCase().startsWith("tre")`
D. `"Tremblay".startsWith("tre".toUpperCase())`
E. `"Tremblay".trim().startsWith("re")`

Exercício 11.4 ★

Considerando a string `palavra` valendo `"autodeterminação"`, quais serão os resultados das expressões `palavra.substring(11)`, `palavra.substring(6,13)` e `palavra.substring(4,9)`?

A. `"ação"`, `"ermina"` e `"eter"`, respectivamente.
B. `"inação"`, `"erminaç"` e `"eterm"`, respectivamente.
C. `"nação"`, `"termina"` e `"deter"`, respectivamente.
D. `"inação"`, `"erminaç"` e `"eterm"`, respectivamente.
E. `"nação"`, `"ermina"` e `"eter"`, respectivamente.
F. `"ação"`, `"termina"` e `"deter"`, respectivamente.

Exercício 11.5 ★

Modifique a classe `JogoDaForca` (listagem 11.2 no livro) para que o construtor filtre a string recebida, garantindo que a palavra a ser adivinhada não terá caracteres maiúsculos (ou seja, convertendo todos os caracteres da string para minúsculos).

Exercício 11.6 ★ ★

Escreva uma classe em Java que represente o nome completo de uma pessoa, composto de três strings (nome próprio, nome do meio e nome da família). Escreva nessa classe o método `rubrica` que retorna somente as iniciais do nome completo em caracteres minúsculos, e o método `assinatura` que retorna as iniciais dos nomes próprio e do meio (com pontos) e o nome de família completo. Por exemplo, se o nome da pessoa representado por essa classe for `"Richard Langton Gregory"`, o método `rubrica` deve retornar `"rlg"` e o método `assinatura` deve retornar `"R.L.Gregory"`. Para facilitar, considere armazenar os três nomes em strings separadas.

Exercício 11.7 ★ ★

Crie uma classe `StringUtils` que contenha vários métodos estáticos para processamento de strings. Crie nesta classe um método estático `desacentua` que recebe como argumento uma string e que substitua todos os caracteres acentuados desta string por caracteres não-acentuados correspondentes. Por exemplo, se a string `"Nação"` for passada como argumento, esse método deverá retornar `"Nacao"`. O método deve considerar maiúsculas e minúsculas como sendo diferentes. *Dica:* Várias chamadas ao método `replace`, em cascata, poderão resolver o problema.

Exercício 11.8 ★ ★

Crie, na classe `StringUtils` (exercício 11.7), o método `alinhaÀDireita`, que recebe como argumentos uma string e um valor numérico, e completa a string com espaços à esquerda até que o comprimento da string fique igual ao valor numérico passado, retornando a string modificada. Escreva também o método `alinhaÀEsquerda`, que faz o mesmo mas adicionando espaços à direita. Se o comprimento da string passada já for maior que o valor passado como argumento, o método deve retornar a string inalterada.

Exercício 11.9 ★ ★

Crie, na classe `StringUtils` (exercício 11.7), o método `replica`, que recebe como argumentos uma string e um valor inteiro, e retorna uma string composta de várias repetições da string passada como argumento, onde o número de repetições deve ser o número passado como argumento. Por exemplo, se os argumentos para esse método forem a string `"Ha!"` e o valor 3, o método deverá retornar `"Ha!Ha!Ha!"`.

Exercício 11.10 ★ ★

Escreva para a classe `StringUtils` (exercício 11.7) um método estático `conta` que receba como argumentos uma string e um caracter, e retorne um inteiro correspondente ao número de ocorrências do caracter na string passados como argumentos.

Exercício 11.11 ★ ★

Escreva na classe `StringUtils` (veja exercício 11.7) um método `reverte` que reverta a ordem dos caracteres de uma string passada como argumento e retorne a string revertida. Um exemplo: se a string `"Java"` for passada para esse método, ele deve retornar a string `"avaJ"`. *Dica:* Use um laço `for` ou `while` e o método `charAt`, e crie uma nova string que receberá os caracteres na ordem invertida. Não use mecanismos da classe `StringBuffer` ou `StringBuilder`.

Exercício 11.12 ★ ★

Escreva na classe `StringUtils` vários métodos sobrecarregados (veja seção 4.3 no livro) para comparação entre strings, baseados nos métodos `equals` e `equalsIgnoreCase`, que recebam duas ou mais strings e retornem `true` se todas forem iguais e `false` se qualquer uma for diferente das outras. *Dica:* Estes métodos podem ser chamados em cascata, de forma que o método que compara três strings pode chamar o método que compara duas, e assim em diante.

Exercício 11.13 ★ ★

Escreva um método `quantasVezes` para a classe `StringUtils` que receba duas strings como argumentos e retorne o número de vezes que a segunda string aparece na primeira. Por exemplo, se a string `"recrearem"` e `"re"` forem passadas como argumentos, o método deverá retornar 3.

Exercício 11.14 ★ ★

Escreva um método `retiraVogais` na classe `StringUtils` que receba uma string como argumento e remova todas as vogais (maiúsculas e minúsculas) dessa string, retornando a string modificada como resultado.

Exercício 11.15 ★ ★

Os métodos `startsWith` e `endsWith` da classe `String` consideram caracteres maiúsculos e minúsculos como sendo diferentes: se a string `cidade` valer `"Rio de Janeiro"` o resultado de `cidade.startsWith("rio")` será `false`. Escreva dois métodos estáticos (`startsWithIgnoreCase` e `endsWithIgnoreCase`) na classe `StringUtils` que recebam duas strings como argumentos e que retornem `true` se a primeira string respectivamente começar ou terminar com a segunda, independentemente de estarem em maiúsculas ou minúsculas.

Exercício 11.16 ★ ★

Em um jogo de tabuleiro chamado *Palavras Cruzadas*, cada palavra formada por um jogador vale um certo número de pontos, que depende das letras usadas. O número de pontos para as letras do alfabeto é dado por:

- Para cada letra `'Q'` ou `'Z'` na palavra some 10 pontos.
- Para cada letra `'J'` ou `'X'` na palavra some 8 pontos.
- Para cada letra `'K'` na palavra some 5 pontos.
- Para cada letra `'F'`, `'H'`, `'V'`, `'W'` ou `'Y'` na palavra some 4 pontos.
- Para cada letra `'B'`, `'C'`, `'M'` ou `'P'` na palavra some 3 pontos.
- Para cada letra `'D'` ou `'G'` na palavra some 2 pontos.
- Para todas as outras letras some 1 ponto.

Caracteres que não sejam letras devem ser ignorados. Caracteres minúsculos devem ser convertidos para maiúsculos, e caracteres acentuados devem ser desacentuados. Por exemplo, o número de pontos para a palavra `"Java"` no jogo será $8+1+4+1=14$ pontos. Escreva uma classe `PalavrasCruzadas` em Java que contenha um método que receba uma string como argumento e retorne o número de pontos que esta string valeria no jogo.

Exercício 11.17 ★ ★

Modifique a classe `JogoDaForca` (listagem 11.2 no livro) para que o construtor filtre a string recebida, garantindo que a palavra a ser adivinhada não terá acentos. Use, para isso, o método `desacentua` da classe `StringUtils` (veja exercício 11.7).

Exercício 11.18 ★ ★ ★

Uma string é dita *palíndroma* se puder ser lida da esquerda para a direita ou da direita para a esquerda da mesma forma. As strings *"radar"*, *"asa"* e *"O breve verbo"* são palíndromas (desconsiderando os espaços e diferenças entre maiúsculas e minúsculas). Escreva dois métodos estáticos na classe `StringUtils` (exercício 11.7) que retornem `true` se uma string passada como argumento for palíndroma e `false` se não for. Um dos métodos deve ser mais estrito que o outro, e considerar espaços como caracteres; o outro não – como diferenciar os dois? *Dica:* Use o exercício 11.11 como base.

Exercício 11.19 ★ ★ ★

Comparação de strings também pode ser feita por similaridade de fonemas, e existem vários algoritmos que permitem a conversão de strings para esse tipo de comparação. Um dos algoritmos mais conhecidos (e antigos) é o chamado *soundex*, que reduz uma string a um código de quatro dígitos, de forma que strings que sejam foneticamente similares tenham códigos parecidos. O algoritmo usa o primeiro caracter da string como primeiro caracter do código, e converte os caracteres restantes da string para valores entre 1 e 6 de acordo com as seguintes regras:

- Caracteres acentuados devem ter os acentos removidos, e caracteres minúsculos devem ser convertidos para maiúsculos.
- Os caracteres `'B'`, `'P'`, `'F'` e `'V'` são convertidos para o dígito 1.
- Os caracteres `'C'`, `'Ç'`, `'S'`, `'G'`, `'J'`, `'K'`, `'Q'`, `'X'` e `'Z'` são convertidos para o dígito 2.
- Os caracteres `'D'` e `'T'` são convertidos para o dígito 3.
- O caracter `'L'` é convertido para o dígito 4.
- Os caracteres `'M'` e `'N'` são convertidos para o dígito 5.
- O caracter `'R'` é convertido para o dígito 6.
- As vogais e os caracteres `'Y'`, `'H'` e `'W'` não são codificados.
- Caracteres adjacentes iguais devem ser codificados como somente um (por exemplo, `"SS"` deve ser codificado como 2).
- O código final deve ter quatro caracteres, devendo ser completado com zeros se tiver menos que quatro caracteres ou truncado se tiver mais que quatro.

Como exemplo de codificação, as strings `"Javanês"` e `"Japonês"` têm o mesmo código (J152), a string `"assimétrico"` é codificada como A253 e a string `"Java"` é codificada como J100.

Crie, na classe `StringUtils` (exercício 11.7), um método estático que implemente o algoritmo *soundex*.

Exercício 11.20 ★ ★ ★

Escreva uma classe `StringDNA` que seja capaz de processar uma string de DNA. Strings de DNA são strings que são formadas exclusivamente pelos caracteres `'A'`, `'C'`, `'G'` e `'T'` – nenhum outro caracter é permitido. Essa classe deve encapsular uma instância da classe `String` e conter ao menos os seguintes métodos:

- Construtor, que recebe uma instância da classe `String` como argumento e copia somente os caracteres válidos para a string encapsulada (por exemplo, se a string passada for `"CATGATTAG"`, a string encapsulada deverá ser `"CATGATTAG"`, mas se a string passada for `"JAVA"`, a string encapsulada deverá ser `"AA"`).
- `toString`, que retorna a string encapsulada,
- `charAt`, que retorna o caracter na posição que for passada como argumento,
- `quantosA`, `quantosC`, `quantosG` e `quantosT`, que retornam, respectivamente, quantos caracteres `'A'`, `'C'`, `'G'` e `'T'` existem na string encapsulada,
- `length`, que retorna o comprimento da string encapsulada.

Exercício 11.21 ★ ★ ★

Escreva, para a classe `StringDNA` (exercício 11.20), um método `reversoComplementar` que retorne o reverso complementar da string encapsulada pela classe. O reverso complementar é calculado em dois passos: primeiramente trocamos cada caracter por seu complementar (isto é, `'A'` por `'T'` e vice-versa, `'C'` por `'G'` e vice-versa), depois revertemos a string de forma que o primeiro caracter seja o último e vice-versa. Por exemplo, se a string encapsulada for `"CTAGGATA"` o método deverá retornar `"TATCCTAG"`. O reverso complementar deve ser retornado como uma nova instância da própria classe `StringDNA`.

Exercício 11.22 ★ ★ ★

Crie, na classe `StringDNA` (veja exercício 11.20), o método `compara` que recebe uma instância da própria classe `StringDNA` para comparação e retorna um valor inteiro, calculado com o seguinte algoritmo:

- Coloque o valor zero em um acumulador,
- Para cada posição nas duas strings, compare os dois caracteres na posição,
- Se os caracteres forem exatamente iguais, some +3 pontos ao acumulador,
- Se os caracteres forem as combinações `'A'` e `'T'` **ou** `'T'` e `'A'`, some +1 ponto ao acumulador,
- Se os caracteres forem as combinações `'C'` e `'G'` **ou** `'G'` e `'C'`, some +1 ponto ao acumulador,
- Quando terminarem os caracteres de uma das strings, o valor acumulado será o valor a ser retornado pelo método.

Exemplo: Se a instância da classe `StringDNA` contiver a string `"ACATTG"` e para o método `compara` for passada como argumento a string `"ATTCCG"`, o valor a ser retornado será $3 + 0 + 1 + 0 + 0 + 3 = 7$.

Exercício 11.23 ★ ★ ★

Escreva a classe `Criptografia`, que conterá alguns métodos estáticos para codificação e decodificação de strings. Escreva nessa classe o método `codificaRot13`, que receberá uma string como argumento e retornará uma string codificada com o algoritmo *rot13*, que substitui cada caracter da string pelo valor do caracter mais treze, subtraindo vinte e seis caso o resultado seja maior que a última letra, de forma que `"abCde"` seja substituída por `"noPqr"`, `"kLmnoPq"` seja substituída por `"xYzabCd"`, e `"UVWxyz"` seja substituída por `"HIJklm"`. Somente os caracteres alfabéticos não-acentuados devem ser modificados. Por exemplo, se a string `"Revolução de 1930"` for passada como argumento para esse método, ele retornará `"Eribyhçãb qr 1930"`. Uma característica interessante do algoritmo *rot13* é que, se uma string codificada por ele for passada de novo pelo próprio algoritmo, a string original será retornada. Escreva também um método `decodificaRot13` que seja somente uma chamada para o método `codificaRot13`.

Exercício 11.24 ★ ★ ★

O *algoritmo de César* de criptografia de strings é uma versão melhorada do algoritmo *rot13* (veja exercício 11.23): o seu funcionamento é o mesmo, só que, em vez de substituir cada caracter por um caracter treze posições depois, o algoritmo de César recebe um valor chamado *chave*, e usa esse valor como o número de posições que devem ser puladas para a criptografia. Por exemplo, se a chave for 1, o algoritmo pulará uma posição ao codificar as letras, então se a string passada for `"Java"`, o resultado será `"Kbwb"`. O algoritmo de decodificação deve receber a mesma chave, só que deve substituir os caracteres da string por valores em posições anteriores.

Escreva um método estático `codificaCésar` na classe `Criptografia` que implemente o algoritmo de César, recebendo como argumentos uma string e uma chave (valor numérico) e retornando a string criptografada. Esse método deve considerar que **somente** as letras não-acentuadas devem ser criptografadas, as letras acentuadas, números, espaços e outros símbolos devem continuar como estão. Escreva também o método `decodificaCésar`.

Dica: Para simplificar o algoritmo, considere que o valor da chave só pode estar entre 1 e 25. Existem ao menos duas maneiras de implementar esse algoritmo.

Exercício 11.25 ★ ★ ★

Uma string contendo RNA é composta somente dos caracteres `'A'`, `'C'`, `'G'` e `'U'` – nenhum outro caracter é permitido. Escreva uma classe `StringRNA` baseada na classe `StringDNA` (exercício 11.20) e que implemente os mesmos métodos da classe `StringDNA`. Devemos usar herança? De que forma podemos reduzir possíveis problemas no desenvolvimento destas duas classes?

Exercício 11.26 ★ ★ ★

Escreva, para a classe `StringUtils`, o método `alfabetoCompleto` que recebe uma string como argumento e retorna outra string contendo o alfabeto completo da string passada como argumento. O alfabeto completo de uma string é o grupo de caracteres que aparece na string, sem repetições (podendo ser mostrado ordenado ou não). Por exemplo, o alfabeto completo de `"desencontradamente"` é `"desncotram"`, e o alfabeto completo de `"colina"` é a própria string `"colina"`.

Exercício 11.27 ★ ★ ★

O *algoritmo das pontas* de criptografia recebe uma string como argumento e produz uma outra string como resultado, e pode ser descrito da seguinte forma: enquanto a string de entrada contiver caracteres, remova o primeiro e o último caracteres da string de entrada e os coloque na string de saída. Dessa forma, se a string `"Programação em Java"` for entrada no algoritmo, este mostrará como saída a string `"ParvoagJr ammea çoã"`. A decodificação de uma string pode ser feita da seguinte forma: crie duas strings temporárias, e para cada par de caracteres extraídos da string codificada de entrada adicione o primeiro no fim da primeira string e o segundo no início da segunda string. A concatenação das duas strings é o resultado da decodificação.

Escreva, na classe `Criptografia`, os métodos estáticos `codificaPontas` e `decodificaPontas` para codificar e decodificar uma string usando esse algoritmo.

Exercício 11.28 ★ ★ ★

O *algoritmo das fatias* de criptografia usa como entrada uma string e um valor numérico, sendo que esse valor numérico (o número de fatias) deve ser bem menor que o tamanho da string. O algoritmo de codificação fatia a string de entrada, tomando caracteres de N em N posições, onde N é o número de fatias, formando N novas strings cada uma com o comprimento M onde M é o comprimento da string original dividido por N. As strings criadas assim são concatenadas, sendo o resultado da codificação da string original. Para decodificar uma string criptografada com esse algoritmo, é necessário ter o valor numérico.
Por exemplo, para criptografar a string "`Programação em Java`" (19 caracteres) usando 4 como número de fatias, o primeiro passo seria completar a string de forma que tenha um número de caracteres múltiplo de 4, para "`Programação em Java `" (20 caracteres), para simplificar o algoritmo. Fatiando essa string, pegando de quatro em quatro caracteres, obtemos quatro novas strings:

```
Prçea
raãmv
omo a
ga J
```

A string criptografada seria o resultado da concatenação dessas strings ou "`Prçearaãmvomo aga J `". Para decodificar essa string, basta repetir o processo de codificação mas usando M como o númeo de fatias, obtendo as strings

```
Prog
rama
ção
em J
ava
```

cuja concatenação resulta em "`Programação em Java `".
Escreva, na classe `Criptografia`, o método `codificaFatias` que recebe uma string e um valor numérico, codificando a string usando o valor e retornando a string criptografada. Escreva também o método `decodificaFatias` que deve fazer o processo reverso. *Dica:* O método `decodificaFatias` pode ser uma chamada para o método `codificaFatias` com o valor numérico adequado.

Exercício 11.29 ★ ★ ★

Escreva uma classe `CodigoMorse` com métodos estáticos que convertam strings de caracteres para strings em código morse e vice-versa. No código morse, caracteres são representados por pontos (correspondentes a um impulso elétrico curto) e traços (correspondentes a um impulso elétrico longo). Os caracteres básicos e seus correspondentes em código morse são mostrados abaixo:

a .–	b –...	c –.–.	d –..	e .	f ..–.
g ––.	h	i ..	j .–––	k –.–	l .–..
m ––	n –.	o –––	p .––.	q ––.–	r .–.
s ...	t –	u ..–	v ...–	w .––	x –..–
y –.––	z ––..			ponto .–.–.–	vírgula ––..––

A cada caracter que for codificado, um espaço deve ser adicionado à string de saída. O método para codificação deve considerar caracteres maiúsculos e minúsculos como sendo equivalentes, e ignorar outros símbolos. Exemplo: se a string `"Farfalhar"` for passada para o método de codificação, este deve retornar `"..–. .– .–. ..–. .– .–.. .... .– .–."`.

Dica: Para a decodificação de código morse para caracteres, use os espaços que devem existir na string codificada como delimitadores dos caracteres do código morse. Se algum código morse não tiver caracter correspondente (por exemplo, `"–––––"`), use o caracter `?` para saída.

Exercício 11.30 ★ ★ ★ ★

O *algoritmo de César* (veja o exercício 11.24) pode ser implementado de maneira mais complexa (e difícil de ser quebrada) se, em vez de uma única chave, várias forem usadas. O primeiro caracter da string será codificado com a primeira chave, o segundo caracter com a segunda chave etc. Quando as chaves acabarem, a primeira será reutilizada, até o final da string a ser codificada. A chave pode ser especificada por outra string, onde cada caracter tem um valor numérico correspondente, de forma que a string `"AEIY"` corresponde a quatro chaves (`1`, `5`, `9`, `25`). Dessa forma, se a string `"Programa"` fosse codificada com a chave `"aeb"` (correspondente aos valores 1, 5 e 2) o resultado seria `"Qwqhwcnf"` (`'P'` foi adicionada a 1 e o resultado é `'Q'`, `'r'` foi adicionada a 5 e o resultado é `'w'`, `'o'` foi adicionada a 2 e o resultado é `'q'`, `'g'` foi adicionada a 1 e o resultado é `'h'`, `'r'` foi adicionada a 5 e o resultado é `'w'` etc.) – note o uso cíclico dos valores da chave.

Escreva um método estático `codificaCésarMelhorado` na classe `Criptografia` (veja exercício 11.23) que implemente o algoritmo de César modificado, recebendo duas strings como argumento: a primeira será a string a ser criptografada e a segunda será a chave. Esse método deverá retornar outra instância da classe `String` correspondendo ao primeiro argumento, criptografado. Esse método deve considerar que **somente** as letras não-acentuadas devem ser criptografadas; as letras acentuadas, números, espaços e outros símbolos devem continuar como estão. Escreva também o método `decodificaCésarMelhorado`, que também recebe duas strings como argumentos e retorna a string decodificada.

Dica: Para simplificar o algoritmo, considere que o valor da chave só pode estar entre 1 e 26, ou seja, as letras `'A'` a `'Z'`. O algoritmo de codificação deve verificar se a string passada como chave é válida, ou seja, se ela contém somente caracteres maiúsculos na faixa prevista.

Exercício 11.31 ★ ★ ★ ★

Escreva, para a classe `StringUtils`, um método `quasePalíndroma`, que retorne `true` se uma string passada para ele for quase-palíndroma, isto é, tiver somente um par de caracteres que descaracterize a string como palíndroma. Por exemplo, `"acamada"` e `"mamaram"` são quase-palíndromas. Veja também o exercício 11.18.

Exercício 11.32 ★ ★ ★ ★

Aminoácidos são definidos por conjuntos de três caracteres em strings de RNA, sendo que cada aminoácido pode ter mais do que um conjunto de três caracteres correspondentes (*codons*). Existem vinte aminoácidos, e algumas combinações de três caracteres formam um *sinal de término*. Os vinte aminoácidos e o sinal de término, seus símbolos (entre parênteses) e as combinações correspondentes são:

- Ácido Aspártico (`D`): `GAU` e `GAC`
- Ácido Glutâmico (`E`): `GAA` e `GAG`
- Alanina (`A`): `GCU`, `GCC`, `GCA` e `GCG`
- Arginina (`R`): `CGU`, `CGC`, `CGA`, `CGG`, `AGA` e `AGG`
- Asparagina (`N`): `AAU` e `AAC`
- Cisteína (`C`): `UGU` e `UGC`
- Fenilalanina (`F`): `UUU` e `UUC`
- Glicina (`G`): `GGU`, `GGC`, `GGA` e `GGG`
- Glutamina (`Q`): `CAA` e `CAG`
- Histidina (`H`): `CAU` e `CAC`
- Isoleucina (`I`): `AUU`, `AUC` e `AUA`
- Leucina (`L`): `UUA`, `UUG`, `CUU`, `CUC`, `CUA` e `CUG`
- Lisina (`K`): `AAA` e `AAG`
- Metionina (`M`): `AUG`
- Prolina (`P`): `CCU`, `CCC`, `CCA` e `CCG`
- Serina (`S`): `AGU`, `AGC`, `UCU`, `UCC`, `UCA` e `UCG`
- Tirosina (`X`): `UAU` e `UAC`
- Treonina (`T`): `ACU`, `ACC`, `ACA` e `ACG`
- Triptofano (`W`): `UGG`
- Valina (`V`): `GUU`, `GUC`, `GUA` e `GUG`
- Sinais de término (`.`): `UAA`, `UAG` e `UGA`

Considerando a lista acima, escreva a classe `StringAminoAcidos`, que encapsule uma string composta somente de símbolos de aminoácidos. O construtor dessa classe deve receber como argumento uma instância da classe `StringRNA` (exercício 11.25) e transformar grupos de três em três caracteres para símbolos dos aminoácidos, armazenando estes na string encapsulada. Por exemplo, se a string encapsulada por uma instância da classe `StringRNA` fosse `"AUGGGUAAAGCCUGGUAG"` e esta string fosse passada como argumento para o construtor da classe `StringAminoAcidos`, a string encapsulada seria `"MGKAW."`. O método deve desconsiderar restos de strings que não formem três caracteres: uma string de oito caracteres corresponderá a dois aminoácidos e dois caracteres sobrarão, sendo descartados.
Dica: Existe mais de uma maneira de calcular o aminoácido a partir das trincas de caracteres. Qual é a mais simples?

Exercício 11.33 ★ ★ ★ ★

Duas sequências de aminoácidos podem ser comparadas entre si, caracter a caracter, para verificar o seu alinhamento. Em um alinhamento ideal, todos os caracteres são iguais nas duas sequências, mas frequentemente algumas divergências existem. Para avaliar a qualidade do alinhamento, um sistema de pontos é usado, que dá diferentes pesos ou notas para diferentes alinhamentos. Esses sistemas de pontos envolvem matrizes de substituição, que contêm valores (pesos) que serão usados quando o aminoácido da coluna da matriz for comparado com o aminoácido na linha da matriz. Uma dessas matrizes de substituição, chamada BLOSUM62, é mostrada abaixo.

	A	**R**	**N**	**D**	**C**	**Q**	**E**	**G**	**H**	**I**	**L**	**K**	**M**	**F**	**P**	**S**	**T**	**W**	**Y**	**V**
A	+4	−1	−2	−2	0	−1	−1	0	−2	−1	−1	−1	−1	−2	−1	+1	0	−3	−2	0
R	−1	+5	0	−2	−3	+1	0	−2	0	−3	−2	+2	−1	−3	−2	−1	−1	−3	−2	−3
N	−2	0	+6	+1	−3	0	0	0	+1	−3	−3	0	−2	−3	−2	+1	0	−4	−2	−3
D	−2	−2	+1	+6	−3	0	+2	−1	−1	−3	−4	−1	−3	−3	−1	0	−1	−4	−3	−3
C	0	−3	−3	−3	+9	−3	−4	−3	−3	−1	−1	−3	−1	−2	−3	−1	−1	−2	−2	−1
Q	−1	+1	0	0	−3	+5	+2	−2	0	−3	−2	+1	0	−3	−1	0	−1	−2	−1	−2
E	−1	0	0	+2	−4	+2	+5	−2	0	−3	−3	+1	−2	−3	−1	0	−1	−3	−2	−2
G	0	−2	0	−1	−3	−2	−2	+6	−2	−4	−4	−2	−3	−3	−2	0	−2	−2	−3	−3
H	−2	0	+1	−1	−3	0	0	−2	+8	−3	−3	−1	−2	−1	−2	−1	−2	−2	+2	−3
I	−1	−3	−3	−3	−1	−3	−3	−4	−3	+4	+2	−3	+1	0	−3	−2	−1	−3	−1	+3
L	−1	−2	−3	−4	−1	−2	−3	−4	−3	+2	+4	−2	+2	0	−3	−2	−1	−2	−1	+1
K	−1	+2	0	−1	−3	+1	+1	−2	−1	−3	−2	+5	−1	−3	−1	0	−1	−3	−2	−2
M	−1	−1	−2	−3	−1	0	−2	−3	−2	+1	+2	−1	+5	0	−2	−1	−1	−1	−1	+1
F	−2	−3	−3	−3	−2	−3	−3	−3	−1	0	0	−3	0	+6	−4	−2	−2	+1	+3	−1
P	−1	−2	−2	−1	−3	−1	−1	−2	−2	−3	−3	−1	−2	−4	+7	−1	−1	−4	−3	−2
S	+1	−1	+1	0	−1	0	0	0	−1	−2	−2	0	−1	−2	−1	+4	+1	−3	−2	−2
T	0	−1	0	−1	−1	−1	−1	−2	−2	−1	−1	−1	−1	−2	−1	+1	+5	−2	−2	0
W	−3	−3	−4	−4	−2	−2	−3	−2	−2	−3	−2	−3	−1	+1	−4	−3	−2	+11	+2	−3
Y	−2	−2	−2	−3	−2	−1	−2	−3	+2	−1	−1	−2	−1	+3	−3	−2	−2	+2	+7	−1
V	0	−3	−3	−3	−1	−2	−2	−3	−3	+3	+1	−2	+1	−1	−2	−2	0	−3	−1	+4

Usando essa matriz, podemos calcular o alinhamento entre as sequências de aminoácidos "`TKVSRYV`" e "`TDVAYYL`" como sendo a soma dos coeficientes mostrados na matriz: $+5-1+4+1-2+7+1$, ou 15.

Escreva na classe `StringAminoAcidos` (exercício 11.32) um método `calculaAlinhamento` que receba como argumento outra instância da classe `StringAminoAcidos` e retorne o valor do alinhamento da string encapsulada com a passada como argumento.

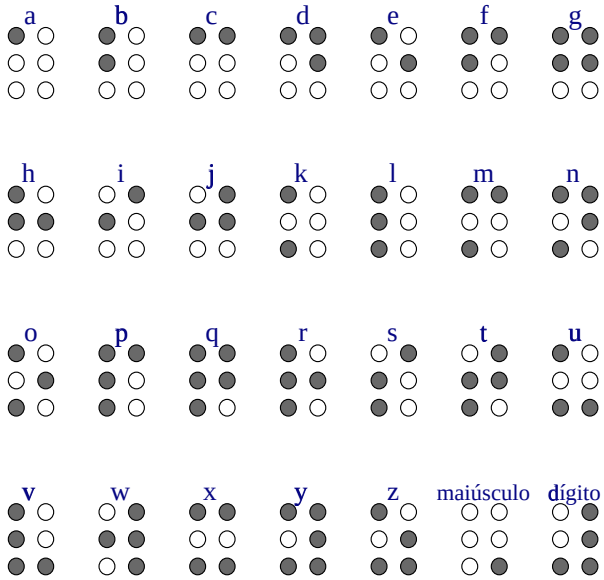
Exercício 11.34 ★ ★ ★ ★

Uma palavra ou frase é dita *anagrama* de outra se ela pode ser formada com os caracteres de outra, sem repetições, modificando as posições e inserindo espaços e pontuação à vontade, valendo também transformar maiúsculas em minúsculas e vice-versa. Dessa forma, "`manda jogar em vapor`" é anagrama de "`programando em java`". Anagramas são curiosidades lingüísticas, geralmente feitos para satirizar nomes de pessoas ou locais. Para que um anagrama seja considerado interessante, as palavras formadas devem ter algum significado. Escreva, para a classe `StringUtils`, um método `éAnagrama` que receba duas strings como argumentos e retorne `true` se uma é anagrama de outra.

Exercício 11.35 ★ ★ ★ ★

Escreva a classe `StringBraille` em Java. Essa classe deve representar internamente uma string e ser capaz de imprimi-la no alfabeto Braille, devendo ter ao menos o construtor (que recebe uma string a ser encapsulada, como argumento) e o método `toString` que imprimirá a string encapsulada em Braille.

A figura abaixo mostra o alfabeto Braille simplificado, onde um círculo preenchido significa uma marca em relevo no papel. Cada letra maiúscula que aparecer no texto deve ser precedida pelo caracter `maiúsculo` do alfabeto Braille. Cada dígito que aparecer no texto deve ser precedido pelo caracter `dígito` do alfabeto Braille. No caso dos dígitos, os caracteres Braille correspondentes a `'a'`, `'b'`, `'c'` ... `'i'`, `'j'` são usados para representar os dígitos `'1'`, `'2'`, `'3'` ... `'9'`, `'0'`. Para simplificar, considere que as strings a serem convertidas não contêm acentos nem símbolos, e que um espaço em Braille pode ser representado por um caracter em Braille sem nenhuma marca em relevo.



A saída do programa pode ser feita usando os caracteres de texto `'X'` para representar uma marca em relevo, `'.'` para representar uma posição onde não há marca em relevo, e o espaço para separar uma letra do alfabeto Braille de outra. Assim, se a string `"Java 123"` for entrada, a saída deverá ser:

```
.. .X X. X. X. .. .X X. .X X. .X XX
.. XX .. X. .. .. .X .. .X X. .X ..
.X .. .. XX .. .. XX .. XX .. XX ..
```

Dica: O método `toString` dessa classe deve criar três strings na saída, cada uma com uma "linha" de pontos dos caracteres Braille. O comprimento dessas três strings é igual, mas deve ser calculado verificando-se se a string encapsulada tem caracteres maiúsculos e dígitos.

Exercício 11.36 ★ ★ ★ ★

Várias sequências de aminoácidos podem ser comparadas para obtenção de um consenso, que pode ser calculado como o caracter que mais aparece em uma determinada posição. Por exemplo, considerando as seguintes sequências de caracteres

```
FNTXSPRNCDE
FCXTSRNRPDE
NNTXSRPNCCE
FNTXSPXRNDE
```

o consenso seria calculado para cada posição como sendo o caracter mais frequente, e o resultado seria a string `"FNTXS???CDE"`. Como em três posições não houve consenso (houve empate entre os caracteres mais frequentes), consideramos o caracter na posição como sendo igual a `'?'`.

Escreva na classe `StringAminoAcidos` (exercício 11.32) vários métodos `calculaConsenso` sobrecarregados, que recebam como argumentos outras instâncias da classe `StringAminoAcidos` e retornem o valor do consenso da string encapsulada com as passadas como argumentos.

Exercício 11.37 ★ ★ ★ ★

Escreva, para a classe `StringBraille` (exercício 11.35), um método `decodifica` que recebe três strings contendo os caracteres 'X', '.' e espaço e decodifique a mensagem em Braille contida nessas strings para uma string comum. Por exemplo, se as três strings passadas forem respectivamente:

`".. .X X. X. X. .. .X X. .X X. .X XX"`,
`".. XX .. X. .. .. .X .. .X X. .X .."` e
`".X .. .. XX .. .. XX .. XX .. XX .."`

o método deverá retornar a string `"Java 123"`. Veja o exemplo dado no exercício 11.35.

Exercício 11.38 ★ ★ ★ ★ ★

Escreva uma classe `JogoSegueLetras` que encapsule array bidimensional de caracteres. Esse array pode ser passado como argumento para o construtor da classe ou criado de outra forma. Para simplificar, considere que os caracteres armazenados serão todos maiúsculos (escreva código no construtor que garanta isso). Escreva para essa classe um método `existe` que receba uma string como argumento e que retorne um valor inteiro.

Uma string existe no array se é possível criar essa string navegando-se no array, um caracter de cada vez, sendo que de um caracter só se pode ir para o próximo se este for vizinho do caracter anterior. Se a string for encontrada, o valor retornado será o comprimento da string. Se a string não for encontrada, o valor retornado deverá ser menos duas vezes o comprimento da string. Por exemplo, se o array encapsulado for o mostrado abaixo, as palavras `"TIRO"`, `"DELTA"`, `"TALENTO"` e `"MAJORITARIAMENTE"` poderão ser achadas no array, e deverão retornar os valores 4, 5, 7 e 16, respectivamente, mas a palavra `"DESPROPORCIONADAMENTE"` deverá retornar o valor -42.

```
IRO
TAJ
LMD
SEO
RNI
OTE
```

Exercício 11.39 ★ ★ ★ ★ ★

Escreva uma classe `JogoLocalizaPalavras` que encapsule um array bidimensional de caracteres. Esse array pode ser passado como argumento para o construtor da classe ou criado de outra forma. Para simplificar, considere que os caracteres armazenados serão todos maiúsculos (escreva código no construtor que garanta isso). Escreva para essa classe um método `localiza` que receba uma string como argumento e retorne `true` caso essa string possa ser localizada dentro do array de caracteres, em qualquer posição e em qualquer orientação. Dessa forma, caso o array encapsulado seja como o mostrado abaixo, as palavras "`LOCALIZAR`", "`TENTATIVA`", "`BAIXA`" e "`TESTE`" poderiam ser localizadas no array.

```
ABLNHEHLLTBQJFRGQH
KJUTWRAZILACOLVMNJ
FEOGEQHTLOIDFMBAOQ
RWBNUSGEVIXOIOXGUZ
BRDARGTENTATIVAYJK
EARHSOWESLFVCDPZJQ
WECSWATLXBMTLCDPNI
```

Capítulo 12

Coleções de objetos

Arrays permitem a referência de várias instâncias ou valores como um conjunto, possibilitando representação de vários valores de forma flexível em classes e aplicações, mas com restrições à forma de representação e tipos que podem ser representados. Algumas classes implementam coleções de objetos de diversos tipos e com diversas características para representação e processamento – é possível representar conjuntos, listas e mapas ou arrays associativos de forma simples e flexível em Java.

Este capítulo apresenta as principais classes e interfaces para representação de coleções em Java, assim como conceitos de tipos genéricos, algoritmos e comparadores utilizados em coleções de objetos.

12.1 Exercícios do Capítulo 12

Exercício 12.1 ★

Cite duas diferenças entre arrays e instâncias da classe `ArrayList`.

Exercício 12.2 ★

Cite duas diferenças entre as classes `ArrayList` e `LinkedList`.

Exercício 12.3 ★

No método `main` da classe `UsaAvaliadorDeExpressoes` (listagem 12.18 no livro), a expressão `"9 5 1 +"` foi avaliada incorretamente como sendo igual a 6. Explique.

Exercício 12.4 ★

Ao final da execução do método `main` na classe abaixo, qual das opções a seguir será verdadeira?

A. A lista a conterá `["um", "três", "um"]`.
B. A lista a conterá `["um", "três"]`.
C. A lista a conterá `["dois", "quatro"]`.
D. A lista a conterá `["um", "dois", "quatro"]`.
E. A lista a estará vazia.

Exercício 12.5 ★

Ao final da execução do método `main` na classe abaixo, qual das opções a seguir será verdadeira?

A. A lista a conterá `["um", "três", "um"]`.
B. A lista a conterá `["um", "três"]`.
C. A lista a conterá `["dois", "quatro"]`.
D. A lista a conterá `["um", "dois", "quatro"]`.
E. A lista a estará vazia.

Exercício 12.6 ★

Ao final da execução do método `main` na classe abaixo, quais das opções a seguir serão verdadeiras?

```
1  import java.util.*;
2  public class TesteLista1
3    {
4    public static void main(String[] argumentos)
5      {
6      List<String> l1 = new LinkedList<String>();
7      l1.add("Alanina"); l1.add("Leucina"); l1.add("Alanina"); l1.add("Triptofano");
8      List<String> l2 = new LinkedList<String>();
9      l2.add("Alanina"); l2.add("Leucina"); l2.add("Alanina"); l2.add("Alanina");
10     }
11   }
```

A. O resultado de `l1.containsAll(l2)` será `true`.
B. O resultado de `l2.containsAll(l1)` será `true`.
C. O resultado de `l1.containsAll(l1)` será `true`.

Exercício 12.7 ★ ★

Usando a classe `OperacoesComConjuntos` (listagem 12.4 no livro) como base, calcule e imprima o conjunto de todos os autores não-atletas.

Exercício 12.8 ★ ★

Usando a classe `OperacoesComConjuntos` (listagem 12.4 no livro) como base, calcule e imprima o conjunto de todos os autores que são nadadores **ou** casados.

Exercício 12.9 ★ ★

Escreva, para a classe `ConjuntoDePalavras` (listagem 12.5 no livro), um método `toString` que retorne as palavras da lista encapsulada, concatenadas e separadas por espaços.

Exercício 12.10 ★ ★

Escreva, para a classe `ConjuntoDePalavras` (listagem 12.5 no livro), um método `remove` que receba uma string como argumento e que remova este argumento da lista de palavras encapsulada na classe.

Exercício 12.11 ★ ★

Escreva para a classe `ConjuntoDePalavras` (listagem 12.5 no livro) um método `contémTodas` que receba como argumento uma string contendo várias palavras separadas por espaços e que retorne `true` se cada uma das palavras existir na lista encapsulada na classe.

Exercício 12.12 ★ ★

Escreva para a classe `ConjuntoDePalavras` (listagem 12.5 no livro) uma versão sobrecarregada do método `contémTodas` que receba como argumento uma outra instância da classe `ConjuntoDePalavras` e que retorne `true` se cada uma dos elementos do array existir na lista encapsulada de palavras. Veja também o exercício 12.11. *Dica:* A solução desse exercício pode ser implementada com operações sobre os conjuntos encapsulados pelas instâncias – não é necessário usar laços `for`.

Exercício 12.13 ★ ★

Escreva, para a classe `ConjuntoDePalavras` (listagem 12.5 no livro) um método `removeTodas` que receba uma lista de palavras (na forma de outra instância de `ConjuntoDePalavras` e que remova todas as strings do array da lista de palavras encapsulada na classe. Veja também o exercício 12.12.

Exercício 12.14 ★ ★

Escreva para a classe `ConjuntoDePalavras` (listagem 12.5 no livro) uma versão sobrecarregada do método `contémTodas` que receba como argumento um array de strings e que retorne `true` se cada uma dos elementos do array existir na lista encapsulada de palavras. Veja também o exercício 12.11.

Exercício 12.15 ★ ★

Escreva, para a classe `ConjuntoDePalavras` (listagem 12.5 no livro), um método `removeTodas` que receba como argumento outra instância da classe `ConjuntoDePalavras` e que remova todas as palavras da lista passada como argumento da lista de palavras encapsuladas. Veja também o exercício 12.13. *Dica:* A solução desse exercício pode ser implementada com operações sobre os conjuntos.

Exercício 12.16 ★ ★

Escreva a classe `ListaDePalavras` que represente uma lista de palavras (strings), de forma que seja possível representar palavras repetidas na lista. As palavras podem ser mantidas em qualquer ordem. Veja a classe `ConjuntoDePalavras` (listagem 12.5 no livro).

Exercício 12.17 ★ ★

Escreva para a classe `ListaDePalavras` (exercício 12.16) um método `conta` que recebe como argumento uma string e retorna o número de ocorrências desta string na lista encapsulada (retornando zero caso a palavra não exista na lista).

Exercício 12.18 ★ ★

Escreva, para a classe `ArrayEsparsoDeDoubles` (listagem 12.21 no livro) um método `éDefinido` que recebe um valor do tipo `long` como argumento e retorna `true` se existe um valor definido para o valor passado como argumento (isto é, se existe um valor cujo índice é o passado como argumento).

Exercício 12.19 ★ ★

Escreva para a classe `ArrayEsparsoDeDoubles` (listagem 12.21 no livro) um método `menor` que retorne o menor valor existente no array esparso.

Exercício 12.20 ★ ★

Escreva para a classe `ArrayEsparsoDeDoubles` (listagem 12.21 no livro) um método `maior` que retorne o maior valor existente no array esparso.

Exercício 12.21 ★ ★

Escreva, para a classe `ArrayEsparsoDeDoubles` (listagem 12.21 no livro) um método `total` que calcule e retorne o total dos valores presentes no array.

Exercício 12.22 ★ ★

Escreva e demonstre um método estático em uma classe qualquer que elimine todos os itens repetidos de uma lista. *Dica:* Existe uma maneira bem simples de resolver esse problema usando construtores das classes mostradas neste capítulo.

Exercício 12.23 ★ ★

Escreva uma classe `MaquinaDeKaraoke` que encapsule o comportamento básico simulado de uma máquina de karaoke. Instâncias desta classe devem representar uma fila de músicas que serão tocadas na ordem que foram entradas na máquina (a primeira música a ser tocada deverá ter sido a primeira a ser entrada na fila). Músicas podem ser representadas pelos seus títulos, como strings.
Escreva para esta classe métodos que permitem a adição de uma música na fila, listam a fila de músicas (na ordem que devem ser tocadas) e que simulem a execução de uma música (removendo-a do início da fila). Qual coleção deve ser usada para representar melhor os atributos desta classe?

Exercício 12.24 ★ ★

Escreva para a classe `MaquinaDeKaraoke` (exercício 12.23) um método `remove` que remova a primeira ocorrência da música passada como argumento para o método.

Exercício 12.25 ★ ★ ★

Modifique o método `remove` da classe `MaquinaDeKaraoke` (exercício 12.23) para que este remova a última ocorrência da música passada como argumento, em vez da primeira. Veja também o exercício 12.24

Exercício 12.26 ★ ★ ★

Escreva para a classe `MaquinaDeKaraoke` (exercício 12.23) um método `removeTodas` que remova todas as ocorrências da música passada como argumento.

Exercício 12.27 ★ ★ ★

Escreva para a classe `MaquinaDeKaraoke` (exercício 12.23) dois métodos, `adia` e `adianta`, que recebam um nome de música como argumento e que, respectivamente, adiaem ou adiantem a execução daquela música em uma posição, modificando a sua posição dentro da fila. Esses métodos não devem fazer nada se a música passada como argumento não existir na fila, e devem tomar cuidado para não adiantarem a primeira música nem adiarem a última.

Exercício 12.28 ★ ★ ★

Escreva, para a classe `ConjuntoDePalavras` (listagem 12.5 no livro) um método `interseção` que receba como argumento uma outra instância de `ConjuntoDePalavras` e que retorne uma **nova** instância de `ConjuntoDePalavras` contendo a interseção da lista de palavras encapsulada com a lista passada como argumento, isto é, a lista de palavras que ocorrem nas duas instâncias. Veja também o exercício 12.12.

Exercício 12.29 ★ ★ ★

Escreva, para a classe `ArrayEsparsoDeDoubles` (listagem 12.21 no livro) um método `soma` que receba como argumento uma outra instância de `ArrayEsparsoDeDoubles` e que retorne uma **nova** instância da mesma classe contendo a soma ou do array esparso encapsulado com o passado como argumento. Se somente um dos dois arrays tiver um valor em uma determinada posição os valores devem ser replicados no array resultante. Se os dois arrays tiverem valores em uma mesma posição, os valores devem ser somados.
Por exemplo, a soma dos arrays `(0->3 17->9 1200->5 3300->8)` e `(0->2 18->7 1201->3 3300->2)` deve ser `(0->5 17->9 18->7 1200->5 1201->3 3300->10)`.

Exercício 12.30 ★ ★ ★

Escreva, para a classe `ArrayEsparsoDeDoubles` (listagem 12.21 no livro) um método `multiplica` que receba como argumento uma outra instância de `ArrayEsparsoDeDoubles` e que retorne uma **nova** instância da mesma classe contendo a multiplicação dos valores existentes nos arrays. *Dica:* use o exercício 12.29 como base.

Exercício 12.31 ★ ★ ★

Escreva uma classe `ContadorDePalavras` baseada na classe `ConjuntoDePalavras` que, além de armazenar palavras, armazene também quantas vezes uma palavra foi armazenada. Escreva métodos para essa classe que recuperem o número de vezes que uma palavra foi armazenada ou zero se ela não tiver sido armazenada. *Dica:* Use um mapa.

Exercício 12.32 ★ ★ ★

Usando as classes que demonstram a ordenação customizada de uma turma de alunos (listagens 12.24 a 12.29) escreva e modifique classes existentes para que seja possível listar os alunos ordenados pela idade (menor para maior).

Exercício 12.33 ★ ★ ★

Usando as classes que demonstram a ordenação customizada de uma turma de alunos (listagens 12.24 a 12.29) escreva e modifique classes existentes que listem os alunos ordenados pelo número de disciplinas que já foram cursadas (mais disciplinas para menos disciplinas).

Exercício 12.34 ★ ★ ★

Considere a classe `ArrayEsparsoDeDoubles` (listagem 12.21 no livro), que representa somente os valores existentes em um array, usando inteiros como chaves em um mapa para representar os índices destes valores. Usando esta classe como exemplo, escreva a classe `MatrizEsparsaDeDoubles`, que use dois valores inteiros para representar o índice em duas dimensões (linha e coluna) de elementos da matriz. *Dica*: um mapa só permite usar uma chave e um valor em suas entradas; será necessário compor estes dois valores inteiros em uma classe que representa a coordenada do elemento da matriz.
Crie nesta classe o construtor e os métodos `set` e `get`.

Exercício 12.35 ★ ★ ★ ★

Crie na classe `MatrizEsparsaDeDoubles` (exercício 12.34) um método `éIgual` que receba outra instância da classe como argumento e retorne `true` se os valores e índices na matriz encapsulada e na passada como argumento forem iguais.

Exercício 12.36 ★ ★ ★ ★

Crie na classe `MatrizEsparsaDeDoubles` (exercício 12.34) um método `criaTransposta` que não recebe argumentos e que retorna uma nova instância de `MatrizEsparsaDeDoubles` correspondente à transposta da original. *Dica:* pode ser útil modificar a classe que representa o índice dos elementos desta matriz.

Exercício 12.37 ★ ★ ★ ★

Crie na classe `MatrizEsparsaDeDoubles` (exercício 12.34) um método `soma` que recebe como argumento outra instância de `MatrizEsparsaDeDoubles`, soma as duas matrizes e retorna uma nova instância de `MatrizEsparsaDeDoubles`. A soma de duas matrizes esparsas A e B resultando em C é feita da seguinte forma:

- Se o elemento existe na matriz A mas não na B o valor será copiado de A para C.
- Se o elemento existe na matriz B mas não na A o valor será copiado de B para C.
- Se o elemento existe nas duas matrizes a soma dos elementos em A e B será colocado na posição correspondente em C

Exercício 12.38 ★ ★ ★ ★

Escreva uma classe `ContadorDePalavras` que represente, ao mesmo tempo, um conjunto de palavras (sem permitir repetições) e um contador para as ocorrências de cada uma das palavras (veja a classe `ConjuntoDePalavras` na listagem 12.5 no livro). Escreva um método `adiciona` para esta classe que recebe como argumento uma palavra e executa um dos seguintes passos:

- Se a palavra não existir no conjunto encapsulado, ela deve ser adicionada, e o contador deve ser criado com o valor 1.
- Se a palavra existir no conjunto encapsulado, seu contador deve ser recuperado e acrescido de 1.

Escreva também métodos para mostrar o conteúdo de todo o conjunto e recuperar o contador para determinada palavra passada como argumento (que deve retornar zero se a palavra não existir no conjunto).
Assumindo que a ordem das palavras no conjunto é indiferente, qual é a coleção mais adequada para representar as palavras e seus contadores?

Exercício 12.39 ★ ★ ★ ★

Escreva, para a classe `ContadorDePalavras` (exercício 12.38), um método `remove` que recebe como argumento uma palavra e que execute um dos seguintes passos:

- Se a palavra existir no conjunto encapsulado e seu contador for maior que um, reduzir o contador em um.
- Se a palavra existir no conjunto encapsulado e seu contador for igual a um, remover a palavra do conjunto.
- Se a palavra não existir no conjunto encapsulado, o método não deverá fazer nada.